Semi-Automated Segmentation of 3D Medical Ultrasound Images

by

John David Quartararo

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Electrical Engineering

October 2008

APPROVED:

Dr. Peder C. Pedersen

Dr. Matthew Ward

Dr. David Cyganski

Dr. Fred J. Looft

Department of Electrical and Computer Engineering

Abstract

A level set-based segmentation procedure has been implemented to identify target object boundaries from 3D medical ultrasound images. Several test images (simulated, scanned phantoms, clinical) were subjected to various preprocessing methods and segmented. Two metrics of segmentation accuracy were used to compare the segmentation results to ground truth models and determine which preprocessing methods resulted in the best segmentations. It was found that by using an anisotropic diffusion filtering method to reduce speckle type noise with a 3D active contour segmentation routine using the level set method resulted in semi-automated segmentation on par with medical doctors hand-outlining the same images.

Acknowledgements

Several people have been essential to this work. I would like to thank Dr. Peder C. Pedersen for advising this work, sharing his extensive knowledge on the subject and always offering helpful suggestions, and Dr. David Cyganski and Dr. Matthew Ward for being a part of the thesis committee. Dr. Thomas Szabo lent his expertise on the subject of ultrasound many times and offered much help throughout the project. Matthew Rowan created the tissue-mimicking phantoms that were scanned in-lab and used as part of the test image set, and his help was appreciated very much. Dr. Aaron Fenster of the Robarts Institute was kind enough to supply three sets of clinical data for the test image set as well. I would like to express my appreciation to Joyoni Dey for her help with speckle reduction techniques and the level set segmentation. Joyoni also helped set up having two medical doctors hand-outline the clinical images supplied by Dr. Aaron Fenster. Dr. Mark Smyczynski and Dr. Saroj Bharitaya lent quite a few hours hand-segmenting many 2D images from the 3D volumes, and their contributions were integral for being able to determine the accuracy of the segmentations of the clinical image volumes.

This research work was supported under Contract No. DAMD17-03-2-0006, "Real-Time Troop Physiological Status Monitoring System Using a Common Wireless Network", from the *Telemedicine and Advanced Technology Research Center* (TATRC). The financial support is gratefully acknowledged.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This document describes the segmentation work done on the Mobile Ultrasound project within the Electrical and Computer Engineering Department [10] at Worcester Polytechnic Instutute [58] at the Ultrasound Research Laboratory [53] under Dr. Peder C. Pedersen with funding from TATRC (Telemedicine and Advanced Technology Research Center) [51]. Work is being done to design and build a mobile ultrasound system that allows for free hand scanning to capture 3D medical images of human patients. This system is small enough to be carried by a clinician into areas where other imaging technologies [X-ray, computer tomography (CT), magnetic resonance imaging (MRI)] are infeasible due to equipment size, such as in rural areas without immediate hospital access and in battlefield situations. This document presents the work done for the segmentation system, which detects boundaries of target objects that can be presented to the clinician as a 3D model.

Chapter 2 will describe what segmentation is and present some different methods for accomplishing segmentation with a brief literature review. To evaluate the segmentation routine, a collection of test images were gathered, which is described in Chapter 3. This includes the simulated images, ultrasound phantoms, and clinical prostate data. To improve segmentation accuracy, several preprocessing methods that were investigated are described

in Chapter 4. Active contours were used for the segmentation method. A previously investigated formulation for tracking active contours as well as the more powerful method (level sets) used for this work are presented in Chapter 5. In order to quantitatively evaluate segmentation accuracy, the generation of ground truth models and two different metrics of measuring segmentation error are described in Chapter 6. The software tools used throughout this work are discussed in Chapter 7. The results for segmentations of all the test images using the different preprocessing methods are given in Chapter 8. Finally, a discussion of the conclusions of this work are given in Chapter 9 and recommendations for future work are presented in Chapter 10.

Segmentation, with respect to imaging, is the process of identifying the boundaries of an object of interest in an image. For this work, using medical ultrasound images, the goal is to segment anatomical structures such as cysts, prostates, fluid volumes (internal bleeding), cardiac structures and other organs. Segmentation in useful in that it gives the clinicial additional diagnotic tools and the ability to view organs as a 3D model. This is a powerful step forward from traditional 2D B-mode ultrasound scans, and from having doctors hand-segment images. For instance, when a patient is bleeding internally and blood collects between the organs, the volume of fluid can be calculated which can help determine the severity of the trauma. Efficacy of treatment can also be evaluated quantitatively, such as tracking the change in size and shape of a cancerous tumor over time when radiation therapy is being used to treat it. In cardiac applications, the pulmonary ejection fraction ratio, which is the ratio of the volume of one ventricle when filled to the volume when contracted, can be used to diagnose heart problems. By using automated segmentation techniques, the volume in 3D of these two states can be determined fairly quickly and easily.

A major problem with ultrasound as compared to other imaging modalities such as CT or MRI is that it suffers from a relatively low signal to noise

ratio (SNR), and is plagued by heavy amounts of speckle noise, which is a multiplicative type of noise that arises from the contructive and destructive intereference of the received acoustic waves. Still other problems with ultrasound images are the low image contrast and shadowing that can obscure object boundaries. To reduce the speckle noise, several image preprocessing methods are explored as to their ability to improve segmentation accuracy by removing speckle noise without signifiacntly degrading the object boundaries.

A segmentation method using active contours and the level set method are presentated in this work. Different test images were gathered and preprocessed using the various speckle reduction methods, and then segmented. Using two different metrics of segmentation accuracy, the results are compared to ground truth models that represent where the true boundaries lie. The results are presented which determine the best preprocessing method of the investigated options and show that the level set method is well suited for 3D ultrasound image segmentation.

Chapter 2

Background

Image segmentation is the method of determining the boundaries of an object in an image. This might be finding the outline of a tree in a picture, human facial recognition, locating the boundaries of the heart in an MRI, CT, or ultrasound image, or locating an object from images captured using in infrared camera. There are several ways to perform segmentation, and this chapter gives an overview of some methods along with strengths of weaknesses of those methods for segmenting medical ultrasound images in 3D. Shown in Figure 2.1(a) is an example of a 2D ultrasound scan of a human prostate, and Figure 2.1(b) shows the segmented prostate (hand-outlined by a doctor) as the collection of white pixels, and the black pixels respresent non-prostate tissue. The prostate boundary is the boundary between white and black pixels in Figure 2.1(b). The goal of this work would be to identify the prostate boundary from Figure 2.1(a) by means of segmentation techniques and present a result similar to what is shown in Figure 2.1(b).

Clustering methods [47] are one class of methods for segmentating data. The goal is to separate a number of pattern vectors into subsets known as clusters. Each cluster is comprised of pattern vectors that are *similar* to each other by some metric. For image segmentation, these pattern vectors may consist of textural based information [18] extracted from a small neigh-

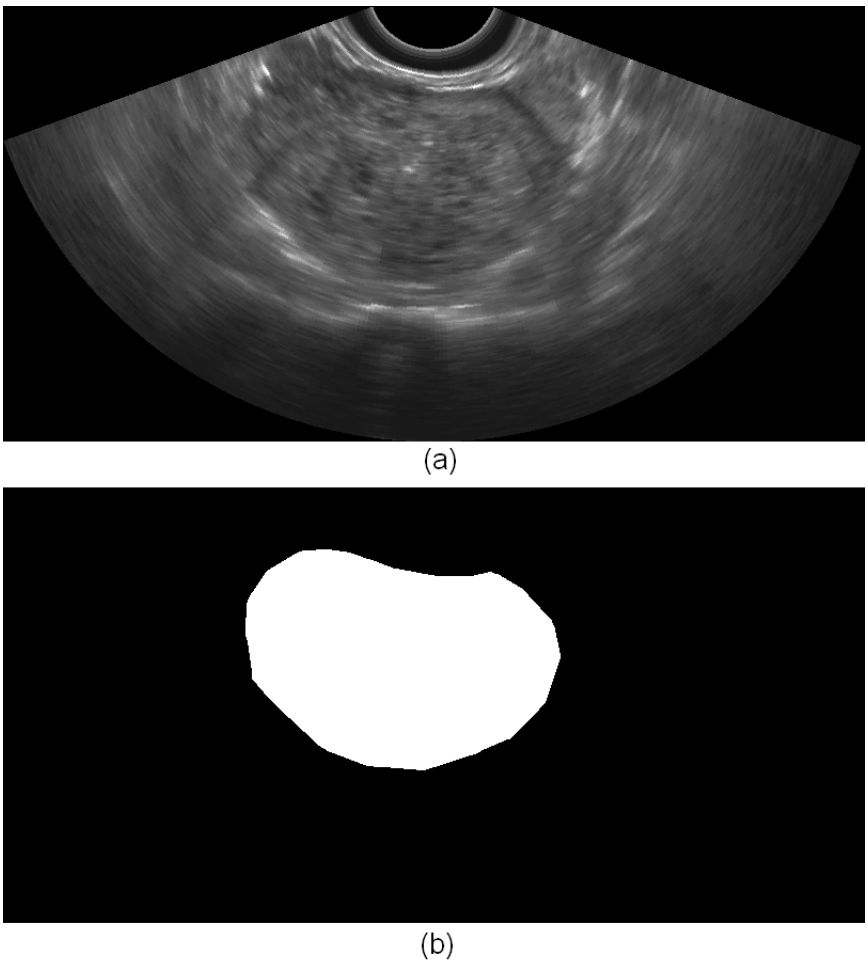

Figure 2.1: *Segmentation example. (a) 2D ultrasound scan of human prostate; (b) segmented prostate (hand-outlined by a doctor). White pixels represent prostate tissue, and the boundary between the white and black pixels represents the prostate boundary.*

borhood surrounding each pixel (voxel, in 3D), one pattern vector for each pixel. The assumption is that the pixels within a single object have similar qualities such as brightness and contrast. The K-means algorithm or the fuzzy K-means algorithm can be used to partition the pattern vectors into disjoint subsets (disjoint within the feature space). However, extracing the textural information is very computationally expensive. For ultrasound images, many different textural features would have to be used to differentiate objects from surrounding tissue, as basic features such as brightness and contrast are typically very similar across the ultrasound image.

Region growing methods [47] are another approach to segmentation. Starting from some manually placed of automatically generated seed point, neighboring pixels (voxels) are considered. Typically according to some statistical measure, those neighbor pixels are either added into the set of voxels representing the object, or rejected. These methods are typically computationally light, although there are problems with applying this to ultrasound images. If there is a weak boundary in the image, many times the region will grow out into the surrounding tissue. The Nearest Neighbor algorithm [40], initially explored in this work, has region growing components within it. It was not found to segment the images in our test set very reliably unless the object intensities were very distinct from the intensities of the surrounding tissue (which rarely happens).

Learning-based methods may incorporate textural and shape information, as well as *a priori* information about what type of object is being segmented. Learning-based methods include but are not limited to neural-net based implementations [29] and methods using kernel-state vector machines [48]. Here, a system is trained to recognize whether a region is part of the target of interest based on textural or shape based information. However, in order to train the system, many test images must be used with associated ground truth models. As this can sometimes take hundreds of test images before the system begins to perform reliably, this was deemed to be infea-

sible due to the the amount of 3D segmentations that would have to be acquired from human hand-outlining. For perspective, this work uses three 3D prostate volumes. Each one contains hundreds of scan planes. We had two doctors hand-outline every 10^{th} scan plane (to reduce the time burden), so each ground truth volume contains less than 100 2D segmentations. Each doctor took several hours to hand-ouline our three image volumes. Extending this to several hundred image volumes, this would be infeasible for doctors to carry out. Also, these learning-based methods require *significant* computation, so we did not pursue these methods for this segmentation system.

Active contour representations have been shown to be very useful for segmenting medical images, especially ultrasound images [49]. An active contour is a contour that can change shape and evolve with time. The idea is to create an active contour, and subject it to forces which push it in the direction of the target boundaries and stop it (converge) once the curve reaches the boundaries. This is an energy minimization problem and has been used successfully for segmentation. However, there are some problems with the traditional parametric method of tracking these contours as they evolve (see Section 5.1.2) that were overcome with the introduction in the 1980s of the level set method by Osher and Sethian [36], [46]. The level set method is a numerically stable method for tracking an active contour (See Section 5.2). This is only one application of level set theory, however. Level sets can also be used for noise removal in signals, modeling flame, smoke and fluids in computer generated animation, seismic analysis, and even in semiconductor manufacturing.

Caselles [5] and Malladi [32] extended Osher and Sethian's application of level set segmentation to include additional stopping forces based on the image gradient, adding edge based information into the segmentation. Kichenassamy [26] and Yezzi [60] added an advection force, a gradient field based off of the gradient of Malladi's stopping force, which helped to reduce leaks through weak target boundaries. There are some advantages in using

the level set method for segmenting 3D medical ultrasound images. For one, the active contour can be tracked directly in 3D (something that is difficult to do using parametric curves; further explanation is presented in Chapter 5). Another advantage is that by adding a surface tension force to the curve, contours can be prevented from "leaking" through weak boundaries, a very common scenario in ultrasound images. One disadvantage to the level set method is that it requires a lot of computational power - however, there are ways to significantly cut down on the required computation, so real-time performance is not infeasible [46].

In this work, the level set method of tracking an active contour was explored, as it has been shown in literature and with recent work to be a very reliable method for segmenting medical images that suffer from low contrast and low SNR.

Chapter 3

Ultrasound Image Sources

Many test images were used to evaluate the performance of the segmentation routines. These images were acquired three different ways: 1) simulation using the Field-II program, 2) scans of phantoms created in-lab, and 3) real human prostate scans from the Robarts Institute. The simulated images are useful because the locations, backscatter level and dimensions of the cyst targets are known exactly, and the accuracy of the segmentation is able to be determined very precisely. Scanned phantoms containing cylindrical inclusions were used to give a more realistic image set than the simulated volumes, and to evaluate the segmentation accuracy of these inclusions with known target dimensions. Finally, real 3D image volumes of human prostate scans were used to evaluate the segmentation performance on real clinical data. The terms used to characterize the properties (dimension in terms of numbers of voxels, and the physical spacing along each dimension of the rectilinear voxels) of the 3D image volumes are shown in Figure 3.1.

3.1 Simulated Images

The Field-II ultrasound simulation program [15] runs under Matlab and was developed by Dr. Jrgen Arendt Jensen of the Technical University of Den-

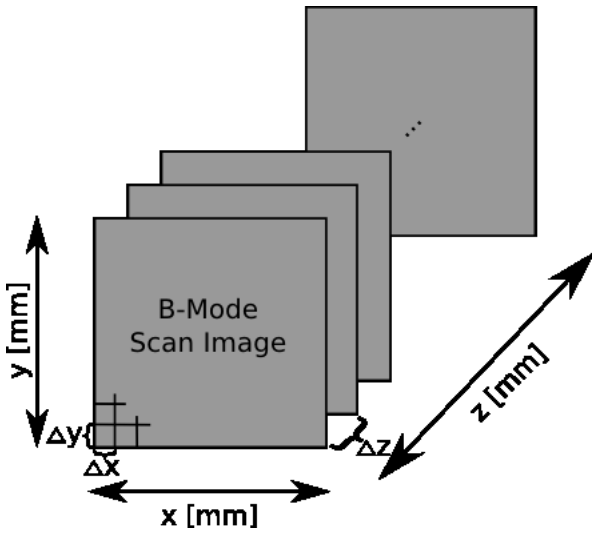

Figure 3.1: *Properties of 3D image volumes, comprised of many 2D B-Mode scan plane images with dimensionality information.*

mark. Field-II can be used to simulate both continuous and pulsed-wave linear ultrasound imaging using the Tupholme-Stepanishen method for the pressure field calculations [24]. Version 2.2 of Field-II for 32-bit Windows platforms, released April 2, 1998, was used with version 7.0.4.365 (R14) of Matlab, service pack 2, for 32-bit WinXP to generate the images.

3.1.1 Theoretical Background on Ultrasonic Transmission and Reflection

The Field-II simulation program works by calculating the spatial impulse function for the given transducer and aperture type. This spatial impulse function is then convolved with the transducer excitation function to find the received acoustic pressure. The transducer transfer function is used with the received acoustic pressure to find the voltage induced at the transducer from the received echo.

Rather than use an analytical function for the spatial impulse function

of known transducer geometries, Field II divides the transducer surface into small rectangles and uses the far-field approximation to sum the contributions of each planar surface to accommodate many transducer geometry types as well as arbitrary apodizations and excitations [25]. This is desirable, as closed-form solutions do not exist for some transducer geometries.

For a transducer mounted on an infinite, rigid baffle, the induced pressure field, $\vec{p}(\vec{r},t)$, can be calculated from the velocity potential as [28]:

$$\vec{p}(\vec{r},t) = \rho_0 \frac{\partial \Phi(\vec{r},t)}{\partial t} \tag{3.1}$$

where ρ_0 is the mean density and $\vec{r}$ is the vector pointing from the transducer to the field point of interest. This is a very useful tool as it relates the pressure at a field point to the velocity potential , and the velocity potential $\Phi(\vec{r},t)$ can be calculated from the particle velocity on the surface of the transducer. This particle velocity function $u_n(\vec{r_S},t)$, normal to the transducer surface S, is dependent on time and the specific point on the surface S, and is related to the velocity potential using (3.2). In this case, the vibrational amplitude is assumed to be constant and not a function of the position on S (hence not dependent on $\vec{r_S}$).

$$\vec{\Phi}(\vec{r},t) = \int_S \frac{u_n(t-|\vec{r}-\vec{r}_S|/c)}{2\pi|\vec{r}-\vec{r}_S|} dS = u_n(t) \star h(\vec{r},t) \tag{3.2}$$

Using (3.1) and (3.2), the pressure field at each point can be found from the transducer geometry and surface particle velocity. The term $|\vec{r}-\vec{r}_S|/c$ in (3.2) is the time delay for the wave to travel from the transducer to the field point at $\vec{r}$, where c is the speed of sound in the medium. The velocity potential can then be expressed as a time-domain convolution of the particle velocity function with $h(\vec{r},t)$, known as the spatial impulse response. The spatial impulse can be found by plugging in a Dirac delta function for the particle velocity:

$$\vec{h}_{unif}(r,t) = \int_S \frac{\delta(t - |\vec{r} - \vec{r}_S|/c)}{2\pi|\vec{r} - \vec{r}_S|} dS \tag{3.3}$$

If the transducer excitation signal has been apodized and the surface is no longer assumed to be vibrating uniformly, the particle velocity function $u_n(r,t)$ becomes a function of both time and the position on the transducer surface with the apodization function denoted $a(r)$. Thus the apodized spatial impulse is given by (3.4) and the approximation of that integral given by (3.5).

$$\vec{h}(\vec{r},t) = \int_S \vec{a}(\vec{r}) \frac{\delta(t - |\vec{r} - \vec{r}_S|/c)}{2\pi|\vec{r} - \vec{r}_S|} dS \tag{3.4}$$

$$\vec{h}(\vec{r},t) \approx \sum_S \vec{a}(\vec{r}) \frac{\delta(t - |\vec{r} - \vec{r}_S|/c)}{2\pi|\vec{r} - \vec{r}_S|} \tag{3.5}$$

Equation (3.1) can be rewritten in terms of the new expression for $\Phi(r,t)$ in (3.2) as:

$$p(\vec{r},t) = \rho_0 \frac{\partial \Phi(\vec{r},t)}{\partial t} = \rho_0 \frac{\partial (u_n(t) \star h(\vec{r},t))}{\partial t} = \rho_0 u_n(t) \star \frac{\partial h(\vec{r},t)}{\partial t} \tag{3.6}$$

The incident pressure formed by the acoustic pulse is now known, and the theorem of Acoustic Reciprocity is invoked to find the pulse reflected back at the transducer. The theorem of Acoustic Reciprocity states that if the locations of the source and field point are interchanged, they will have the same effect on each other. Using this, we can solve for the received pressure at the transducer (and eventually the induced voltage) by treating the field point as a small source (small rectangular tile) which now emits a perfectly reflected copy of the original incident pressure (The field point is characterized as having infinite acoustic impedance, hence the perfect reflection). The surface velocity of the source (tile with area dA) is dependent on the pressure at that point, as shown in (3.7), which creates the velocity potential due to the

returning acoustic wave, given in (3.8):

$$u_s(\vec{r},t) = -\frac{p_i(\vec{r},t)}{\rho_0 c} \tag{3.7}$$

$$\Phi(\vec{r},t) = \frac{u_S(\vec{r},t-|\vec{r}-\vec{r}_S|/c)}{4\pi|\vec{r}-\vec{r}_S|}dA = -\frac{p_i(\vec{r},t-|\vec{r}-\vec{r}_S|/c)}{4\pi|\vec{r}-\vec{r}_S|\rho_0 c}dA \tag{3.8}$$

By convolving with a second instance of the spatial impulse response, the pressure received at the transducer after the total trip to the scatterer and back is found as:

$$\begin{aligned} p_r(\vec{r},t) &= \rho_0 \cos(\theta(\vec{r},t))\frac{\partial\Phi(\vec{r},t)}{\partial t} \\ &= -\cos(\theta(\vec{r},t))\frac{\partial p_i(\vec{r},t-|\vec{r}-\vec{r}_S|/c)}{\partial t}\frac{dA}{4\pi|\vec{r}-\vec{r}_S|c} \end{aligned} \tag{3.9}$$

where $\theta(\vec{r},t)$ represents "the angle between the reflector surface unit normal and the particle velocity vector at $\vec{r}$" (Li Wan thesis FIX!!). For a given acoustic-electric receiving transducer impulse response $E_r(t)$, a given acoustic-electric transmitting transducer impulse response $E_t(t)$, and transducer excitation voltage $v_{exc}(t)$, such that $u_n(t) = v_{exc}(t) \star E_r(t)$, the final voltage waveform is [57]:

$$dv_r(\vec{r},t) = \frac{\rho_0}{c}\cos(\theta(\vec{r},t))E(t) \star v_{exc}(t) \star (\frac{\partial^2}{\partial t^2}(h_t(\vec{r},t) \star h_r(\vec{r},t))dA \tag{3.10}$$

where $E(t) = E_t(t) \star E_r(t)$.

A consequence of Huygens Principle is that the small rectangular tiles, or "planar vibrating points" [24] will emit spherical waves, and the field can be found by summing the contribution from many spherical waves emitted by the sources (vibrating rectangular tiles) (for an isotropic, homogeneous, perfectly elastic (lossless) medium). Transducer apodization is incorporated by appropriately weighting the surface velocity values for each tile, and then

weighting them when the reflected pressure field is calculated. The size of the rectangles is chosen such that they are small enough for the far-field approximation. The width of the rectangles must satisfy [24]:

$$w \ll \sqrt{4lc_0/f} \tag{3.11}$$

where w is the largest side of the rectangle, l is the distance to the field point, and c_0/f is the wavelength.
Some assumptions have been made about the setup, namely that [24]:

1. The transducer is large (compared to the ultrasonic wavelength) and only slightly curved [41].

2. The excitation function $u_n(\vec{r}, t)$ and transducer geometry (spatial impulse response $h(\vec{r}, t)$) are separable into spatial and temporal components.

3.1.2 Field-II Simulation

Values for the simulation parameters are given below.

Transducer type: Linear Array	Kerf: 0.05 mm
Transducer Center Freq: 3.5 MHz	Transmit focal depth: 60 mm
Sampling Frequency: 100 MHz	Number of elements: 192
Speed of sound: 1540 meters/second	Num. active elements: 64
Element height: 5 mm	Apodization: 64-point Hanning

Three ultrasound volumes were simulated, each with multiple targets. Fifty scan lines were simulated for each 2D image, and the RF data saved from the Field-II simulation for each one. Once the simulation was complete, the data was processed to create the ultrasound images from the RF data. First, the Hilbert transform of each line was calculated and the magnitude of the result used as the envelope. Next, once the envelopes for each scan line had been computed, the data was linearly interpolated along the horizontal

direction, resulting in ten data points for each original point. The fifty lines were thus interpolated to 500 pixels along the horizontal direction. The resulting image was then stored in an 8-bit unsigned integer format (2D image). This process was repeated for each 2D image and once complete, Matlab was used to place all of the images into a 3D image volume and exported (RAW format). The MHD header file was then created manually.

The first simulated image was of five cyst targets. These were spherical and designed to mimic smooth, curved boundaries commonly found in medical images. Four identical, small targets were created, and placed at two different depths, to see how the segmentation accuracy changed for identical targets at different depths (23mm and 47 mm). The four small targets were spheres with a radius of 4mm. One large target with radius 12mm was placed at a depth of 35mm. One slice of the simulated volume is shown in Figure 3.2, and a cutaway of the 3D volume shown in Figure 3.3. Figure 3.3 was rendered with Volsuite 3.3t using a maximum intensity projection (MIP) method of reconstruction, as well a clipping plane (cut to halfway through the volume). The MIP method essentially assigns each voxel a transparency proportional to its brightness value dark voxel are very transparent, while bright voxels are more opaque.
Simulation 1 Information (see Figure 3.1):

- Image Dimensions: $[x, y, z] = [40, 62.5, 30]$ mm
- Resolution: $[\Delta x, \Delta y, \Delta z] = [0.08, 0.08, 0.5]$ mm
- Image Size: $[x/\Delta x, y/\Delta y, z/\Delta z] = [500, 781, 60]$ (number of voxels)

Computation was an issue which needed to be addressed. The Field II program did not require an exorbitant amount of memory for these volumes, but did require a lot of computation time. Each vertical RF line took between 5 and 8 minutes with a single-core, 2.4 GHz CPU and 1 gigabyte of RAM, and with 50 lines per 2D image, each image could take up to 7 hours to simulate. Sixty-one 2D images per 3D volume resulted in over 400 hours

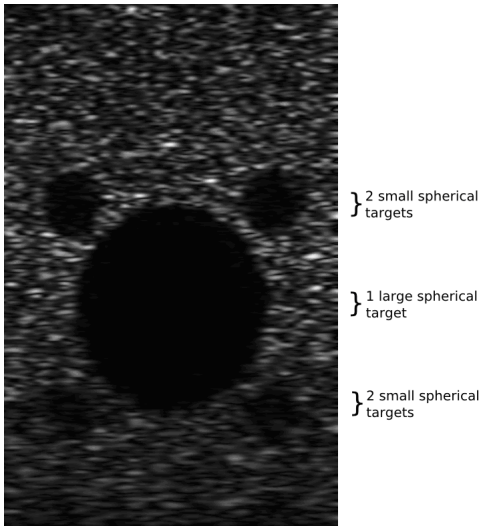

Figure 3.2: *2D scan plane of 3D simulated volume, with five spherical targets (Volume 1).*

of computation per 3D image. For three volumes, this amounts to about 1200 hours of number crunching, or over 7 weeks. To accommodate this, several options were investigated. The first option was to use Matlabs Distributed Computing Toolbox to automatically spread the calculations over many CPUs however, the cluster available on-campus runs Linux and the version of Field-II at the time did not support the Linux version of Matlab. Second, companies that rent computation on their own clusters were contacted but that same problem was found, where Field-II did not support the specific version of Linux-based Matlab that was available. Finally, ten WinXP computers within the WPI Electrical and Computer Engineering Department were secured for a few days, and each computer was assigned a specific set of 2D images to simulate. The images were gathered up by hand and combined to create the 3D volumes.

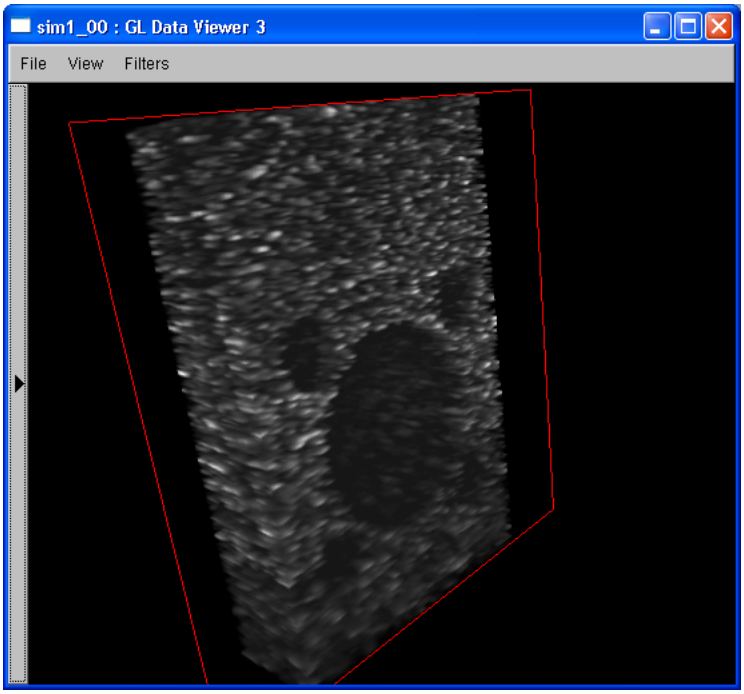

Figure 3.3: *3D view of simulated volume, with five spherical targets (Volume 1).*

3.2 Tissue-Mimicking Phantoms

Phantoms were created in-lab with help from Matthew Rowan, a fellow graduate student in the Ultrasound Laboratory, using a cyst-mimicking recipe by Brooke Buccholz of Boston University. The recipe as given below was used to mimic the tissue surrounding the target cysts. This agar-based phantoms recipe mimics human tissue, with material values similar to human tissue [3]:

- Density = 1045 kg/m^3 (Human 1000-1100 kg/m^3)
- Sound speed = 1551 m/s (Human 1450-1640 m/s)
- Attenuation = 10.17 $Np/m/MHz$ (Human 4.03-17.27 $Np/m/MHz$).
- Attenuation = 0.884 $dB/cm/MHz$ (Human 0.350-1.50 $dB/cm/MHz$)

To create the cysts, three variations of the recipe were used, where the amount of graphite of the cyst was reduced to 35%, 45%, and 55% of the surrounding tissue mimicking material graphite amount (by weight). These values were chosen because at 35% of the graphite concentration of the surrounding tissue, the cysts were well defined. Increasing the value up to 55% reduced the contrast to the point where ultrasound image of the cyst became nearly indiscernible from the surrounding tissue to the human eye. The goal was to present the segmentation methods with "easy" and "difficult" images, to challenge the methods and determine the most robust one. Cylinders and rectilinear box shapes were used for the cyst targets. Box shapes were chosen because of their sharp edges, which would challenge the segmentation routine, especially at the corners. These sharp edges were desirable because the segmentation routine would be geared more towards typical clinical shapes, which are typically curved in nature. By evaluating the accuracy on sharp objects, the segmentation could be shown to be more robust to irregular shapes, such as fluid volumes which may have sharp corners and turns. The cylinders were chosen to use more natural, curved shapes, but have sharp edges as well. Spherical targets were tried but were found to be very difficult to create with reasonable accuracy in shape using the tissue mimicking material.

As mentioned above, several versions of the recipe were used to create multiple phantoms with differing levels of contrast. The cyst material was poured into the molds and allowed to harden overnight in a refrigerator. Next, a small rectangular box used as a mold was filled halfway with the surrounding tissue material, and allowed to cool for a few minutes to slightly gel, as the freshly made material is not hard enough to support the cysts and they would sink to the bottom. The targets were then placed on top of the slightly cooled layer, and a layer of surrounding tissue material poured in to completely cover the targets. The phantom targets' dimensions are shown below in Table 3.1. Note that the scanning was carried out so that

the cylinders were oriented along the z-axis, and such that the circle appears in the x-y plane. The precise locations of the phantom targets were actually unknown, and presented a unique difficulty that is explained in Chapter 7.

Table 3.1: *Phantom Dimensions.*

Phantom Target	Dimensions [mm]
Box, 35% Graphite	6.34 x 6.34 x 30 [x,y,z]
Box, 45% Graphite	6.34 x 6.34 x 30 [x,y,z]
Box, 55% Graphite	6.34 x 6.34 x 30 [x,y,z]
Cylinder, 35% Graphite	Radius 3.584, Length 35.5239
Cylinder, 45% Graphite	Radius 3.584, Length 32.7456
Cylinder, 55% Graphite	Radius 3.584, Length 35.2645

Recipe (Courtesy of Brooke Buccholz and Matthew Rowan) [3], [44]:

1. 600ml Distilled Water
2. 18g Agar
3. 0.75g Methyl Paraben
4. 54g Graphite Powder
5. 50ml 1-Propanol

Procedure:

1. Heat the distilled water to a minimum of 85 °C.
2. Add the agar and mix well with water at the above temperature.
3. Add the methyl paraben and mix well.
4. Cool the mixture to 80 °C.
5. At 80 °C add the graphite powder and mix well.
6. Allow the mixture to cool to 70 °C.

7. Add the 1-propanol.

8. Allow the mixture to degas under vacuum (lose internal air bubbles) for 15 minutes while still maintained at 70 °C.

9. Pour solution gently into the mold.

10. Cover the mold tightly to prevent in-gassing.

11. Let sit for approx 12 hours.

12. Store the phantom in a degassed environment. This solution can be used for upwards of 2-3 weeks without losing its properties.

Once the phantoms were complete, an xy recorder was used to perform the ultrasound scan. The Terason transducer was mounted on the movable xy recorder, and many B-mode ultrasound scans recorded using Terason's program on a PC. Each scan was taken after moving the transducer laterally, in increments of 0.794 mm (Δz). The x and y image resolutions were 0.055147 and 0.0547009 mm, respectively (Δx and Δy). These 8-bit grayscale 2D images were stored and a program built on ITK used to gather the images into a single 3D volume (MHD/RAW formats) with associated spatial information (spacing).
Phantom Images Information (see Figure 3.1):

- Image Dimensions: Varies for each image.
- Resolution: $[\Delta x, \Delta y, \Delta z] = [0.055147, 0.0547009, 0.794]$ mm
- Image Size: $[x/\Delta x, y/\Delta y, z/\Delta z]$ = Varies for each image.

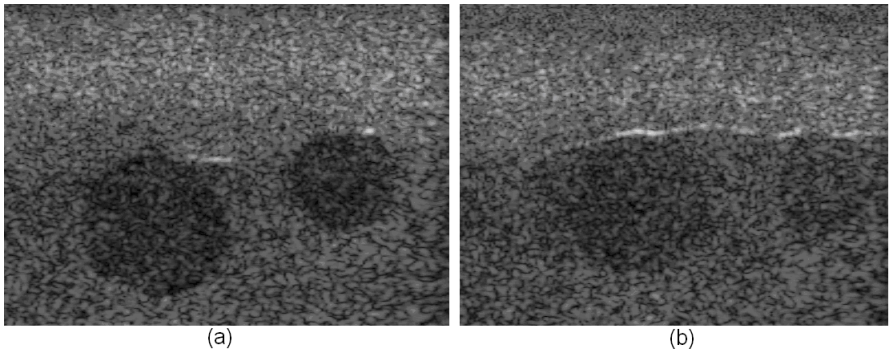

Figure 3.4: *2D view of cylinder phantom volumes. (a) 35% graphite; (b) 45% graphite).*

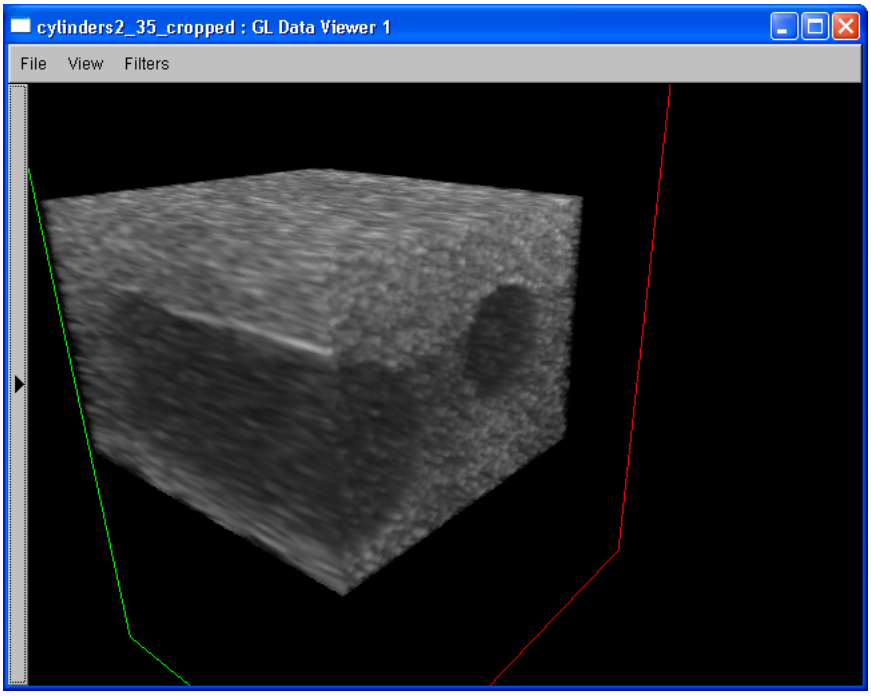

Figure 3.5: *3D view of cylinder phantom volume, with cutaway (35% graphite).*

3.3 Real 3D Ultrasound Prostate Scans of Human Subjects

Real scans were acquired from the Robarts Institute, courtesy of Dr. Aaron Fenster and his team. These comprise three complete datasets of prostate ultrasound scans, each from a different patient. The three male patients were aged 74, 76, and 77 years old. The Philips ATL HDI-5000 transducer [43] was used to acquire the image data, which can operate with a center frequency between 5 and 9 MHz. The curved imaging mode was used, which has a 150 field of view. The images were stored as 8-bit grayscale images in the L3D format. Note that no identifying information has been or will be provided for these three patients. The Ethics Approval Notice number for the Use of Human Subjects at The University of Western Ontario is 12682E, and the title is "Comparison of 3D transrectal ultrasound (TRUS) to conventional 2D TRUS in the measurement of prostate volume". The Principal Investigator for this approval was Dr. C. Romagnoli (Radiologist).

A custom program from Dr. Fenster's lab was used to convert the L3D volumes into RAW volumes and the MHD header files were created manually. Note that a bug with the conversion program led to a partially incorrect third volume. However, the error manifested itself at only one small section near the edge of the image, and was acceptable. Information about each image volume is given below, using the terms defined in Figure 3.1.
Image 1 Information (see Figure 3.1):

- Image Dimensions: $[x, y, z] = [54.21, 62.83, 124.28]$ mm
- Resolution: $[\Delta x, \Delta y, \Delta z] = [0.154007, 0.154007, 0.154007]$ mm
- Image Size: $[x/\Delta x, y/\Delta y, z/\Delta z] = [352, 408, 807]$ (number of voxels)

Image 2 Information:

- Image Dimensions: $[x, y, z] = [62.95, 74.23, 145.08]$ mm

- Resolution: $[\Delta x, \Delta y, \Delta z] = [0.182999, 0.186005, 0.186005]$ mm
- Image Size: $[x/\Delta x, y/\Delta y, z/\Delta z] = [344, 399, 780]$ (number of voxels)

Image 3 Information:

- Image Dimensions: $[x, y, z] = [53.59, 65.76, 132.45]$ mm
- Resolution: $[\Delta x, \Delta y, \Delta z] = [0.154007, 0.154007, 0.154007]$ mm
- Image Size: $[x/\Delta x, y/\Delta y, z/\Delta z] = [348, 427, 860]$ (number of voxels)

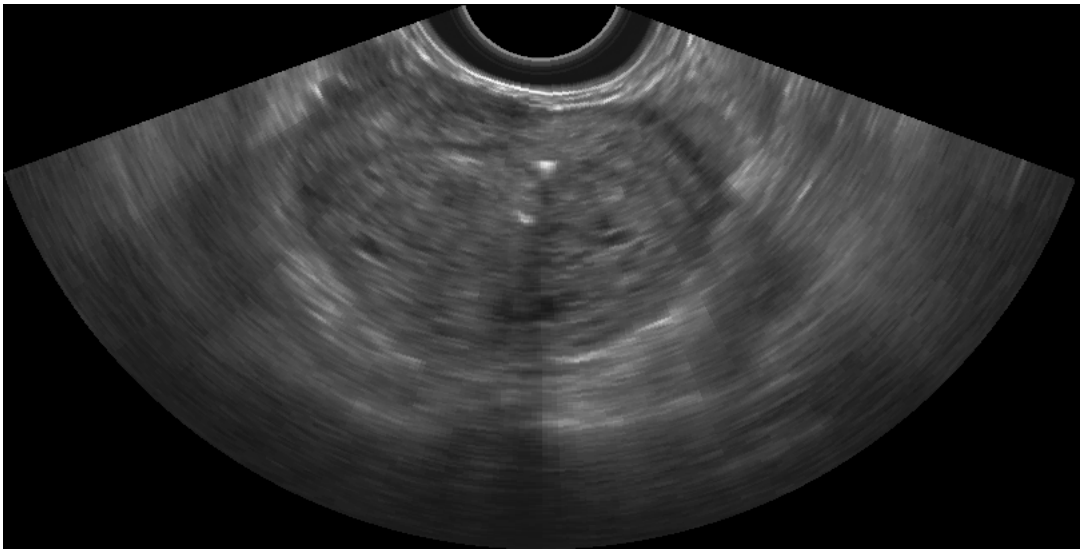

Figure 3.6: *2D slice from real prostate volume (Patient 1).*

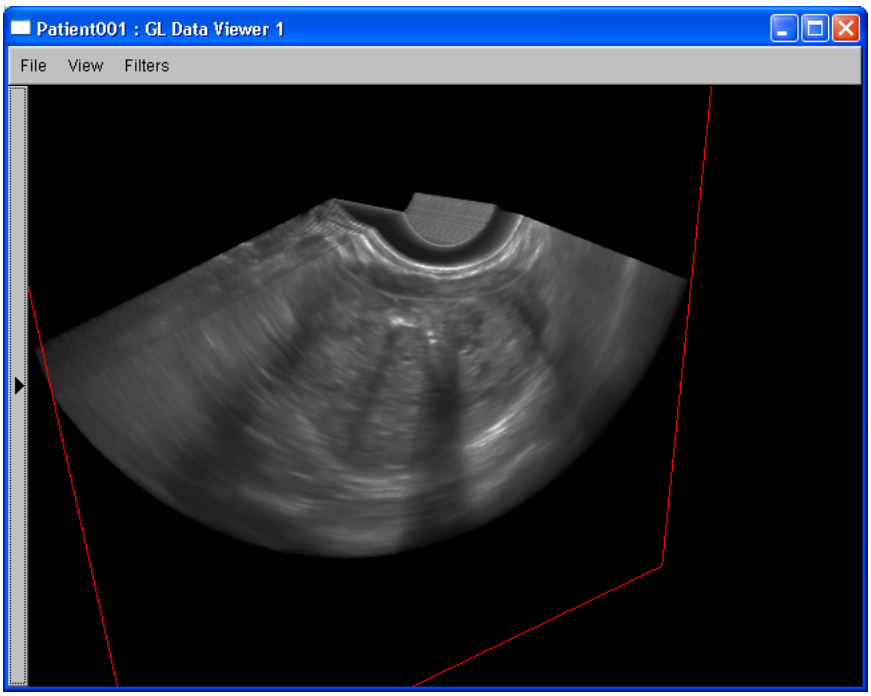

Figure 3.7: *3D view of real prostate volume, with cutaway (Patient 1).*

Chapter 4

Preprocessing

4.1 Introduction

This chapter covers the preprocessing methods that were explored. As ultrasound images are inherently very noisy, contain artifacts and have a low signal-to-noise ratio (as compared to imaging modalities such as CT or MRI), it is necessary to try to reduce this noise as well as artifacts before segmentation to get the most accurate results possible. The integrated backscatter technique and the various histogram-altering methods were evaluated early in the research, utilizing Vikramjit Mitra's Nearest Neighbor algorithm [40] as the segmentation accuracy baseline, whereas speckle reduction methods were examined later on with the level set segmentation methods. Also, is should be noted that preprocessing techniques using the Nearest Neighbor algorithm (IBS, histogram operations) were also only applied to 2D images, as the 3D volumes were not available until later which used the level set segmentation method. The basic procedure is outlined in Figur 4.1. The input image is first normalized (explained in the next section), then some preprocessing methods are performed, the processed image segmented, and the performance of the segmentation evaluated. The performance was used to determine which preprocessing method gave the best segmentations. It

should also be noted that the performance evaluation used for NN results differs from the performance evaluation developed for the 3D segmentations using the level set method, although the NN results were clear enough to the eye to decide whether to (not) use IBS or histogram modifications. The 3D performance evaluation is described in Chapter 6.

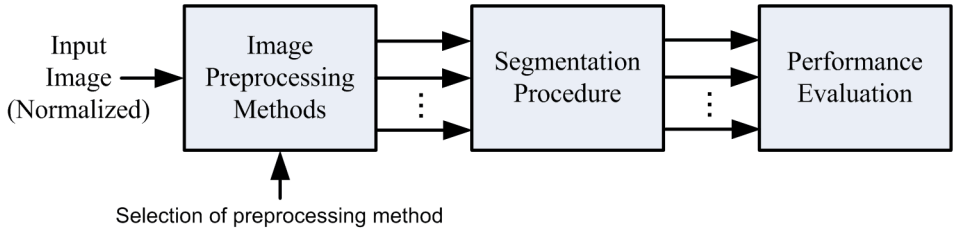

Figure 4.1: *Flowchart procedure for determining the best pre-processing method.*

To put the research in some perspective before the full results are presented in a later chapter, here is the basic sequence of events that transpired in the research with respect to the preprocessing selection:

1. IBS was used using NN segmentation, only with 2D images.
2. IBS was not found to increase segmentation accuracy.
3. Histogram operations explored (without IBS) with NN segmentation baseline.
4. Histogram operations were not found to increase segmentation accuracy.
5. Level set segmentation method supplants NN segmentation.
6. Speckle reduction schemes evaluated w.r.t. level set segmentation performance using true 3D images.
7. Speckle reduction scheme chosen.
8. Final procedure: no IBS, no histograms operations, one speckle reduction method.

4.2 Image Normalization

All of the images discussed in Chapter 3 were normalized to the full range of 0-255 available with 8-bit unsigned integers before processing. The purpose of this was to increase the contrast as much as possible, such that the image histogram filled the entire range. The normalization was accomplished by first subtracting the lowest value in the image from every voxel, such that the minimum value was zero. Then, the image was scaled up such that the maximum value was 255. This was the first step of the image processing pipeline, and was performed for every technique described hereafter. The equation below describes the normalization procedure for an N-dimensional input image, I.

$$I_{norm} = 255 \cdot \frac{I - I_{min}}{max[I - min(I)]} \tag{4.1}$$

4.3 Nearest Neighbor Algorithm

The segmentation method initially used to evaluate IBS and histogram modification performance was the Nearest Neighbor algorithm [40]. This section serves to provide background on this boundary detection method. Before the Nearest Neighbor algorithm was applied to the 2D ultrasound images, a few pre-processing steps were performed (as defined in [40]). First, the image was thresholded with an image histogram-dependent value given by (after normalization, as previously mentioned):

$$t_h = \frac{\sigma}{4\mu} \tag{4.2}$$

where μ is the mean pixel brightness of all pixels in the image, and σ the standard deviation. This equation was developed by trial-and-error to find an appropriate threshold level for identifying low-intensity valued cysts. All pixels with values below this threshold were increased by (255-th), and values

above discarded (set to zero). This was due to the fact that the cysts explored had very low intensity values compared to the surrounding tissue.

After thresholding, the new image was filtered using a 2-level Discrete Wavelet Transform with a Daubechies mother wavelet to reduce noise, as it was found to increase boundary detection accuracy for real ultrasound images (but no real difference for simulated images). The DWT-filtered image was then median-filtered, using the same method described later in this chapter with a 6x6 sized kernel. The median filtered image was then filtered using a 2D adaptive Wiener filter with a 50x50 window size for noise parameter estimation. The Wiener filter acts to reduce additive noise present in the image. The Wiener-filtered image then underwent a morphological closing and then opening operation to erase small artifacts and smooth the boundaries of the objects present. A circular morphological structuring element with a three-pixel radius was used for both steps. The output image of these operations $a(x, y)$ was then presented to the Nearest Neighbor algorithm.

The Nearest Neighbor formulation in [40] is given as:

$$a(x,y) = 1, \text{when} \frac{1}{N} \sum_{r=x-n}^{x+n} \sum_{k=y-n}^{y-n} a(r,k) > t_h \tag{4.3}$$

$$a(x,y) = 0, \text{when} \frac{1}{N} \sum_{r=x-n}^{x+n} \sum_{k=y-n}^{y-n} a(r,k) < t_h \tag{4.4}$$

where N is the number of pixels in the image and n is the neighborhood size (length on pixels of one side of a 2D square). The output depicts the probability that a given pixel belongs to an object, given the information about its surrounding neighbors. Note that this output was formulated to return binary values of either zero or one, with a one representing pixels that belong to an object.

To evaluate the segmenation accuracy, the metric used by the NN authors and for the results presented in Section 8.1 is given as:

$$\text{Accuracy} = 100 \cdot \left(1 - \frac{\sum_{r=1}^{N}\sum_{c=1}^{M}\left[b(r,c) - b_d(r,c)\right]^2}{\sum_{r=1}^{N}\sum_{c=1}^{M} b(r,c)}\right) \tag{4.5}$$

where $b(r,c)$ is the ground truth object and $b_d(r,c)$ is the segmented object.

4.4 Integrated Backscatter

The Nearest Neighbor algorithm was initially used to evaluate the usefulness of the integrated backscatter calculation (IBS) with regard to segmentation accuracy. Some preliminary results using the NN algorithm are presented in this section, however full results using the level set segmenation (see Chapter 5) are presented in Chapter 8. The IBS serves to find the energy of the signal reflected by the scatterers situated within the volume. As shown in Figure 4.2, the raw RF data and IBS-processed data were used before the image formation process (described in Chapter 3). RF data was only available from the simulated Field II images and the phantom images; only post-image formation data was available fr the clinical data (human prostate scans), so IBS could not be performed on those.

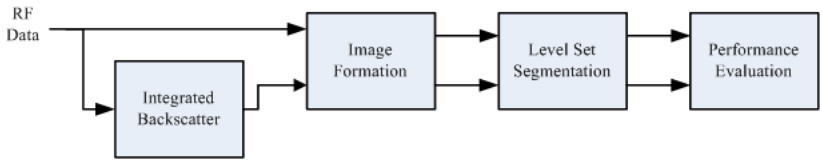

Figure 4.2: *Procedure for evaluating usefulness of the Integrated Backscatter calculation.*

Essentially the energy of the backscattered signal is calculated for each RF scan line. This energy signal is then convolved with a sliding window. The equation for IBS is given by Szabo as [50]:

$$S_r = \int_{f_0-\Delta f}^{f_0+\Delta f} \frac{|V_0(f)|^2}{|V_{ref}(f)|^2} df = \frac{\int_{-\infty}^{\infty} |v_0(t)|^2 dt}{\int_{-\infty}^{\infty} |v_{ref}(t)|^2 dt} \tag{4.6}$$

where $|V_0(f)|^2$ is the signal power spectrum, Δf is half the usable bandwidth, and $|V_{ref}(f)|^2$ is some reference signal. The equality between the energy of the spectral and time domains is a result of the application of Parseval's Theorem. As the reference will be the same scalar value for all RF lines and for these purposes is essentially arbitrary, the reference signal energy was set to one for all cases. A sliding window was used to "extract" many small segments from the full RF envelope - an 85-point Hanning window was utilized. Similar results were found for other window sizes. Additionally, the IBS result was scaled such that the energy of the entire image was equal to that of the original, non-IBS data.

It was found that the squaring operation led to images with a high dynamic range (some very high intensities that dwarfed the bit range and darkened the rest of the image noticeably). The IBS operation was also found to blur the image (only along one dimension in the final images - the vertical RF scan lines). This smearing, only along one direction, led to some high-frequency noise along the horizontal axis which did not improve the image quality. The smearing also obscured boundaries that were much clearer in the original images by blurring them. The IBS calculation was not found to be useful due to the blurring along the vertical axis, the addition of high-frequency noise and discontinuities along the horizontal axis, and mostly because of the tendency for a few bright pixels to suddenly dominate the entire image bit-range, making the rest of the image very dark.

Presented here are some segmentation examples for the NN algorithm (see Chapter 8 for full results using level set segmentation). An images from a simulation using the Field-II program was used, so that exact boundaries were known and accuracies could be computed. Two different image creation techniques were applied to the image, which are listed below. The RF and processed waveforms are shown in Figure 4.3. An example image is shown in Figure 4.4, along with the IBS case.

1. Field-II created image

2. Field-II data with IBS (integrated backscatter)

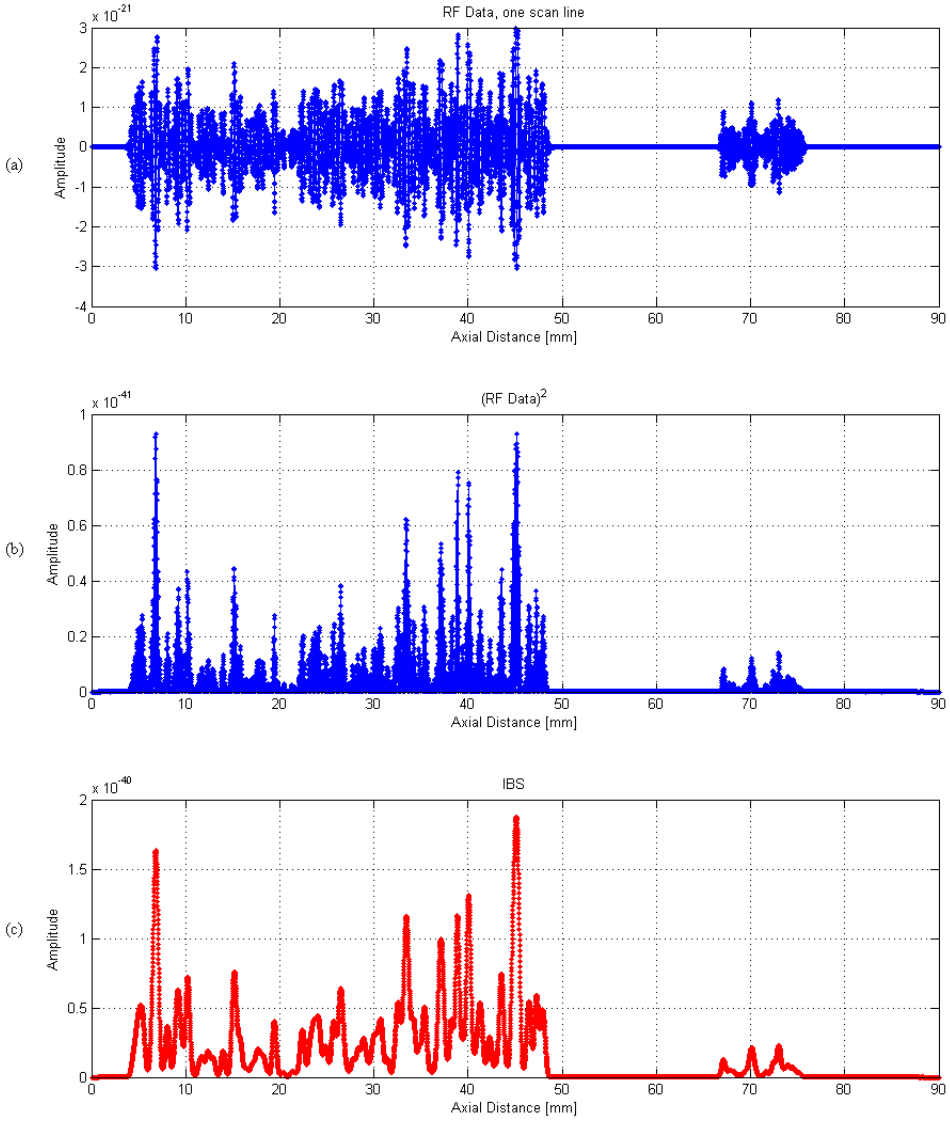

Figure 4.3: *(a) Waveform showing RF data for one scan line; (b) result of squaring the RF data; (c) final IBS data (RF data squared, convolved with an 85-pt Hanning window).*

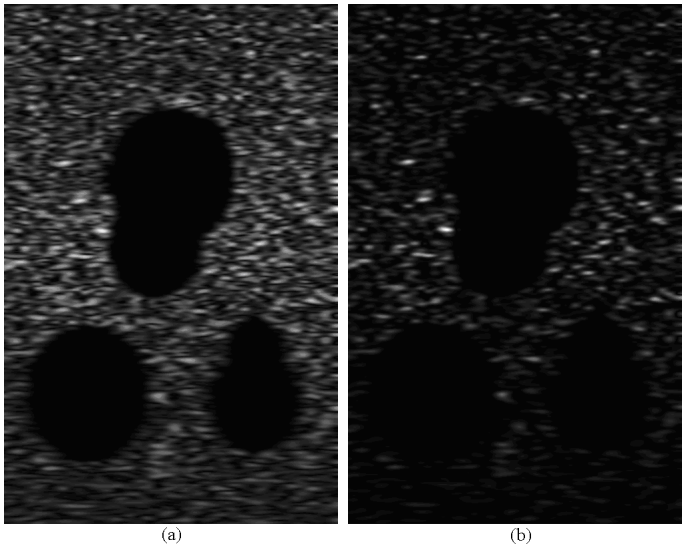

Figure 4.4: *Field-II simulated 2D images. (a) - Original image; (b) - IBS calculated image. Note the much lower contrast ratio and eclipsing of entire image in IBS case by a few small groups of bright pixels. Note that the IBS image has been normalized and occupies the same brightness range as the image on the left, but looks very dark.*

Some results of the Nearest Neighbor segmentation routine with and without the IBS preprocessing. are shown below in Figure 4.5. The IBS version fares rather poorly, while the segmentation does rather well without the IBS.

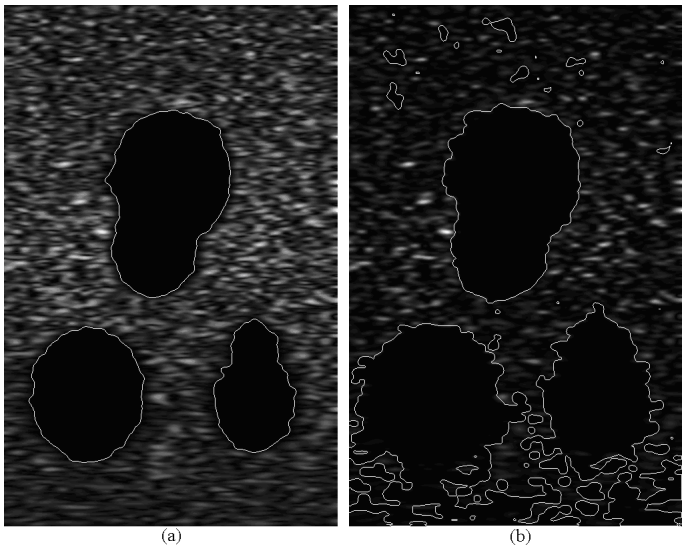

Figure 4.5: *Nearest Neighbor segmentations for original images. (a) No IBS; (b) with IBS.*

4.5 Histogram Modification

To improve boundary detection accuracy, image histogram modifications were investigated. An image histogram is simply a tabulation of the frequency of pixel gray-levels that occur in the entire image. For these 8-bit images, 256 gray-level bins were used. As an example, a dark image would have a lot more data in the lower range, whereas a bright image would see histogram peaks near the right (near 255). An image not utilizing the full bit range would see a lot of empty bins near the low or high ends of the intensity range. And example of an image histogram comprised of mostly dark pixels is shown in Figure 4.6.

The goal of this section is to investigate methods of modifying an image's

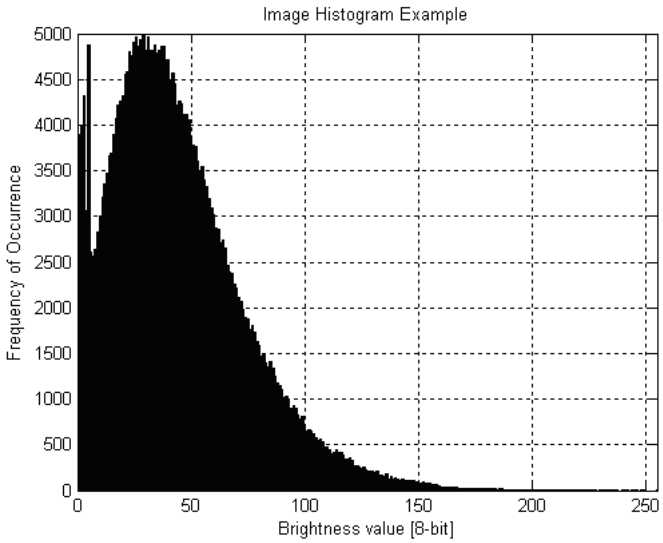

Figure 4.6: *Example of image histogram from a very dark image with very few bright pixels.*

histogram to possibly enhance boundaries of interest or bring out features that are initially hard to discern. Such modifications may simply shift the image histogram into a higher or lower range, stretch or squeeze the histogram, perform equalization, or transform the image histogram into a desired (specified) shape. Such operations are in general non-unique. Histogram modification is typically used for enhancing photographs to make them look more pleasing to the human eye, and was investigated near the beginning of the research using Vikramjit Mitra's Nearest Neighbor algorithm. It was not found to be generally very useful for the Nearest Neighbor segmentation. However, the problems associated with images that have undergone such transformations, such as the introduction of high frequency noise in general carry over to the segmentation methods explored later in the research.

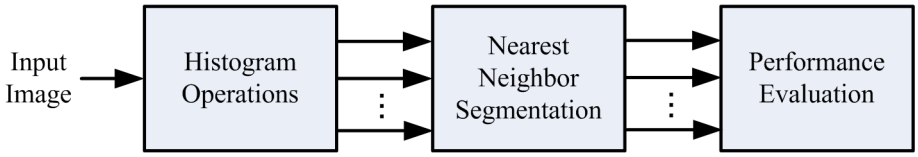

Figure 4.7: *Procedure for evaluating performance of various histogram modification operations.*

4.5.1 Histogram Equalization

The standard method for equalizing an image histogram is to apply a transformation that is defined by the normalized cumulative distribution of the original image histogram. $T(k)$ is a monotonically increasing grayscale transformation function that defines how input intensities are remapped to new output intensities. For example, if $T(k = 1) = 5$, then all pixels with intensity 1 are changed to an intensity of 5. If $hist_{in}(k)$ is the input image histogram, then the probability distribution function (PDF) of pixels, normalized to a sum of one is:

$$p(k) = \frac{hist_{in}(k)}{N} \tag{4.7}$$

where N is the number of pixels in the image and k is the pixel brightness value under consideration. From this, the cumulative distribution function (CDF) can be formulated:

$$c_0(k) = \sum_{i=0}^{i<k} p(i) = T(k) \tag{4.8}$$

The transformation function is taken directly from the CDF [34]. The input image is then subjected to the transformation function $T(k)$, by replacing brightness values k with the new brightness values $T(k)$. The output image is then normalized to the [0,255] range using (4.1).

For example, take the following randomly generated 6x6 image, which has been purposely created from a uniform PDF, $U \sim [0, 128]$:

Table 4.1: *Randomly generated 6x6 image.*

59	54	72	74	47	52
88	58	65	111	37	41
92	88	105	16	104	40
41	29	26	49	100	113
36	4	120	26	113	82
64	42	68	113	96	80

The histogram for this randomly generated image is shown in Figure 4.8.

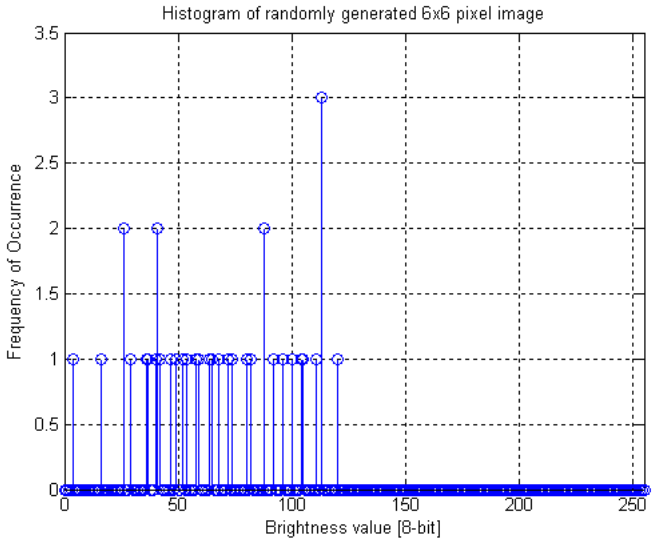

Figure 4.8: *Histogram of randomly generated 6x6 image.*

The normalized CDF of Figure 4.8 is shown in Figure 4.9.

Once the transformation function in Figure 4.9 has been applied to the input image (Table 4.1), the normalized output is:

The histogram for the image values given in Table 4.2 is:

Note that the histogram from Figure 4.8 has been stretched to fill more of the available bit range, as shown in Figure 4.10.

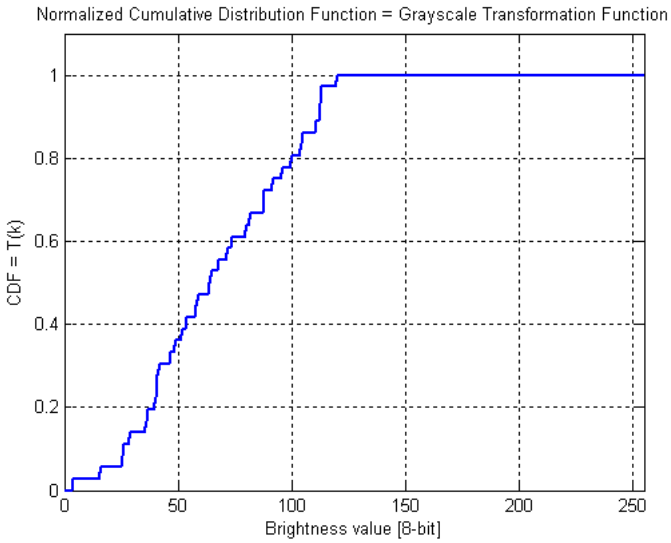

Figure 4.9: *Normalized CDF (Transformation function) of histogram given in Figure 4.8.*

Table 4.2: *Equalized output of image from Table 4.1.*

117	102	146	153	81	95
175	110	132	226	44	59
190	175	219	8	212	51
59	30	15	88	204	234
37	0	255	15	234	168
124	73	139	234	197	161

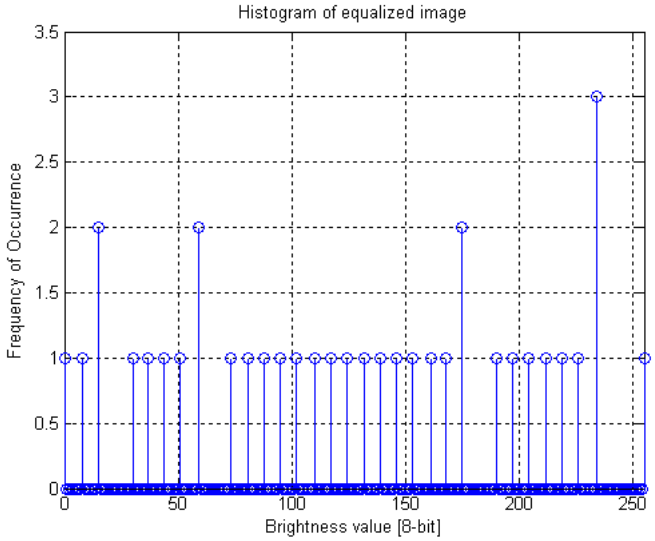

Figure 4.10: *Histogram of equalized image from Table 4.2.*

4.5.2 Histogram Specification with Matlab Image Processing Toolbox

The textithisteq() function in Matlab's Image Processing Toolbox will transform an input image given a desired histogram, but will not transform it perfectly as the transformation is subject to some constraints. In the equation given below (4.9), $c_1(k)$ is the CDF of the desired histogram and $c_0(k)$ is the CDF of the input image histogram, where k = [0,255] is the graylevel intensity. $T(k)$ is the monotonically increasing grayscale transformation that is chosen to minimize the error of [34]:

$$|c_1(T(k)) - C - 0(k)| \tag{4.9}$$

Note that the Matlab implementation of histogram equalization does not directly use the CDF for the transformation function, but seeks an optimized version of it. $T(k)$ is constrained in that $c_1(T(a))$ cannot overshoot $c_0(a)$ by more than half of the graylevel intensities. In other words, the input

pixels with graylevel '0' (black) can only be remapped as high as '128' (the middle gray value). For images with a very large spike or peak localized near either end of the brightness spectrum, this constraint is mainly why the output histogram will approach but still be much different from the desired histogram.

The following were used as the desired histograms for the histeq() function (Note that the outputs will merely approach the desired histograms, but rarely obtain it exactly). Typically, the implementation of histogram modification is in the context of photo manipulation software. As such, the user will click and drag points on a curve, immediately preview the image results, and alter it further. The end result is usually some non-uniform, arbitrary histogram. For this research, a few desired histograms were chosen arbitrarily to investigate. Note that the overall y-axis (frequency of occurrence) scaling in Figure 4.11 is dependent on the image size (number of pixels).

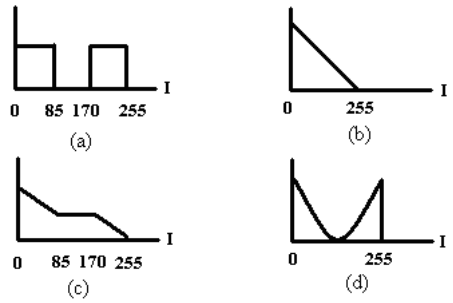

Figure 4.11: *Desired histogram types for histogram specification. (a) - Bandstop; (b) - ramp down; (c) two ramps down; (d) - parabolic* $[g(x-128)^2]$.

As example of the process for the parabolic histogram (Figure 4.11(d)) is shown below in Figures 4.12-4.14. Figure 4.12 shows the input image and the output image. Figure 4.12 shows the input image histogram, as well as the contrasted desired and output histograms (as mentioned, this process will approach the desired histogram, but not reach it perfectly). Figure 4.14 shows the grayscale transformation function that was solved for and does the best possible job at approximating the desired output histogram. It

was found that nearly all of these techniques degraded image quality and introduced (rather than reduced) noise.

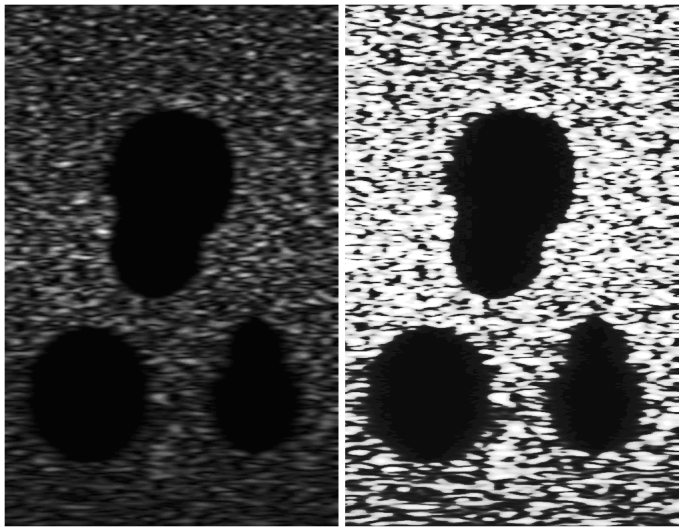

Figure 4.12: *Input and output images for histogram specification method, using parabolic shape as the desired histogram. (a) Input image; (b) output image. Note that the image looks washed out, and the boundaries enhanced only for the case of targeting a perfectly black cyst (only the simulated images- the non simulated images do not contain targets quite as strikingly different from the tissue intensity). Both images have been normalized.*

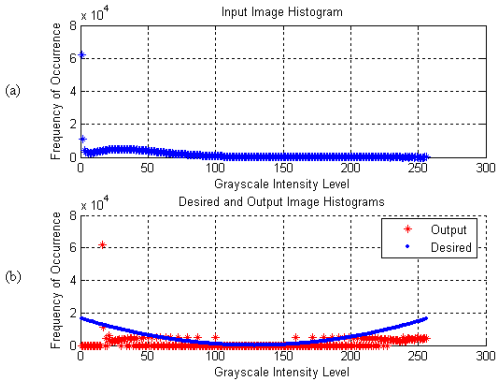

Figure 4.13: *Image histograms for histogram specification method, using "parabolic" as the desired histogram. (a) Input image histogram (Figure 4.12(a)); (b) desired histogram in blue, actual output histogram shown in red. Note that there is some discrepancy between the desired and actual output histograms, and the output contains a large amount of oscillation.*

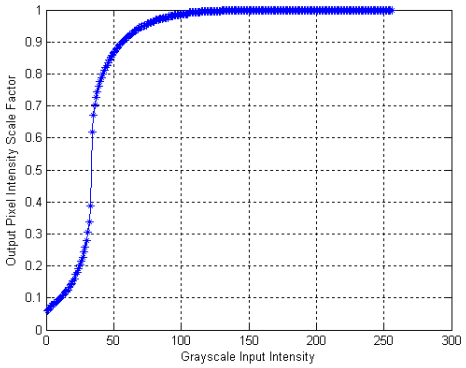

Figure 4.14: *Grayscale transformation function for histogram specification method that maps input intensities to output intensities for Figure 4.12, using parabolic shape as the desired histogram. Note that this transformation function is in the range of [0,1], and normalization is performed after transformation.*

4.5.3 Perfectly Flat Histogram Equalization

The technique described in [30] will take an input image and always output an image that has a perfectly flat histogram, and it was developed specifically

for enhancing x-ray images. Our implementation of their algorithm was implemented with Matlab, and the code is included in FIX!Appendix!A!FIX. There are three main phases in this technique:

1. Histogram Spike Redistribution
2. Histogram Specification (Using a pre-defined histogram with Matlab histeq.m)
3. Histogram Smoothing

The first phase was omitted due to the difference in application (the cyst targets for this ultrasound work differ from the x-ray enhancement that this process was developed for) but is presented for clarity. For a given number of pixels in an input image, $m \cdot n$, there is a target number of pixels for a perfectly flat histogram, where $tgt = \frac{m \cdot n}{256}$. If a graylevel has more pixels than $tgt \cdot thresh$, where $thresh$ is a thresholding parameter set to 16 by default, then pixels with those graylevels are replaced with a horizontal gradient that stretched from black (left) to white (center) to black (right). This is not desirable for our ultrasound application, as the dark cysts are our targets for segmentation and re-mapping them to a bright gradient severely impedes the boundary detection performance.

The second phase uses the *histeq()* function described in section 4.5.2. The output histogram is defined as $c_{ph2}(k)$. The desired histogram is described by:

$$hist_{des}(k) = \begin{cases} 4 \cdot tgt & \text{if } k/4 \text{ is in } \{0, 1, 2 \cdots\} \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

The third phase traverses the histogram of the phase two output, smoothing it as it goes. Starting at $k = 0$, the number of pixels, $c_{ph2}(k)$, is compared with tgt. If there are excess pixels, then $[c_{ph2}(k) - tgt]$ are moved into the

$k+1$ graylevel. This is accomplished through a random pixel selection process, where $[c_{ph2}(k) - tgt]$ of $c_{ph2}(k)$ pixels are changed to graylevel $k+1$.

If $c_{ph2}(k)$ is less than tgt, then pixels are taken from a higher intensity and moved into graylevel k. Once the first $c_{ph2}(k+n)$ intensity that has more pixels than $c_{ph2}(k)$ is found, then either:
a) Enough pixels are taken from $k+n$ to reach tgt for k, or
b) Some pixels are moved from $k+n$ into k, and n is incremented to find enough excess pixels in successive graylevels to satisfy the target for k

The random pixel movement process is slightly different for these cases from when the k^{th} graylevel has more pixels than tgt. Pixels that have been changed are tracked, and those pixels are prioritized for the selection process ahead of those that were originally at that graylevel (have not been changed since the phase two output).

It was found that processing an image to have a perfectly flat histogram was not desirable for this application because of the noise added by changing the value of random pixels. As shown in the images below in Figure 4.15, what were once sharp boundaries between the cysts and surrounding tissue have now been obscured due to pixels being assigned new brightness values. Figure 4.15(a) and (b) show the original image and histogram, and Figure 4.15(c) and (d) show both the phase 2 output image and histogram, and e and f show the final (phase 3 output) image and histogram. The small deviation from "perfectly flat" for the final output is due to the fact that the number of pixels in the image was not perfectly divisible by 256 (the number of possible brightness levels for these 8-bit images). Note that the noise introduced in the final output may be difficult to see in printed versions of this document.

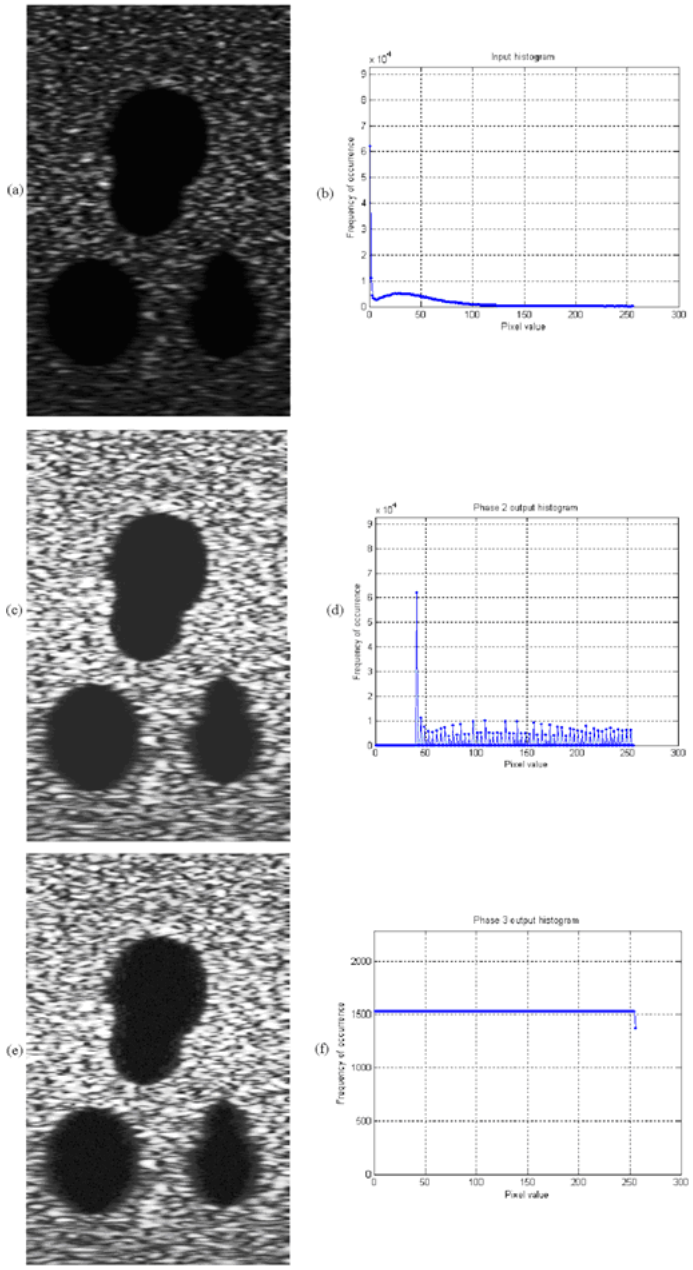

Figure 4.15: *Perfectly flat histogram equalization steps. (a) Input image; (b) input image histogram; (c) phase 2 output; (d) phase 2 output histogram; (e) phase 3 (final) output; (f) phase 3 output histogram.*

4.5.4 Noise Corruption

To find the most robust histogram modification method with respect to the NN algorithm's performance, three different noise types were added to the images before boundary detection: Gaussian, salt-and-pepper, and multiplicative speckle-type noise. These were added using the Matlab function imnoise(). The levels of noise were determined from test cases to find how much noise was required to push the BD accuracy down to about the 70-80% range, and then two values on either side of that value were tested as well. The following parameters were used:

- Gaussian: $\mu = 0, \sigma^2 = [0.1, 0.4, 0.6]$
- SnP: Percent of pixels affected $= [10\%, 40\%, 60\%]$
- Speckle: $\mu = 0, \sigma^2 = [0.5, 1.5, 2.5]$

4.6 Speckle Reduction Methods

Several methods for reducing the speckle of the ultrasound images were investigated. Speckle is a multiplicative-type noise that arises from constructive and destructive interference of the received acoustic signal. Speckle looks like a texture pattern in the image that manifests itself as small peaks in the image intensity. Since these intensity peaks can fool a segmentation routine into thinking it is a legitimate "edge" to be detected, reducing speckle noise can significantly increase segmentation accuracy. Several methods for reducing the speckle are presented below. Most methods are derived from the heat equation (diffusion) where the image is iteratively smoothed, the smoothing factor itself being a function of the image's local sharpness. These speckle reduction schemes were applied to the images and then segmented using the method outlined in the chapter on active contours. The performance of the segmentation was used to determine which pre-processing method was the best.

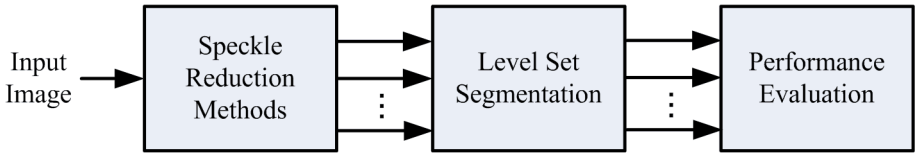

Figure 4.16: *Procedure for determining the best speckle reduction method, with respect to level set segmentation accuracy.*

Shown in Figure 4.17 is one 2D slice from the 35% graphite cylinder phantom. Figure 4.18 is a 2D Gaussian-filtered version of the original image. The noise has been reduced, but the boundaries between the cysts and the surrounding tissues are much less distinct now, and this shows one example of how where Gaussian filtering is an undesirable method for reducing ultrasound image speckle. The original image has undergone a 2D convolution with a 15x15 gaussian window ($\mu = 0$, $\sigma = 1$). Such a filtering scheme is known as isotropic, in that the same amount of filtering is applied to each section regardless of the local intensities. Median filtering, discussed next, is also an isotropic filtering process. The more advanced speckle reduction methods discussed afterwards are anisotropic in nature, where the amount of filtering applied to a specific patch is dependent on the local image intensities.

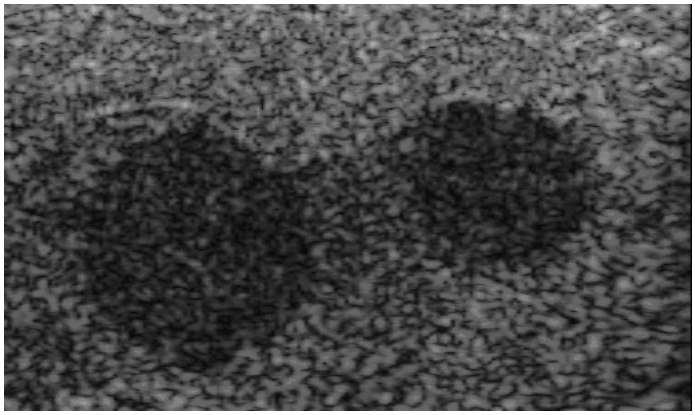

Figure 4.17: *Original image slice from 35% graphite cylinder phantom.*

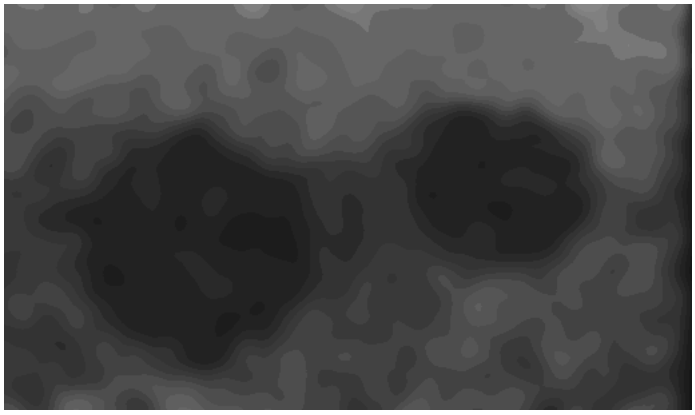

Figure 4.18: *2D Gaussian filtered version of original image, 15x15 kernel size. Noise is reduced but boundaries are very blurry (undesirable).*

4.6.1 Median Filtering

Median filtering is a way to reduce noise by essentially removing data outliers. A 3D kernel is slid through the image and for each iteration, the pixel values within that local neighborhood ordered. The center pixel is then assigned the median pixel value of the neighborhood. A 2D version of this process is shown in Figure 4.19, which shows the neighborhood operator concept. For each step though the image, the center pixel is under consideration. A neighborhood of surrounding pixels, defined by the kernel size, is extracted and ordered along with the center pixel value. The center pixel is assigned the median value of this neighborhood.

Shown in Figure 4.20 are the results of median filtering the image from Figure 4.17 using three different kernel sizes - the left shows a 3x3x3 neighborhood size, the middle a 5x5x5, and a 7x7x7 kernel size is shown on the right. The speckle (and possibly other types) noise has been reduced, while the cyst targets and boundaries are still visible to the eye. It should be noted that unlike the diffusion-based speckle reduction methods outlined in the next few sections, median filtering does not take the anisotropic nature of the voxel (pixel) spacing into account. Since the voxels are not isotropic

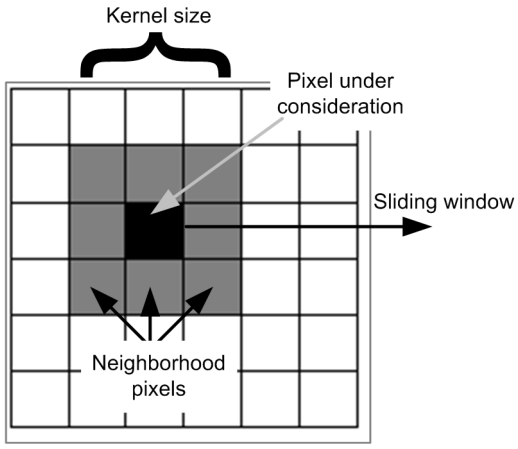

Figure 4.19: *Neighborhood operator concept. Center black pixel is under consideration, gray pixels represent neighborhood pixels associated with center. The window slides thought every pixel in the image.*

(cubic), the distance represented by 3 voxels in the x-dir is not the same as the distance represented by 3 voxels along the y or z-directions. This is undesirable and could be remedied by interpolating the 3D volume into equally-spaced dimensions, but the results obtained were satisfactory and did not seem to present any major problems.

4.6.2 Anisotropic Diffusion Image Filtering

Anisotropic diffusion methods of image filtering differ from isotropic diffusion methods in that the *direction* of smoothing is dependent on the local image features, which seek to preserve regions with strong transitions of intensity by smoothing *along* the "edges", rather than *across* them. First imagine the 3D image $I(x, y, z) = I(\vec{x}) = I$ is subjected to a deformation over time, leading to the set of derived images $I(\vec{x}, t)$ that are intensity functions of both time and space. The solution to the heat diffusion equation is given by [22] as (4.11), which relates the differential change in time to the spatial derivatives [19]:

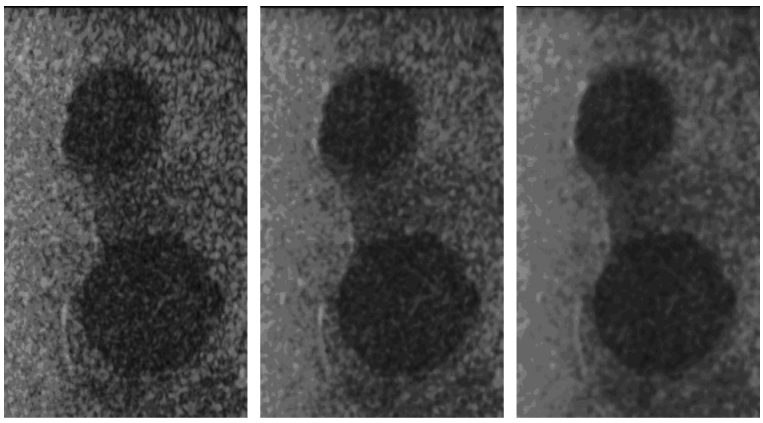

Figure 4.20: *2D slices from 35% graphite cylinder phantom, after 3D median filtering operations. (a) 3x3x3 kernel size; (b) 5x5x5; (c) 7x7x7. Note that the speckle-type noise has been reduced without significantly distorting boundaries (regions where brightness changes drastically in a small space, or rather, where the image gradient is high). The images have been rotated 90° CCW for display purposes.*

$$\frac{\partial I(\vec{x},t)}{\partial t} = \nabla \cdot c\nabla I(\vec{x},t) \tag{4.11}$$

Here, the deformation over time is related to the local image intensities. For regions where the spatial derivatives of the intensity are high, the change in intensity over time will be proportionally large. In terms of heat transfer, this can be viewed as very high temperature regions of a non-uniformly heated metal plate cooling off much faster when near a cold region than those near hotter parts. The heat will move much more quickly over time from hot to cold regions than hot to warm ones. Speckle analogously manifests as small high-temperature regions which we would like to allow to "cool" for a small amount of time, but keeping in mind we don't want the entire image to cool as quickly as the speckle spikes. In this way the spikes due to speckle are reduced without degrading the rest of the image equally. (4.11) can be expanded to show the individual components:

$$\frac{\partial I(\vec{x},t)}{\partial t} = c\left(\frac{\partial^2 I(\vec{x},t)}{\partial x^2} + \frac{\partial^2 I(\vec{x},t)}{\partial y^2} + \frac{\partial^2 I(\vec{x},t)}{\partial z^2}\right) \tag{4.12}$$

where c is a constant (in terms of the original heat equation, this is dependent on the thermal conductivity, density, and heat capacity of the material under consideration) and the original image is defined as $I(\vec{x}, t = 0)$. We can extend this by not restricting c to be a constant, but rather making it a function of the original image $I(\vec{x}, t = 0)$. In this case, as a consequence of a non-constant c (defined as $C(\vec{x})$), (4.11) becomes (4.13) [42]:

$$\frac{\partial I(\vec{x},t)}{\partial t} = C(\vec{x})\nabla I(\vec{x},t) + \nabla C(\vec{x})\nabla I(\vec{x},t) \tag{4.13}$$

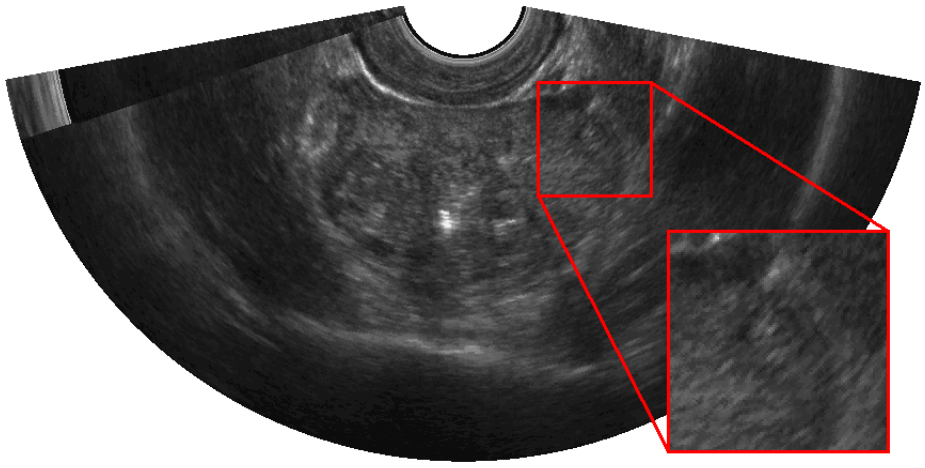

Figure 4.21: *2D slice from real human prostate scan, Patient 039.*

Figure 4.21 shows one slice from one 3D image volume, which is a real scan from a human patient of the prostate. Shown in Figure 4.22 is a filtered version of the original image volume. Notice that the image has been slightly de-noised but the boundary around the prostate is still relatively clear to the eye. The variation of the basic anisotropic diffusion filtering process outlined in Section 4.6.4 differs in that a spatially-varying $C(\vec{x})$ is used instead of a constant c.

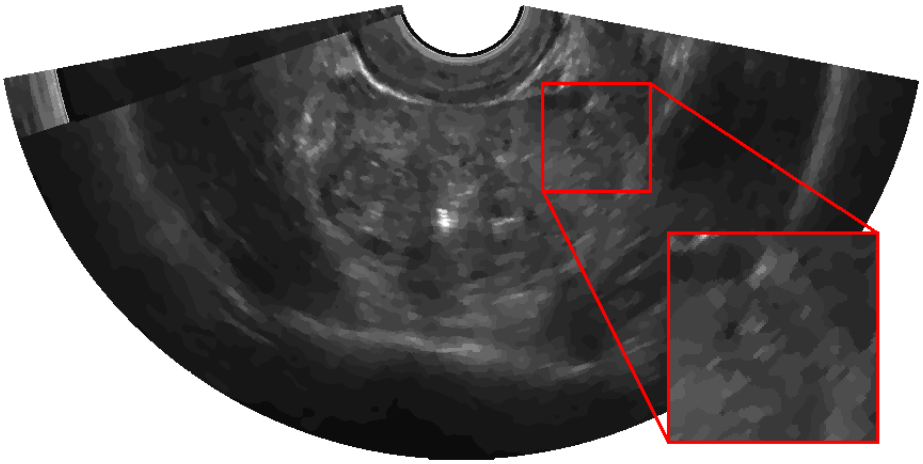

Figure 4.22: *Same slice as Figure4.21, after the 3D volume has been processed with the anisotropic diffusion procedure (50 iterations, time step 0.125, conductance (c) 1.0).*

4.6.3 Mean Curvature Evolution

This speckle reduction method [9] actually uses a level set formulation for speckle reduction, of which much more information will be presented in the active contours chapter, 5. Essentially, this negative curvature flow formulation works in a non-linear, anisotropic fashion by smoothing areas by an amount proportional to the local curvature of the image intensity (brightness). Regions with large spikes in the image intensity are smoothed more than areas that are already relatively homogeneous in brightness. This works to reduce the speckle type noise, as speckle is multiplicative in nature and manifests as distinct spikes in the image. The image I is subjected to the following iterative equation:

$$I_{i+1}(x, y) = I_i(x, y) + \Delta t \cdot C_i(I_i(x, y)) \tag{4.14}$$

where the coordinates (x, y) represent the pixel index along the horizontal and vertical dimensions, i represents the iteration number, and Δt represents the time step. This works to evolve an image according to its local curvature,

and apply forces along the surface normal proportional to that curvature. The curvature force along the surface normal, C_i, is calculated from (4.15):

$$\begin{aligned} C(I(x,y)) &= \frac{D_{xx}(I)D_y^2(I) - 2D_x(I)D_y(I)D_{xy}(I) + D_{yy}(I)D_x^2(I}{D_x^2(I) + D_y^2(I)} \\ &= -\kappa|g(I(x,y))| \end{aligned} \tag{4.15}$$

D_x represents the first derivative in the x-direction, D_{xx} the second derivative in the x-dir (and the same for D_y and D_{yy}, along the y-dir), and D_{xy} represents a mixed derivative in both and x and y directions. Periodic boundary conditions are enforced, where edge pixels are wrapped around to the other side of the image, rather than inserting zeroes for the unknown values. The time step parameter dictates how fast the image evolves for each iteration. Note that the units of time are arbitrary and the time step should be less than 0.5 in 2D image processing for stability concerns [9].

Shown in Figure 4.23 are two images that have undergone this speckle reduction process (The original image is shown in Figure 4.17). Both images were subject to a time step of 0.15, but the left image was allowed to run for 100 iterations, and the right allowed 200 iterations. In addition, shown in Figure 4.24 are two images, both subject to a time step of 0.30. The left image represents the state of the processing after 100 iterations, and the right shows 200 iterations.

4.6.4 Curvature-flow-based Image Filtering

Curvature is a measure of how fast a contour is changing direction. For example, the curvature along the circumference edge of a small circle has a high curvature compared to that of one with a larger radius. Contrast a small cone against a bowl shape that comes to a much smoother peak at the bottom, and has a larger (circular) base than the cone, shown in Figure 4.25. If a planar slice (parallel to the bases) were extracted near the tops of both of these objects, and the resulting projected contours examined, the cone would have a small circle, while the bowl would be a larger circle. We can say the

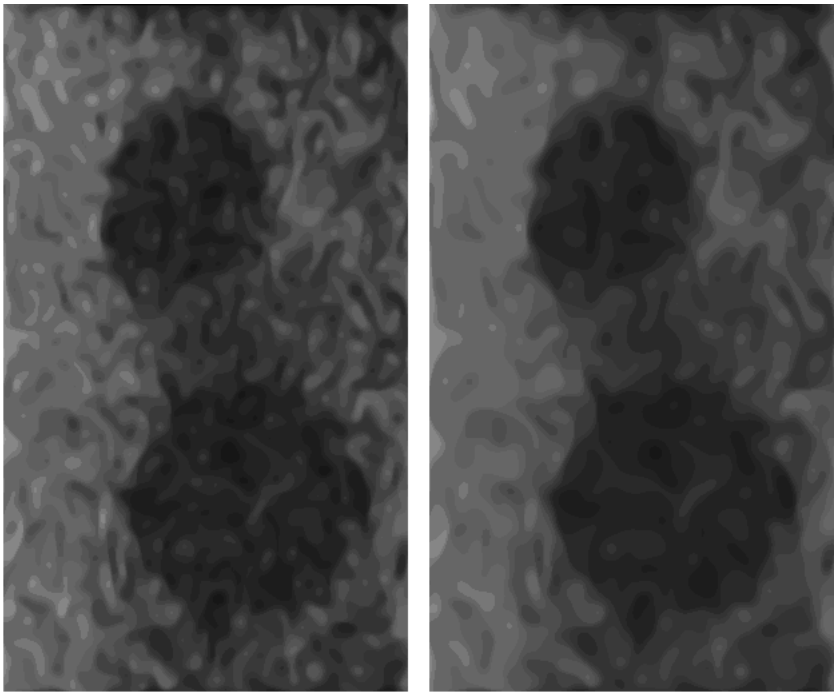

Figure 4.23: *Joyoni Dey's speckle reduction scheme, after processing image from Figure 4.17, with a time step of 0.15. (a) 100 iterations; (b) 200 iterations. The images have been rotated 90 CCW for display purposes.*

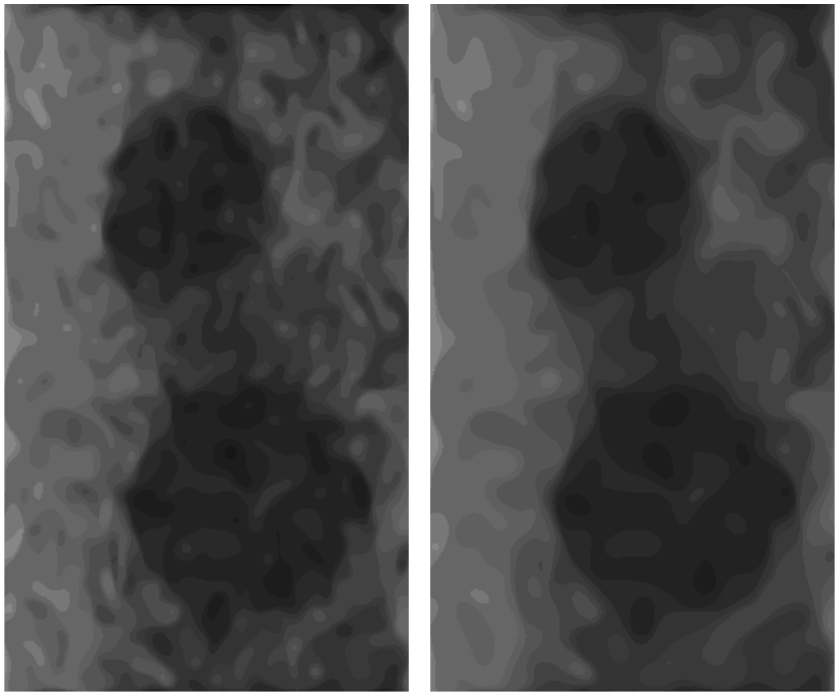

Figure 4.24: *Joyoni Dey's speckle reduction scheme, after processing image from Figure 4.17, with a time step of 0.30. (a) 100 iterations; (b) 200 iterations. The images have been rotated 90 CCW for display purposes.*

curvature of the bowl's surface is smaller than that of the cone because the curves representing that object are not as *sharp*. The point on the edge of the cone has a low curvature, compared to that of the slightly more curved bowl-shaped surface. The tip of the cone has a high curvature because it is a sharp curve.

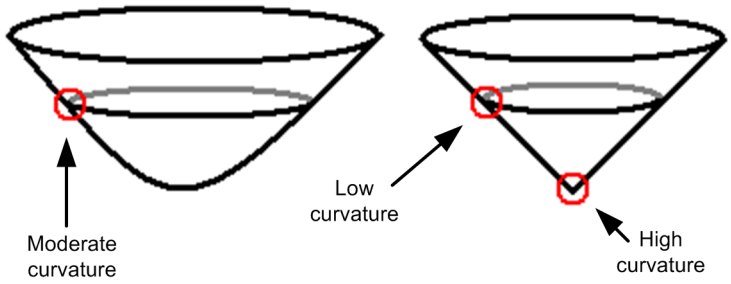

Figure 4.25: *Visualizing curvature strength at various points on two different surfaces.*

This idea of curvature extends to N-dimensions, and in fact every point on the surface of the cone/bowl has a 3D curvature value, defined as the divergence of the unit surface normal.
First, let's consider evolving an image over time according to the following equation:

$$\frac{\partial I(\vec{x})}{\partial t} = \nabla^2 I(\vec{x}) \tag{4.16}$$

For a given 2D image $I(x, y)$, the front $\Psi_i(x, y)$, representing a set of iso-intensities (intensity i) from I, $[\Psi_i(x, y) = \{I(x, y)|I(x, y) = i\}]$, the unit normal vector of that front is defined as [46]:

$$\vec{n_i} = \frac{\nabla \Psi_i}{|\nabla \Psi_i|} \tag{4.17}$$

It follows that the curvature, defined as the divergence of this unit normal vector (4.17), is:

$$\kappa = \nabla \cdot \vec{n_i} = \nabla \cdot \frac{\nabla \Psi_i}{|\nabla \Psi_i|} = \frac{\Psi_{i,xx}\Psi_{i,y}^2 - 2\Psi_{i,x}\Psi_{i,y}\Psi_{i,xy} + \Psi_{i,yy}\Psi_{i,x}^2}{(\Psi_{i,x}^2 + \Psi_{i,y}^2)^{3/2}} \tag{4.18}$$

We now wish to show that the equation given in (4.16) is actually a formulation for the propagation of iso-intensity contours that are proportional to their local curvature, such that sharp peaks with high curvatures evolve (dissipate) quickly, while leaving smoother iso-intensity curves to evolve much more slowly (preservation). Iso-intensity contours can be thought of analogously to the 2D curve that results from slicing the cone or bowl from the previous example, where we are now referring to a common image intensity instead of a common height w.r.t. the physical objects. Indeed, these collections of iso-intensity contours will be explored in the active contours chapter, and is where the term "level sets" is derived from. The image is evolved by an amount proportional to the local curvature of these level sets, given as:

$$\frac{\partial I}{\partial t} = \kappa |\nabla I| \tag{4.19}$$

Note that the example of a single set of iso-contours, $\Psi_i(x, y)$, has been expanded to include the entire image $I(x, y)$, and hence the collection all of the iso-intensity contours. Generalizing to N-dimensions, and substituting for κ given in (4.18), (4.19) becomes:

$$\frac{\partial I}{\partial t} = \nabla \cdot \frac{\nabla I}{|\nabla I|} |\nabla I| = \nabla \cdot \nabla I = \nabla^2 I \tag{4.20}$$

Therefore, (4.16) is a formulation for evolving an image according to the local curvature of the image intensity. The physical interpretation is that small, sharp peaks decay away faster over time while broader, smoother intensity peaks are relatively preserved. Two examples of this filtering process are shown in Figures 4.26 and Figure 4.27, both with a time step of 0.125. Figure 4.26 shows the result after 50 iterations, and Figure 4.27 after 100

iterations. Notice that a lot of noise has been removed, but the prostate boundary is relatively intact.

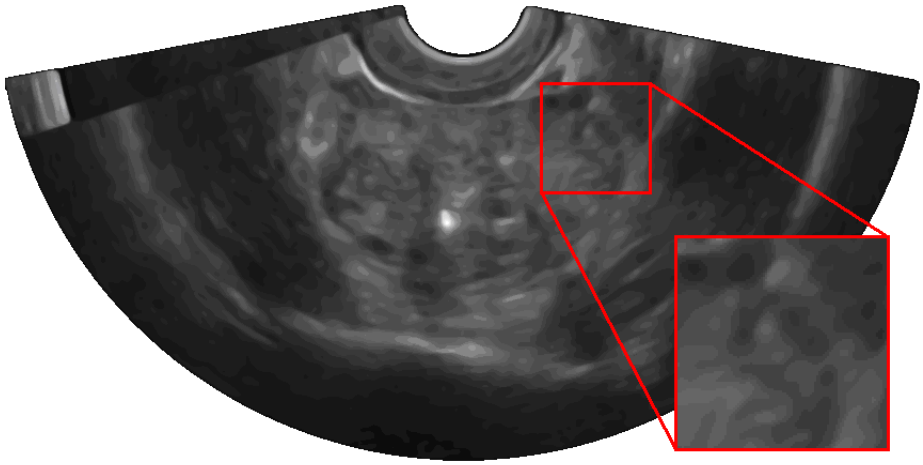

Figure 4.26: *Result of processing Figure 4.21 using curvature flow filtering for 50 iterations with a time step of 0.125.*

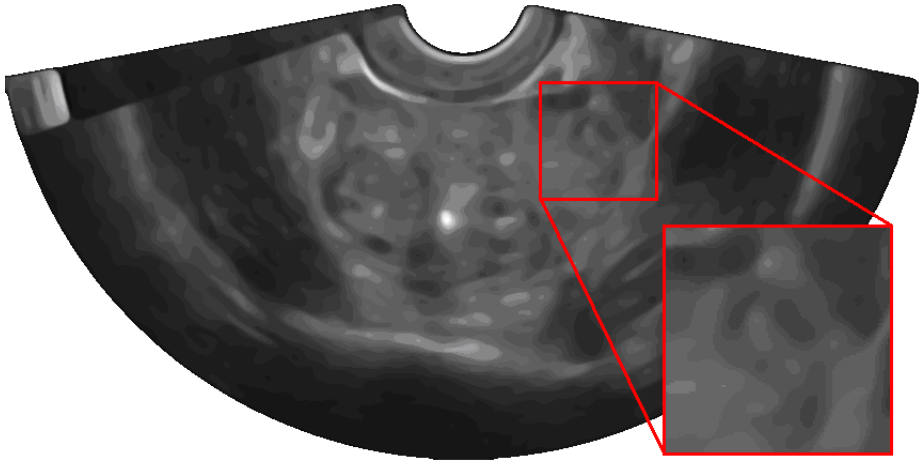

Figure 4.27: *Result of processing Figure 4.21 using curvature flow filtering for 100 iterations with a time step of 0.125.*

Chapter 5

Active Contours

Segmentation is something humans do unconsciously. The eyes send visual data to the brain which, for example, may see a blob of green in the field of vision - that is, a spatially grouped collection of green. Segmentation is recognizing that blob as a single object, or perhaps recognizing the outline of a guitar in a picture. The brain does this by recognizing qualities similar qualities across that object such as color, shape, texture, and contrast versus the qualities of surrounding objects. Still other processes are most certainly at work that we do not understand yet. Classification - the process of recognizing what that object is - would be possibly recognizing that green blob as a tree leaf. This work deals solely with segmentation - the grouping of pixels (voxels) in an image together as a single entity, separate from the rest of the image. The segmentation methods used here rely on edge-based information (gradient magnitude based) and a limited amount of spatial information (curvature forces acting on active contours).

This chapter deals with the segmentation method that was used in this research project to extract features of interest from 3D medical ultrasound images. Active contour representations were created, and various forces, derived from the images (both functions of the evolving surface and the local image intensity information), allowed to influence the deformation of the

surface over time. In 3D, these 2D-counterparts (of contours) are actually *surfaces*. Both cases will be referred to simply as "contours" from here on. The first section in this chapter gives an overview of the earlier Lagrangian method of tracking a curve. There are some rather serious drawbacks and implementation issues with this method, and subsequent sections detail the *Level Set Method* which seeks to overcome these issues. The term 'level set' comes from the fact that in a scalar level set field, the set of points where that field equals zero is the curve under consideration; this will be explained in more detail later. The level set method of tracking an active contour is outlined, along with the general level set equation. Presented next are the various types of forces that are used in the level set equation, in particular the speed image, propagation, curvature and advection forces. The background on signed distance maps, parameters used and finally an example of a level set segmentation are presented.

5.1 Active Contour Introduction

5.1.1 Curvature Evolution

A curve is evolved through time by applying a force, typically in the direction normal to the curve. An example of a curve at two discrete time instances is shown in Figure 5.1. Here, $\mathbf{X}(s,t)$ describes the curve where s is the arclength and t is time. A differential equation is formed that describes the velocity of the propagating front as a function of arclengths location s and time t; this differential equation may be dependent on local information such as an image gradient, local intensity as well as curve parameters such as curvature or rigidity. $\mathbf{X}(s,t)$ can be viewed as a function of both s and t in that for each position s along the curve, an x and y coordinate is specified that maps the curve, and t describes how the curve changes over time. The arrow in Figure 5.1 shows the direction normal ($\vec{n}$) to the curve at a single s location on the curve at time t_0. As will be seen later, this is not the only way to

represent an evolving curve; the level set method evolves a field that has a curve embedded implicitly within it. However, the embedded curve can be viewed as a function of s and t.

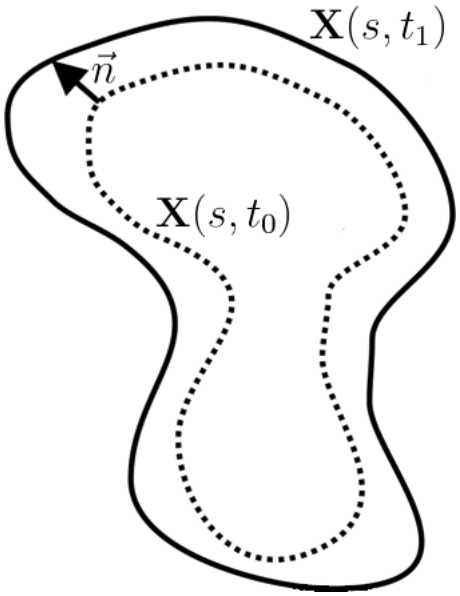

Figure 5.1: *Example of time-varying curve at two discrete time instances.*

5.1.2 Parametric Curves

This section details parametric curves, in order to highlight their drawbacks in traditional marker-and-string methods, which make the alternative level set formulation desirable. The term "marker-and-string methods" refers to the method of tracking a curve by moving sets of points (markers) that have connectivity information (strings) associated with them. The basic parametric active contour framework is described as follows, taken from [46] and [59]. A 2D contour is defined by the *Lagrangian* representation $\mathbf{X}(s,t) = [x(s,t), y(s,t)]$ where $s \in [0, S]$ describes the closed contour, $t \in \Re^+$ is time, and $\mathbf{X}(s,t) \in \Re^N$ for an N-dimensional image. Periodic boundary conditions

state that $\mathbf{X}(s = 0, t) = \mathbf{X}(s = S, t)$, that is, the beginning point and the end point of the contour is the same. A parametric curve is said to be a Lagrangian representation in that the analysis follows the *curve* as it evolves, but the analysis does not follow *every point in space*. Geometric active contours and level sets are said to be *Eulerian* representations, which means that a fixed coordinate system is used and changes are tracked for each location in space (a fixed grid) [59].

A discrete representation of the curve introduces some problems that arise from quantization errors in the discrete domain and the finite difference approximations that must be used for derivatives [45]. Consider a closed curve that has been discretized and is represented by M marker points, which are connected by M line segments of (initially) equal length Δs. Each marker is located along $[0, S]$ at $i\Delta s$, where $i = 1, \ldots M$. The time domain is equivalently discretized into N time segments of duration Δt. A given marker point (x_i, y_i), observed at time $n\Delta t$, is described by the notation (x_i^n, y_i^n), and future time instances are calculated recursively as (x_i^{n+1}, y_i^{n+1}). The forces acting upon these points are typically derived from the 1^{st} and 2^{nd} derivatives of the curve's (points') motion, for which difference approximations are used.

Equation (5.1) can be used to recursively update the curve as it moves through time with some applied force, F, acting along the surface normal:

$$(x_i^{n+1}, y_i^{n+1}) = (x_i^n, y_i^n) + F|\vec{n}|\Delta t \tag{5.1}$$

where the vector pointing in the direction of the surface normal, $\vec{n}$, is given by [45]:

$$\vec{n} = \frac{y_s}{\sqrt{(x_s)^2 + (y_s)^2}}\hat{y} + \frac{-x_s}{\sqrt{(x_s)^2 + (y_s)^2}}\hat{x} \tag{5.2}$$

The central difference approximations given below are used to calculate the derivatives along the curve (with respect to s):

$$x_s = \frac{dx_i^n}{ds} \approx \frac{x_{i+1}^n - x_{i-1}^n}{2\Delta s} \qquad y_s = \frac{dy_i^n}{ds} \approx \frac{y_{i+1}^n - y_{i-1}^n}{2\Delta s}$$
$$x_{ss} = \frac{d^2x_i^n}{ds^2} \approx \frac{x_{i+1}^n - 2x_i^n + x_{i-1}^n}{\Delta s^2} \quad y_{ss} = \frac{d^2y_i^n}{ds^2} \approx \frac{y_{i+1}^n - 2y_i^n + y_{i-1}^n}{\Delta s^2} \tag{5.3}$$

The curvature, κ, is defined as:

$$\kappa = \frac{y_{ss}x_s - x_{ss}y_s}{((x_s)^2 + (y_s)^2)^{3/2}} \tag{5.4}$$

Substitution of the difference approximations (5.3), surface normal (5.2), and curvature (5.4) into the curvature evolution equation, (5.1), yields the recursive evolution equation for some function of the curvature acting along the surface normal:

$$(x_i^{n+1}, y_i^{n+1}) = (x_i^n, y_i^n) + F(\kappa_i^n) \cdot \frac{(y_{i+1}^n - y_{i-1}^n, -(x_{i+1}^n - x_{i-1}^n))}{\sqrt{(x_{i+1}^n - x_{i-1}^n)^2 + (y_{i+1}^n - y_{i-1}^n)^2}} \Delta t \tag{5.5}$$

where $F(\kappa_i^n)$ is some function of the curvature, κ_i^n, which is now defined for each marker and discrete time instance as:

$$\kappa_i^n = 4\frac{(y_{i+1}^n - 2y_i^n + y_{i-1}^n)(x_{i+1}^n - x_{i-1}^n) - (x_{i+1}^n - 2x_i^n + x_{i-1}^n)(y_{i+1}^n - y_{i-1}^n)}{[(x_{i+1}^n - x_{i-1}^n)^2 + (y_{i+1}^n - y_{i-1}^n)^2]^{3/2}} \tag{5.6}$$

Using this concept, a prior shape (initial curve) is evolved over time until the velocity of the curve converges to zero. However, there are some implementation issues which arise because of the discrete approximations. Initially, the markers are equally separated by the arc-length distance Δs. As the curve propagates, small roundoff errors can accumulate in the positions of these points. Moreover, as the points move, the marker-to-marker distances change and can alter the integrity of the difference approximations. As an example, consider two markers that have moved extremely close to one another where a large time step (Δt) was used. At this very small

scale, even small errors in position can lead to large errors in the difference approximations, and hence their velocities. As a result, the markers propagate with slightly incorrect velocities, which in turn create larger errors in position. This feedback cycle can drastically alter the validity of the result, and large oscillations may be present in the final curve. To ensure stability in the front propagation, smaller time steps must be used. In some cases, hundreds of thousands of time iterations may be required to ensure stability, and the evolution becomes computationally infeasible for real time applications. There are some ways around using such a small time step, such as filtering techniques to remove high frequency oscillations along the curve, or techniques which separate closely spaced markers using interpolation. However, the marker positions are still being altered, and these methods are generally undesirable due to their computational load and algorithmic complexity.

The other major limiting factor in the use of parametric active contours is their inability to change topology. The curve remains a single closed contour, and cannot branch into two separate curves. If two curves intersect it would be desirable to unify them as one, but it can be quite complicated to algorithmically decide which markers to keep, which to delete, and how to re-connect the markers when two curves coalesce. This becomes increasingly harder for 3D, (whose 2D equivalent of a curve is a surface), and even more so for higher dimensional surfaces. The level set methods allow segmentation to be done directly in three dimensions, where topological changes are handled automatically.

5.2 Level Set Methods

5.2.1 The Level Set Method

The level set method proposed by Osher and Sethian [36] uses an Eulerian framework to evolve an N dimensional function $\Phi(\vec{x}, t)$ whose zero-level set comprises the N-dimensional (N spatial dimensions, one time dimension)

surface (front) $\Gamma(t)$, defined as:

$$\vec{\Gamma(t)} = \{\vec{x} | \Phi(\vec{x}, t) = 0\} \tag{5.7}$$

By appropriately deforming this field $\Phi(\vec{x}, t)$ over time, we can also deform the curve (the *isolevel* where the field is equal to zero). An example of a 2D field is shown in Figure 5.2(a). Figure 5.2(b) shows the embedded, implicit curve as a thick black line. The convention is to set the field to negative values inside the boundary, zero along the boundary, and positive values outside the closed curve/surface, and is why the fields in Figure 5.2 shows the negative of the field purely for display purposes.

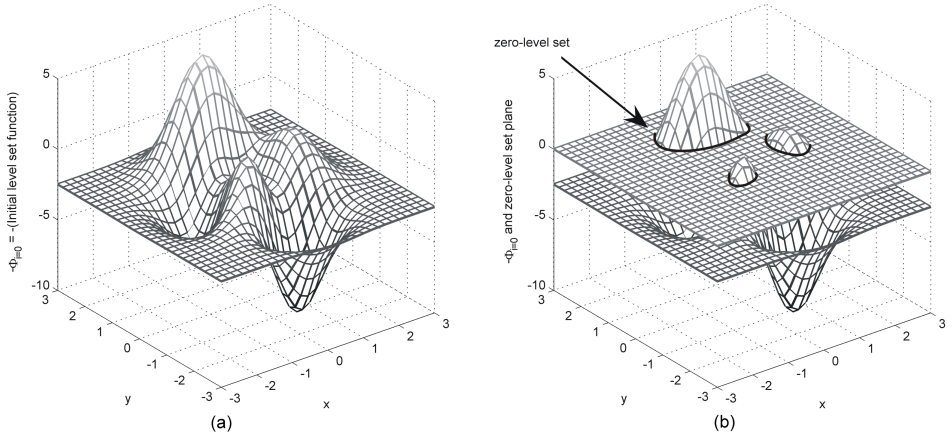

Figure 5.2: *(a) - Example of 2D level set function, $-\Phi(x, y, t_0)$; (b) - The active contour under consideration is embedded as the zero-level set of Φ.*

Defining $\Gamma(t)$ to represent the set of points at which $\Phi(\vec{x}, t) = 0$:

$$\Phi(\Gamma(t), t) = 0 \tag{5.8}$$

We ultimately would like to determine the velocity of Φ, so that we can iteratively evolve it through time. Taking the time derivative and invoking the chain rule yields ($\vec{\Phi_t} \equiv \frac{\partial \Phi}{\partial t}$):

$$\vec{\Phi_t} + \nabla\Phi(\Gamma(t), t) \cdot \vec{\Gamma_t}(t) = 0 \tag{5.9}$$

where the function $\vec{\Gamma_t}$ is the velocity of the curve ($\vec{\Gamma_t} \equiv \frac{\partial\Gamma}{\partial t}$) and is described in terms of forces (F) acting along the surface normal:

$$\vec{\Gamma_t} = F\vec{n} \tag{5.10}$$

F is the applied force that can be a scalar value or a scalar field dependent on curvature and other local properties (such as the image gradient). More details on the forces employed in this work will be given in Section 5.2.2. The function $\Phi(\vec{x}, t)$ is a scalar field that evolves with time. The unit normal for each point in Φ is computed using [46]:

$$\vec{n} = \pm\nabla\Phi/|\nabla\Phi| \tag{5.11}$$

The surface normals for the example field in Figure 5.2(a) are shown in Figure 5.3. This shows the direction in which each surface point will propagate as time is increased, the speed of which is defined by the force (F) pushing in the normal direction. Note that the z-axis has been flipped for better visibility of the surface normals.

Following the discussion in [46] and [27], the final result for the level set evolution is given below:

$$\vec{\Phi_t} + F|\nabla\Phi| = 0 \Rightarrow \vec{\Phi_t} = -F|\nabla\Phi| \tag{5.12}$$

Yielding the following iterative method for calculating the subsequent iterations of Φ, given $\Phi(\vec{x}, t_0)$:

$$\Phi_{i+1} = \Phi_i - F|\nabla\Phi_i|\Delta t \tag{5.13}$$

Δt is a time step value that must be small enough to ensure stability. This value determines how quickly the field is deformed for each iteration, so a

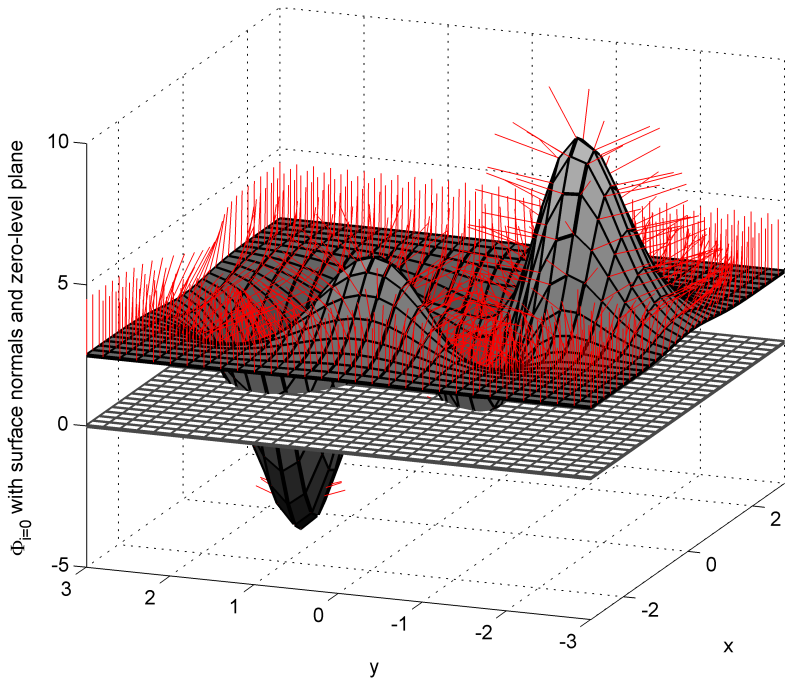

Figure 5.3: *Surface normals for example field in Figure 5.2.*

large time step is desirable to achieve convergence in the fewest iterations possible, while a smaller value is desired for stability. The stability threshold is generally dependent on the dimensionality of the field under consideration. The typical stability threshold is given as:

$$\Delta t < \frac{1}{2^N} \tag{5.14}$$

where N is the number of field dimensions (not including the time dimension). For 2D images, $\Delta t < 0.25$, and for 3D images, $\Delta t < 0.125$. In practice, a variable time step was used to speed the segmentation up as much as possible while maintaining stability. To achieve this, the largest possible (optimal) time step on a per-iteration basis was calculated that would not advance the

curve by more than one voxel.

Note that in this level set framework, individual curves are not tracked or evolved explicitly, but rather, implicitly. The surface is evolved in the direction of the surface normal, and the many curves are implicitly embedded as level sets of Φ. In this way, the curve parameters can be calculated from the evolved surface, but are not required to advance the surface evolution.

By using level sets, many problems with tracking a Lagrangian curve are eliminated. One of the most powerful advantages, however, lies in the topological flexibility offered by the level set methods. As was mentioned in the previous section (5.1.2), it can be very difficult to handle topological changes with parametric curves, such as when two surfaces grow until the intersect (merge), or one surface pinches off into two separate ones (splits). However, these problems are eliminated with the level set framework. Shown in Figure 5.4(b) is the same field as Figure 5.2(b), but advanced one iteration ($i = 1$). The two back curves have been implicitly merged into one larger curve, without having to worry about combining two sets of connected markers (as in a Lagrangian formulation). This would involve determining which markers to delete, and how to best re-connect those markers that were left. This idea extends to the merging of 3D surfaces which would be a much more difficult problem with markers and associated connectedness information. In this way, the use of level sets allows one to use a single, smooth surface whose embedded topology at the zero level set can be very complicated and allows curves to split apart or combine together in ways that would be infeasible to track with parametric curve representations.

5.2.2 Level Set Forces

The level set methods presented thus far still suffer from problems in noisy medical images, including MRI (magnetic resonance imaging), CT (computed tomography a.k.a. "CAT" scan) and especially ultrasound images. Stopping terms are spatially varying modifiers that influence the local curve propaga-

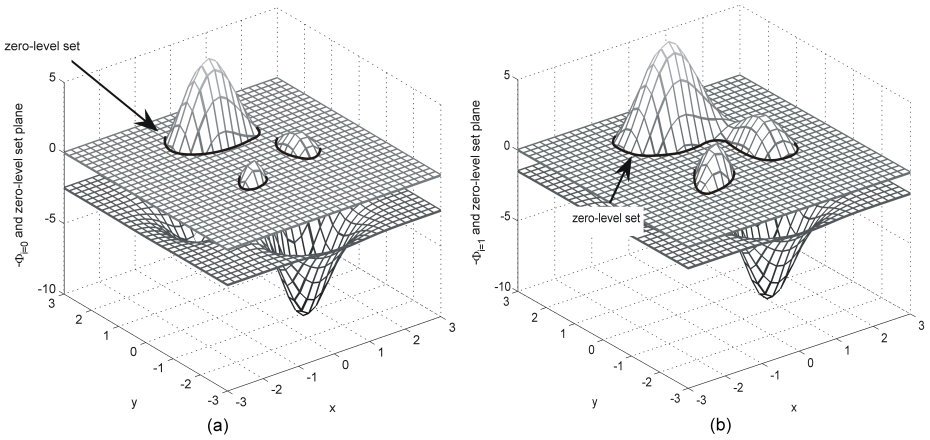

Figure 5.4: *(a) Example field from Figure 5.2(b); (b) - Same field as Figure 5.2(b), advanced one iteration. Note that the two back curves from Figure 5.2(b) have merged into one.*

tion. For example, an inverse function of the image gradient magnitude is used to slow the curve at regions where strong boundaries are present. The stopping terms are many times troublesome in that noise and artifacts can trick the curve into stopping where it should not. Also, low contrast regions or breaks along the boundaries can allow the surface to leak through. Several methods have been proposed to solve this problem, although real time implementation can still pose many problems due to the enormous amount of required processing.

The generic level set equation governing motion given by [46] is:

$$\vec{\Phi}_t + \alpha\vec{A}(\vec{x}) \cdot \nabla\Phi + \beta P(\vec{x})|\nabla\Phi| = \gamma Z(\vec{x})\kappa|\nabla\Phi| \tag{5.15}$$

where $\vec{A}(\vec{x})$ is the advection force, $P(\vec{x})$ is the propagation (or balloon) force, and κ is the curvature of the local surface. The advection force is a vector field that pushes the curve in the direction of the gradient of $\nabla\Phi$. $Z(\vec{x})$ is a spatial modifier term applied to the curvature force. The three scalars α, β, and γ are used to weight the effect of each type of force. The curvature κ is defined by (5.16) and (5.17) (2D or 3D), and is similar to the Lagrangian

definition from (5.4). The curvature term is used as a smoothing term that is defined (here, in two dimensions) as the divergence of the unit normal [36] [49]:

$$\kappa = \nabla \cdot \frac{\nabla \Phi}{|\nabla \Phi|} = \frac{\nabla^2 \Phi}{|\nabla \Phi|} = \frac{\Phi_{xx}\Phi_y^2 - 2\Phi_x\Phi_y\Phi_{xy} - \Phi_{yy}\Phi_x^2}{(\Phi_x^2 + \Phi_y^2)^{3/2}} \tag{5.16}$$

The curvature in 3D is given as [46]:

$$\kappa = \begin{array}{l} \frac{(\Phi_{yy}+\Phi_{zz})\Phi_x^2+(\Phi_{xx}+\Phi_{zz})\Phi_y^2+(\Phi_{xx}+\Phi_{yy})\Phi_z^2}{(\Phi_x^2+\Phi_y^2+\Phi_z^2)^{3/2}} - \\ \frac{2\Phi_x\Phi_y\Phi_{xy}+2\Phi_x\Phi_z\Phi_{zz}+2\Phi_y\Phi_z\Phi_{yz}}{(\Phi_x^2+\Phi_y^2+\Phi_z^2)^{3/2}} \end{array} \tag{5.17}$$

The curvature values for the field given in Figure 5.2(a) are shown in Figure 5.5(b). Notice that the field's curvature values are very high near sharp peaks and generally small for smoother regions. This acts to evolve the field more quickly if it has a high curvature in that region, and smooth the boundary curve faster. As a result, the curve tends to remains more round and smooth for higher values of γ, which is the curvature strength coefficient in the level set equation (5.15). This acts as a surface tension force that helps bridge weak gaps in the object boundaries by preventing the curve from leaking through.

This speed image is a scalar field that has a multiplicative influence on the speed of the curve at each location in the field. The goal is to slow the curve when near or crossing a boundary, and to increase the curve velocity when in a homogeneous region of non-interest (non-boundary). Following the implementation in [4], a speed image is used as the spatial modifier for both the curvature and propagation forces:

$$C(\vec{x}) = P(\vec{x}) = Z(\vec{x}) \tag{5.18}$$

Different fields may be used for $P(\vec{x})$ and $Z(\vec{x})$ in other level set applications (such as fluid transport in a fluid simulation), but for image segmentation an appropriate speed image will reduce both the propagation and curvature

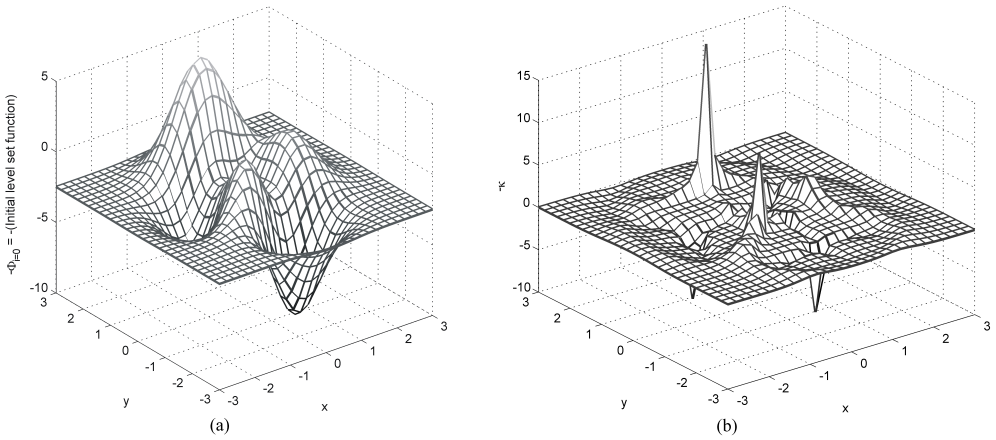

Figure 5.5: *(a) - Example field (Figure 5.2(a)); (b) - 2D curvature values of field from (a).*

forces sufficiently to slow (stop) the curve. The advection force, $\vec{A}(\vec{x})$ is a vector field constructed from the negative gradient of the speed image:

$$\vec{A}(\vec{x}) = -\nabla C(\vec{x}) \tag{5.19}$$

The goal of the speed function $C(\vec{x})$ is to slow the curve at the boundaries while quickly propagating the curve through regions of non-interest. This is constructed by taking the gradient magnitude of the speckle-reduced image volume and applying an inverse transformation function. The sigmoid function was used to map high gradient magnitude values to low speed values, and vice versa. The speed function (using a sigmoid function) is given as:

$$C(\vec{x}) = [1 + exp[-\frac{|\nabla I(\vec{x})| - \beta_S}{\alpha_S}]]^{-1} \tag{5.20}$$

where α_S and β_S are user-supplied parameters indicating the dropoff intensity level and the sharpness of the dropoff. Note that these do not have any connection to the α and β from (5.15). For the collection of ultrasound images, it was found that setting $\alpha_S = -5$ and $\beta_S = 10$ yielded speed images with sufficient stopping power to accurately stop the curve. Note that α_S

is set to a negative value to provide the necessary inverse intensity mapping (high gradient magnitudes to low speed values). Also note that the entire bit-range is not utilized for the input gradient magnitude with the sigmoid transformation function - the gradient strengths rarely occupy much more than the lower range (weak image contrast). The transformation function is graphed in Figure 5.6.

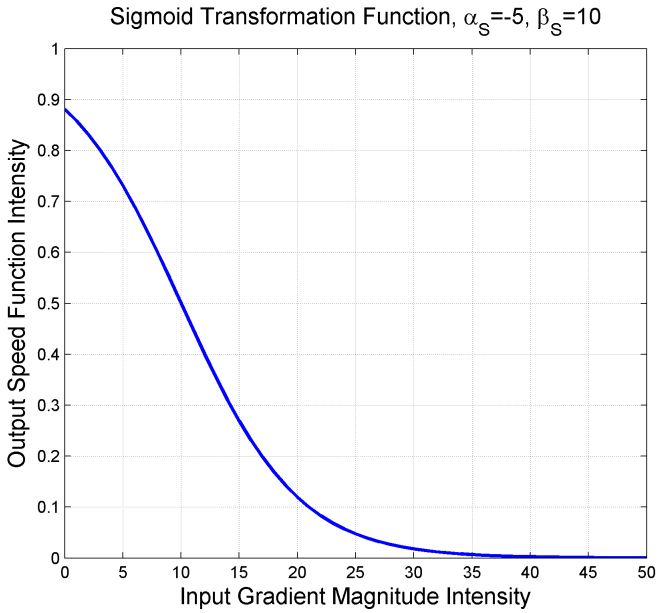

Figure 5.6: *Sigmoid transformation function for mapping gradient magnitude of speckle-reduced image to speed image.*

An example of these operations can be seen in Figure 5.7. Figure 5.7(a) shows one of the 3D, isometric, simulated images after speckle reduction (anisotropic diffusion), and Figure 5.7(b) the gradient magnitude. Note that the Field-II generated image has one large cyst and 4 smaller ones, but only the large cyst ended up being used as a target. Figure 5.7(c) shows the speed image, obtained by applying the sigmoid transformation function to the gradient magnitude. Note that there are regions of low speed (dark pixels) surrounding the targets of interest, along the cysts' boundaries. For

an initial sphere (or any arbitrary initialization shape) placed within the large cyst, a positive propagation weight value (β) will inflate the surface. The speed image, which has high values (near 1.0) *inside* the cyst, will allow the curve to inflate without much resistance. As the surface expands and reaches the black circle surrounding the cyst (*boundary*), the speed image will work to oppose the inflation force, and slow the surface down. The goal of a good speed image is one that forces the surface to stop at these boundaries of interest.

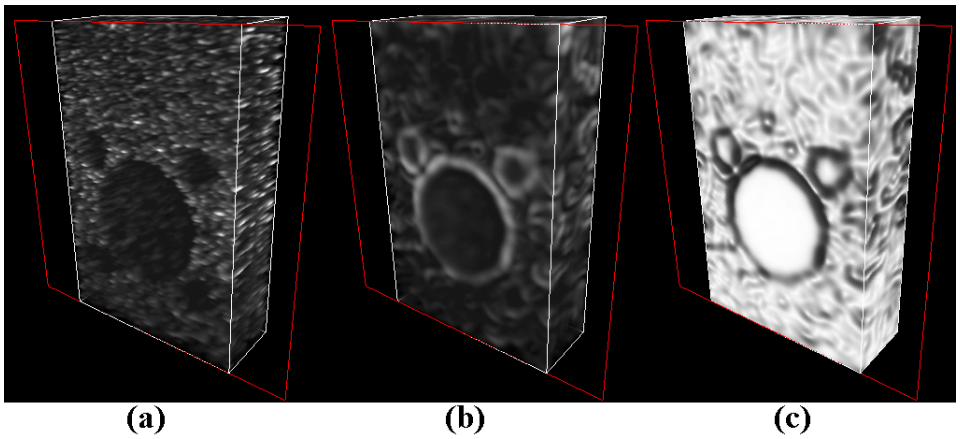

Figure 5.7: *Speed image creation. (a) - Speckle reduced simulated image; (b) - Gradient magnitude; (c) - Speed image (sigmoid transformation applied to gradient magnitude). Red indicates cutaway (clipping plane).*

As seen in Figure 5.7(c), the speed image has a few small gaps, and the dark boundary does not completely capture the boundaries of the smaller cysts near the bottom of the volume. The curvature force acts like a surface tension that will "hold" the curve from leaking out the small gaps in the boundary.

5.2.3 Level Set Initialization - Signed Distance Maps

The first iteration of a level set evolution is initialized as a *signed distance map* whose zero-level set (isosurface) comprises the "best" guess at where the target boundaries lie, or at least a starting point for the contour. A signed distance map (*distance map* for short) is constructed from the initial boundary (guess) and is used as the first iteration of Φ, $\Phi(\vec{x}, t_0)$. In practice, distance maps were constructed from spheres, manually placed inside the target boundaries and of interior voxel magnitude one (zero elsewhere), using a custom program. The distance map is negatively valued inside the object(s), zero on the boundary, and positive outside of the object. The values of the distance map at each voxel are defined as the length of the path from that point to the *closest* point on the initial boundary ($\Phi = 0$). As a result, points that are far away from the boundary have a large (positive or negative) value, and these peaks would take a lot of deformation through evolution to swap the regional classification (inside or outside). Another perspective on the same process is that the curve ($\Phi = 0$) would take longer to propagate to that point (with a single force in the normal direction).

The example given in Figure 5.8 and Figure 5.9 shows a boundary and the associated signed distance map function. Figure 5.8 shows several arbitrary objects in a 2D image. White pixels represent the object and the interface between white and black pixels represents the boundary, or initial surface that is to be deformed using the level set method. The signed distance map shown in Figure 5.9 has been computed from the initial surface, and this scalar field is used as the first level set ($\Phi_{i=0}$). The pixels inside the objects are dark (negatively valued), and get darker as the distance from the boundary increases. Outside the objects, the pixels are light (positive), and get lighter as the distance from the boundary increases. It is important to note that the object borders in Figure 5.9 have been artificially lightened for display purposes; in reality, the object borders in the distance map have field values equal to zero, which would be some shade of gray in this image.

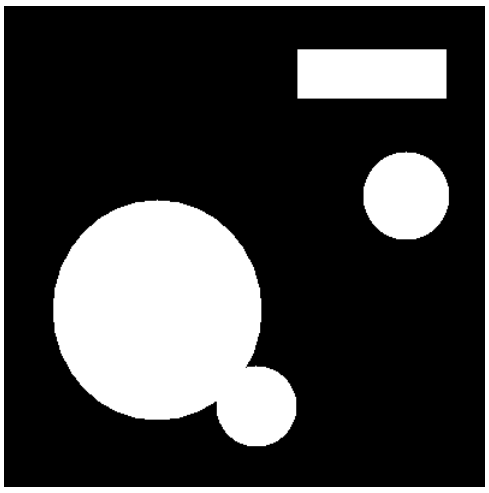

Figure 5.8: *Initialization volume (initial boundary) with several arbitrarily placed objects. White pixels represent the interior of the object, while the boundary encapsulating the white pixels represents the surface to be deformed.*

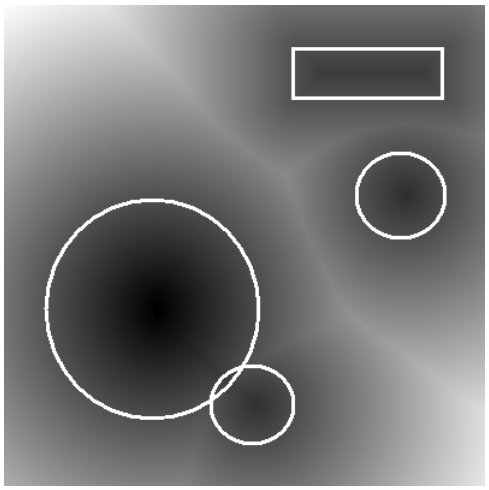

Figure 5.9: *Signed distance map, to be used as the first iteration of the level set field. Note the dark (negative) values inside the objects from Figure 5.8, and positive (lighter) values the further from the boundary on the exterior. Note that the object borders have been artificially lightened for display purposes.*

5.2.4 Evolution Parameters

Appropriate values for α, β and γ in (5.15) were chosen through some trial and error to yield reasonbly accurate segmentations across all images - simulated, phantoms, and clinical. α was set to 1.0 - this was sufficient to attract the surface to the boundaries and hold them there. Values less than 0.9 or higher than 1.2 did not seem to hold the surface at the target boundaries, resulting in much less accurate results. β was set to 0.3, as the level set was initialized inside the targets of interest and an expanding force needed to be used. Higher values of β, between 0.5 and 1.0, were too powerful and the curves would expand past the target boundaries. Smaller values, less than 0.2, were not sufficient to expand the surface fast enough without taking an excessive number of iterations (over 2000). γ was set to 1.5, as this was sufficient to keep the surface from leaking through the speed image boundary gaps while allowing the surface to take on shapes with enough detail. Higher values of γ led to very smooth, round, and inaccurate segmentations, and lower values let the surface leak throughout the weak object boundaries.

These global parameter values were chosen so that the different speckle reduction methods could be compared across all images using the same segmentation method. As will be shown in Section 8.4, there is a certain amount of fine tuning of the level set parameters that can be done for each image type to achieve the best results. For example, for images with weak boundaries (such as the low contrast phantom images, 55% graphite), too strong of a balloon force (β) will push the contour past the weak boundaries. For images with strong boundaries (simulated images), this balloon force can be higher to achieve convergence in fewer iterations without risking the contour pushing past the boundaries. For the box phantoms, a lower curvature strength coefficient (γ) will allow the contour to grow out closer into the sharp corners, whereas the boundary gaps in the clinical images require a higher γ value to prevent leakage. For the best results, future work might focus on finding the best combinations of level set parameters for given image/organ types under

consideration.

5.2.5 Evolution Example

An example of a cyst segmentation using the level set method is now described. The original image can be seen in Figure 5.10(a), and the corresponding ground truth volume in Figure 5.10(b). This was a Field-II simulated cyst image volume (close to spherical in shape), which was initialized with 4 spheres for this example to purposely display the automatic topological merging, and can be seen in Figure 5.11(a). Figure 5.11(b) shows the surface representing the zero-level set of the level set function after 110 iterations have passed. Notice how the spheres are expanding due to the outward propagation force and the spheres have merged together into one large expanding surface. Figure 5.11(c) shows the segmentation after 330 iterations; notice how the bottom right of the image shows the surface approaching the boundary and sticking there, while the top left is still expanding and filling the rest of the spherical cyst. Figure 5.11(d) shows the final result, after 1000 iterations have elapsed. [62]

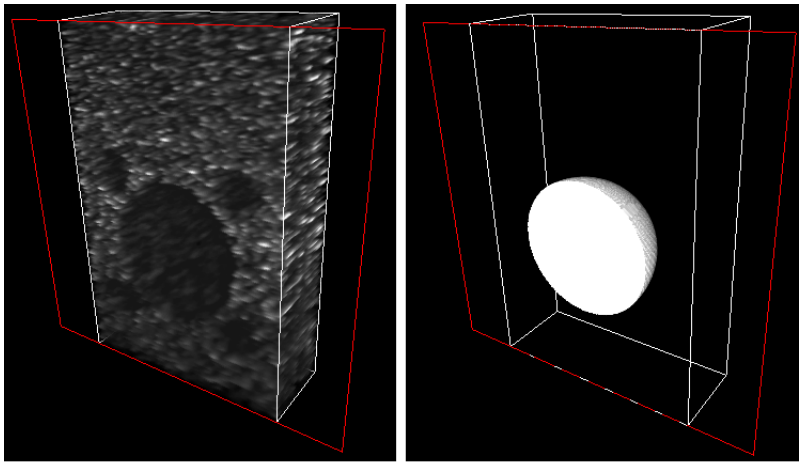

Figure 5.10: *(a) Field-II simulated spherical cyst, with cutway; (b) Ground truth volume, with cutway.*

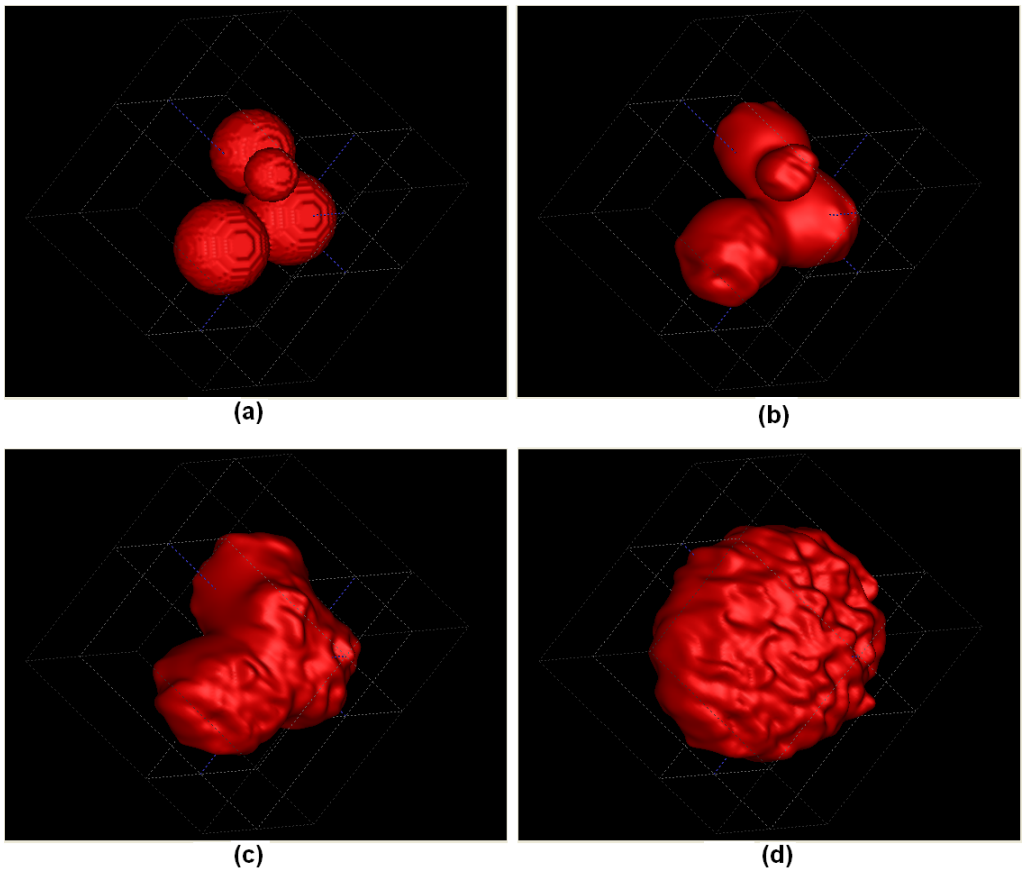

Figure 5.11: *Level set evolution segmentation example, of a simulated spherical cyst. (a) Initial level set; (b) - 110 iterations; (c) - 330 iterations; (d) - 1000 iterations, convergence reached.*

Chapter 6

Ground Truth Models and Performance Metrics

In order to quantify the performance of the segmentations, two metrics were used to calculate the accuracy of the results versus the ground truth - a volume-based accuracy measurement metric one and a surface-based accuracy measurement metric. The ground truths for the simulated images were known exactly because the cyst sizes and locations were specified exactly. The ground truth models for the real prostate scans were based on hand-segmentations from two doctors, a Radiation Oncologist and an M.D. practicing Internal Medicine at the University of Massachusetts Medical School, so both exact sizes and locations were also known. For both the simulated and clinical data, exact voxel-to-voxel correspondences were known between the images and their ground truth counterparts. However, while the exact dimensions of the phantom images were known, their exact locations were not. The phantoms were placed into the tissue-mimicking material while it was still soft (a necessary step), which allowed the targets to move slightly. In order to correctly evaluate the accuracy of the surface reconstruction of the phantom images, before the surface-based accuracy metric was used on the phantom images, the phantom segmentation results were first aligned with

the ground truth models using the ICP (Iterative Closest Point) algorithm, detailed later in Section 6.4.

6.1 Generation of Ground Truth Models

To generate the ground truth models for the simulated images, first an empty image volume of zeroes, of the same image size and voxel dimensions as the scanned image, was constructed. A program using the ITK toolkit was written and used to place spheres inside the volume at the exact locations and with the exact dimensions that were used to generate the simulated image volume to begin with. These spheres had voxel values of 1 for voxels along the boundary, and also for the interior of the objects. One example of a ground truth volume for a Field-II simulated image is shown in Figure 6.1. Note that the sphere is comprised entirely of voxels with intensity 1.0 (or voxel value of 1.0), and the shading along the sphere surface is purely for display purposes.

Similarly, a custom ITK program was used to generate volumes with the rectilinear box shapes and cylinders that corresponded with the targets created in the lab. While the exact dimensions were known, the exact locations of the ground truth models with respect to the scanned images (and hence segmented results) were not. As such, the locations of the ground truth models do not correspond with the locations of the object in the 3D ultrasound images. This comes into play especially in the surface error metric, and a solution to this problem is presented later in this chapter. An example of a ground truth volume for one of the cylinder phantoms is shown in Figure 6.2.

The ground truth models for the real prostate scans were based on hand segmentations by two doctors at the University of Massachusetts Medical School, Dr. Saroj Bharitaya, M.D., Internal Medicine, and Dr. Mark Smyczynski, M.D., Radiation Oncology. They were kind enough to spend the time to hand-outline the prostate boundaries in the three real image vol-

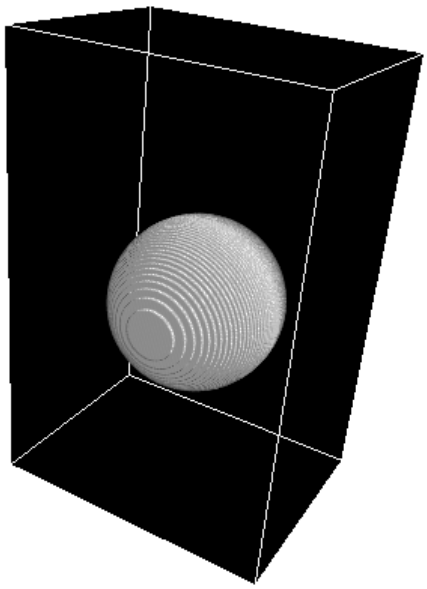

Figure 6.1: *Ground truth image volume for the corresponding Field-II simulated spherical cyst.*

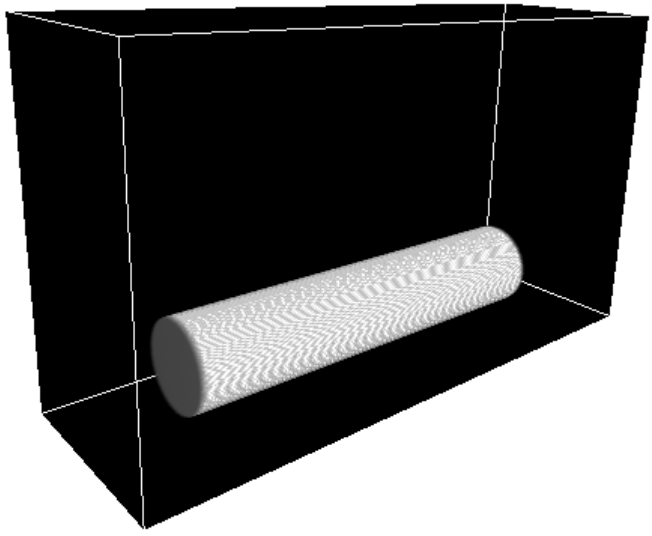

Figure 6.2: *Ground truth image volume for cylindrical cyst-mimicking phantom. Exact dimensions for phantom images are known, but exact locations are not.*

umes. Each volume was hand-outlined twice, once by each doctor. The ground truth models were created by segmenting each 2D scan plane. However, the real volumes had between 344 and 352 scan planes each, so to reduce the amount of time necessary to hand-segment them, only every 10th scan plane was hand-segmented. This resulted in 34-35 scan planes for each volume (not every one actually contained a piece of the target object, hence the hand-segmented prostate image volumes are made up of fewer than 34-35 scan planes). The depth for each scan plane was 0.154 mm in the original volumes, giving a thickness of 1.54 mm for each 2D hand-segmented scan plane. Two hand-segmented results are presented in Figure 6.3. Notice that there is a certain amount of scan-plane to scan-plane discrepancy, even from a medical professional analyzing successive scan planes (it is assumed that the true prostate boundary is smoother than this).

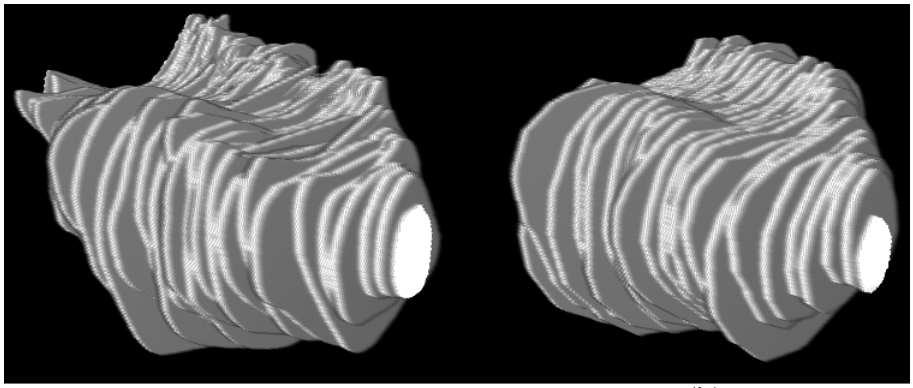

Figure 6.3: *Two hand-segmented ground truth models of the same real patient prostate volume, created by two different doctors at the University of Massachusetts Medical School. Note a certain amount of scan-plane-to-scan-plane human error, as well as doctor-to-doctor discrepancy for the same image volume.*

6.1.1 Clinical Smoothing

The hand-segmented images shown in Figure 6.3 have rather jagged edges, which cannot be truly representative of the actual anatomical structure. Instead, this is likely due to the fact that the volumes were outlined one 2D scan plane at a time, hence the low scan plane to scan plane correlation. To obtain more accurate ground truth models, the hand segmented images were subjected a smoothing operation. First, an appropriate smoothing operation had to be decided upon. For each clinical image, there was a certain amount of discrepancy between the two doctors' ground truths, according to the volume and surface error metrics. If the smoothed versions of each doctors' segmentation had increased their similarity as compared to the non-smoothed versions, it is argued that human error had been reduced and a more true representation of the anantomical object had been achieved.

A binary morphological opening operation was used to smooth the clinical hand-segmented ground truth image volumes [47]. A morphological opening is defined as an erosion followed by a dilation. A structuring element defines the shape of the surrounding neighborhood of voxels that are evaluated for a given pixel under consideration. Every voxel that has a value of 1.0 is considered. An erosion works by stepping through each pixel with value 1.0, and if every pixel in that neihborhood has value 1.0, the central pixel retains its value. If any of the surrounding neighborhood voxels do not have value 1.0, the central pixel is set to zero. This reduces jagged edges and removes small artifacts if any are present. The dilation works similarly, by traversing all pixels with value 1.0 and setting all of the neighborhood pixels to 1.0. The dilation works to increase the mass of voxels with value 1.0, and results in a surface that is enlarged and smoother. By performing an opening, artifacts and jagged edges are reduced in the erosion phase, and the surface is restored to nearly its original size using the dilation phase, but in a much smoother form. A ball shaped structuring element was used with varying radius sizes. As the hand segmented images have a voxel spacing along the x-dimension

approximately 10 times the y and z, the ball radii have x sizes 10 times the y and z sizes. Originally, the clinical volumes were nearly isotropic (equal voxel spacing). To save time, the doctors only hand-outlined every 10^th scan plane, resulting in ground truth volumes with voxel spacing along the x-direction 10 times larger than the original images. The different smoothing types were defined as:

Smoothing Type 1: [x, y, z] = [1, 10, 10] (less smoothing)
Smoothing Type 2: [x, y, z] = [2, 20, 20]
Smoothing Type 3: [x, y, z] = [3, 30, 30] (more smoothing)

Table 6.1 describes the results found by applying each type of smoothing:

Table 6.1: *Results of using varying smoothing methods on clinical ground truth volumes.*

	Input 1	Input 2	Vol Discrep (%)	Surf Discrep (RMS)
Image 1	Doc1	Doc2	9.27	1.68
	Doc1,ST1	Doc2,ST1	10.23	1.89
	Doc1,ST2	Doc2,ST2	10.68	1.94
	Doc1,ST3	Doc2,ST3	11.04	1.98
Image 2	Doc1	Doc2	3.16	2.88
	Doc1,ST1	Doc2,ST1	0.93	2.96
	Doc1,ST2	Doc2,ST2	0.38	2.11
	Doc1,ST3	Doc2,ST3	0.34	2.01
Image 3	Doc1	Doc2	3.32	2.80
	Doc1,ST1	Doc2,ST1	1.47	2.93
	Doc1,ST2	Doc2,ST2	0.05	2.80
	Doc1,ST3	Doc2,ST3	3.17	1.79

As can be see from the table, it appears that the smoothing does tend to reduce the discrepancy between the two doctors with respect to the volume and surface error metrics. For Image 1, all of the smoothing options actually increased the discrepancy, however only slightly. For Images 2 and 3, the

discrepancy was reduced, and in some cases, noticeably. This would imply that these smoothing operations did indeed reduce the human error present in the original hand-outlined images, and that they more accurately represent the true anatomical structures represented by the ultrasound images.

In addition to the confidence given by these metrics that the smoothed ground truth models are truly more accurate, visual inspection of the image volumes agrees with these findings. The smoothed versions do not have the jagged edges in the hand-outlines images, and one might reasonably assume that the true anatomical features are in fact smoother than those seen in the original ground truth volumes. Shown below in Figures 6.4 and 6.5 are examples of these smoothing operations on two ground truth volumes from the same image (image 3). Figure 6.4(a) shows the original hand-outlined image for doctor 1. Figure 6.4(b-d) show smoothing types 1-3. Note that the smoothed version have much less jagged surfaces and appear to be truer to the anatomical structure one might expect.

From the results in Table 6.1, Figure 6.4 and Figure 6.5, it was decided to use smoothing option 2, a binary morphological opening operation using a ball-shaped structuring element of radius [2, 20, 20]. Except for image 1, this method reduced the doctor-to-doctor discrepancy, and in the case of image 1, only slightly increased it. Upon visual inspection, smoothing option 2 appears to provide a more accurate anatomical model.

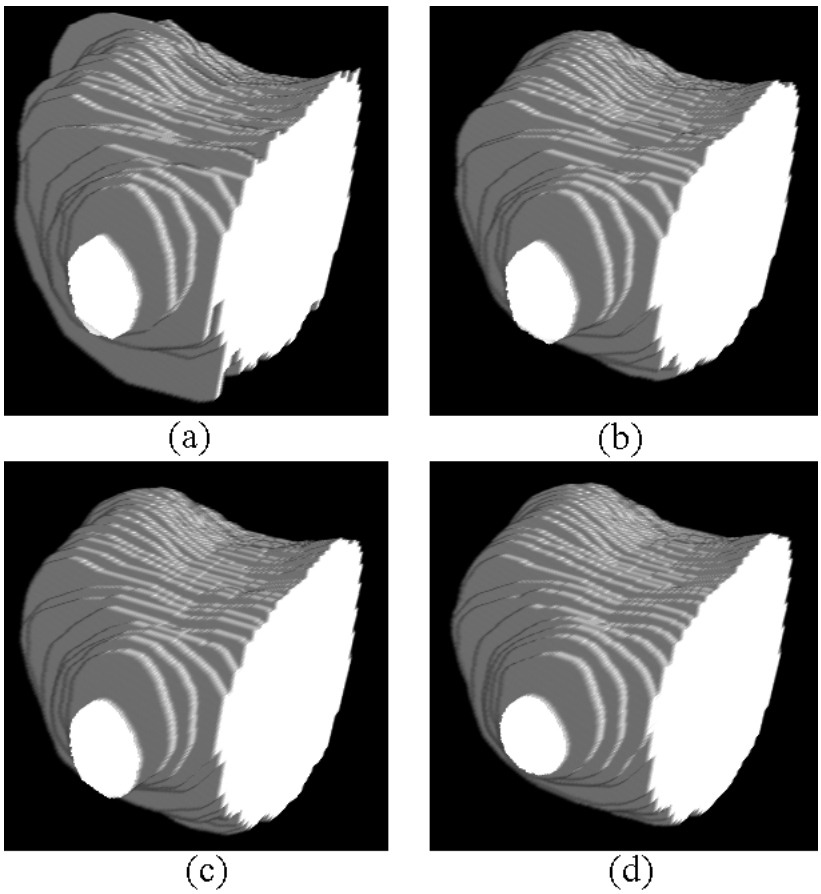

Figure 6.4: *Clinical image 3 ground truth, doctor 1, with cutways. (a) - Original; (b) - smoothing type 1; (c) - smoothing type 2; (d) - smoothing type 3.*

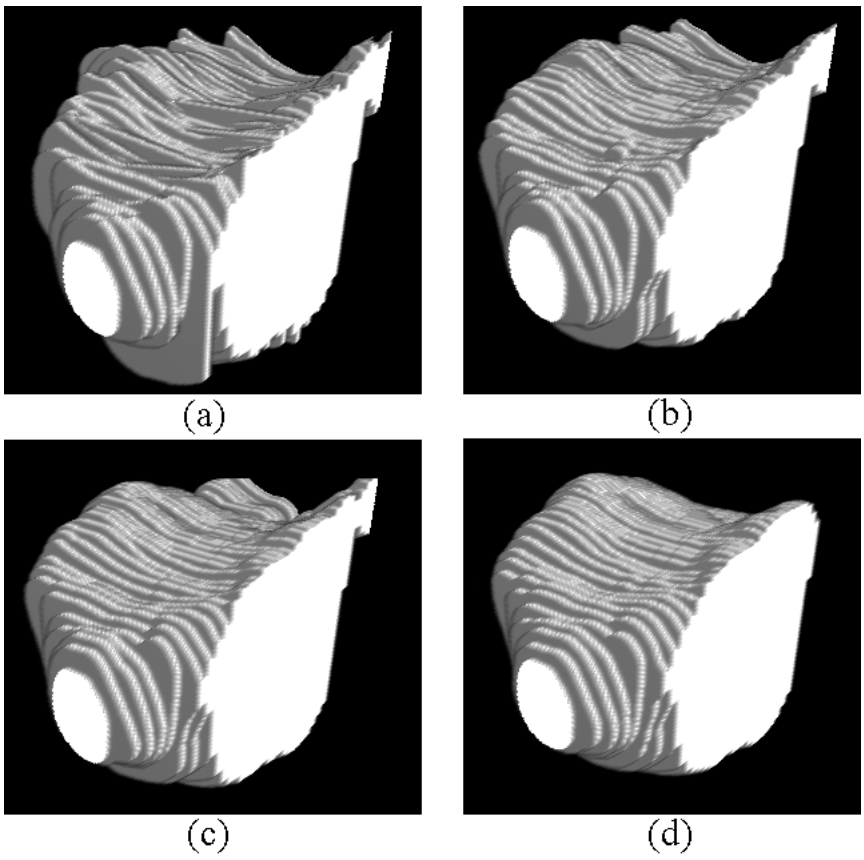

Figure 6.5: *Clinical image 3 ground truth, doctor 2, with cutways. (a) - Original; (b) - smoothing type 1; (c) - smoothing type 2; (d) - smoothing type 3.*

6.2 Volume Error Metric

The first of two metrics used to quantify the segmentation performance is the volume error metric. The volume of the segmentation result is compared to the volume of the ground truth model. First, the volume per voxel is calculated from the voxel spacing (see Chapter 3 for voxel spacing information for each image):

$$Vol_{vox} = \Delta x \cdot \Delta y \cdot \Delta z \ [\text{mm}^3] \tag{6.1}$$

The volume for a single result is found by multiplying the number of voxels inside the object by the volume per voxel:

$$Vol = N_{vox} Vol_{vox} \ [\text{mm}^3] \tag{6.2}$$

where N_{vox} is the number of voxels for a given target object. This volume is computed for both the ground truth model and for the segmentation results. The volume error metric (as a percent error) is thus computed as:

$$Err = 100 \cdot \frac{Vol_{gtruth} - Vol_{seg}}{Vol_{gtruth}} \ [\%] \tag{6.3}$$

6.3 Surface Error Metric

The volume error measurement does not contain any shape information. As a result, two shapes with very different shapes may be found by the volume error metric to very similar because the volumes are similar. To more accurately evaluate the match between the segmented image and the ground truth, a surface error metric was developed to quantify the shape accuracy as well. The idea is that when two surfaces are compared, the RMS (root mean square) distance between the two surfaces is computed, where a small value indicates that the surfaces are very close to one another.

To implement this, first the two results are converted to point sets, i.e.,

the ground truth model and the segmentation result. A program developed in ITK accomplishes the task of extracting the boundary locations from a binary volume and storing them as $[x, y, z]$ coordinates. The coordinate of each point is taken as the center of the face of two adjoining voxels of differing intensity (zero vs. one). The two point sets (ground truth G with J points, and the segmentation result S with I points, where generally $I \neq G$) are then fed into the surface error metric.

Every point in the segmentation result S is traversed. This is important because no point-to-point correspondence exists between the two sets. Also, the two sets will almost always have a different number of points. For each point in S, the distance to *all* points in G are calculated, the distance to the *closest* point in G is recorded. This is tabulated for each point in S, and the collection of distances to the closest neighbors in G used to determine the similarity of shape (Note that for the phantom images, this is done *after* the alignment procedure - which is outlined in the next section). This could have been done in a similar fashion by traversing every point in G, but if there were some outlying points in S the distances to those points might not ever be recorded, because they never qualify for the closest point to one in G. By traversing S, we are assured that every point in the segmentation result will be covered in the metric.

An example of two point sets and the distances to their closest neighbors in the other sets is shown in Figure 6.6. G is the ground truth set, represented by the dark circles, with a solid line connecting them for display purposes. As mentioned previously, every point in S is traversed - not every point in G ends up being used for the metric. The points in G that are not used are shown in gray - because no member of S found G_2, G_3, or G_4 to be their closest neighbor. S is the segmented result represented by the dark circles with dashed lines connecting them. Note that G and S are purely point sets and do not have any connectedness information associated with them, as implied by the lines connecting the points. For each point in S, a dashed line

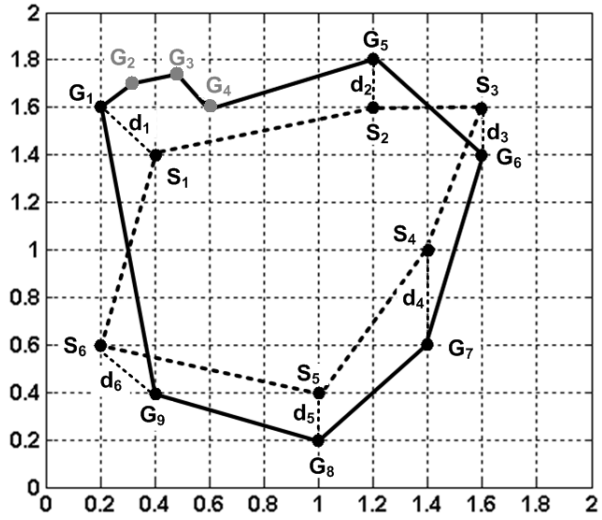

Figure 6.6: *Example of two objects represented by point sets (G_i and S_i), and the point-to-point distances, d_i.*

is drawn to the closest neighbor of the points in set G. The set of distances are calculated as d_i:

$$d_i = \min_{j \in J}(\text{dist}(S_i, G_j)) \tag{6.4}$$

where $\text{dist}(x, y)$ is the standard definition of Euclidean distance:

$$\text{dist}(A, B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2} \tag{6.5}$$

for two points, A and B, located in three-dimensional space. The RMS distance is calculated as:

$$RMS = \sqrt{\frac{1}{I}\sum_{i=1}^{I} d_i^2}[\text{units}] \tag{6.6}$$

where I is the number of points in set S.

6.4 Phantom Alignment using ICP

As the exact locations of the phantom ground truths with respect to the ultrasound volumes are not known, an alignment was performed before calculating the surface error metric. This alignment was done with 6 degrees of freedom in three dimensions - translation along the x, y, and z axes as well as rotation about the x, y, and z axes. This was done using the Iterative Closest Point (ICP) algorithm [63].

The ICP method is used to estimate the six unknown parameters of the 3D rigid transformation that optimally aligns the two surfaces by minimizing the mean of the squared distances between corresponding pairs of points. One point set is considered the moving point set and is the point set that the transformation is applied to; the other the fixed set. Like ths surface error metric, every point in the moving set is compared to the fixed set - as a result, the segmentation result is set as the moving set so that every point is guaranteed to contribute to the cost function (accuracy metric). The ground truth set is used as the fixed set. Once the moving-to-fixed inter-point squared distances have been minimized by way of a locally optimal 3D Euler transformation, the segmentation result is passed to the surface error metric calculation.

The transformation is defined by three translational and three rotational parameters $\{T_x, T_y, T_z, \theta_x, \theta_y, \theta_z\}$. The ICP is an iterative process and must be supplied an initial set of transform parameters. The unknown parameters can be initialized with the identity transform; however, the method will by definition converge to the nearest local minima and consequently should be initialized with a best guess. In practice this was achieved by first aligning the ground truth model with the segmented result models using a best guess (visual) before converting to point sets, and initializing the ICP algorithm with the identity transform.

Once the transform has been initialized, a distance metric quantifies the shape and spatial similarity between the two surfaces. The metric works by

traversing the point-set of the segmentation result. The distance from the current "result" point to the closest point of the model point-set is recorded and the sum of the all of these squared distances are used as the metric after the segmentation point-set has been fully traversed. This is exactly the same process as the surface error metric discussed in the previous section. By using this formulation, the sets are not constrained to have the same number of points, which can be difficult (nearly impossible) to achieve when sampling two boundaries that have different shapes and surface areas. The reason for traversing the result set, and not the model set, were discussed in the previous section as well.

The role of the optimizer is to update the transform parameters at each iteration in such a way that the distance metric is reduced in a least squares sense. The process is repeated until the metric reaches the local minimum within the parameter space (convergence) and this optimal transform is applied to the moving point set before the segmentation accuracy is calculated. It should be noted that we cannot ensure that the local miminum is indeed the global minimum - which is why the sets are carefully aligned with a visual best guess before the fine tuning the alignment using ICP. The Levenberg-Marquardt method was used for the optimization component.

The Levenberg-Marquardt method [35] is an iterative process that minimizes a non-linear function over a given parameter space. In this case, we are solving for the optimal transform parameters that minimize the inter-surface distance metric in a least squares sense. The distance is minimized by iteratively updating the transform parameters in a direction that reduces the metric (and hence increases similarity) from iteration to iteration.

Consider the two example point sets shown in Figure 6.7. Represented by the 929 dots in black is a ground truth point set representing a (rough) sphere of radius 3 (the points were extracted from a 3D image of a sphere created with a coarse resolution to reduce the number of resultant points, and allow them to be visible for this illustration). The ground truth points

were replicated and translated by $\{-3, -3, -3\}$ to create the moving set, shown as red dots. The two sets, prior to alignment, have a surface error of 3.14767 RMS units. The expected transformation to move the red set to the black set would be $\{3, 3, 3, 0, 0, 0\}$. Plugging these example point sets into the ICP algorithm, the resulting transformation is returned as:

Table 6.2: *ICP transformation result.*

$T_x = 2.99767$ [units]	$T_y = 2.99785$ [units]	$T_z = 2.99703$ [units]
$\theta_x = 0.0867102$ [rad]	$\theta_y = 0.0798582$ [rad]	$\theta_x = 0.079938$ [rad]
RMS: $3.14767 \rightarrow 0.142431$ [units]		

which agrees very closely with the expected transformation. The final RMS is 0.142431 units after the alignment. Had the moving set not been allowed to rotate as it was aligned, we might have gotten a nearly exact alignment (for this contrived example using duplicated point sets), but we need to allow for rotation of the phantom images and in this case the points didn't end up matching perfectly. Note that this translation was the starting point before ICP as used - different results might have been obtained with a different starting translation. When this work was performed with the segmentation data, the visual best guess initialization was much more accurate than the starting point used in this example, ensuring that the local minimum found by the alignment was as close to the global minimum as possible.

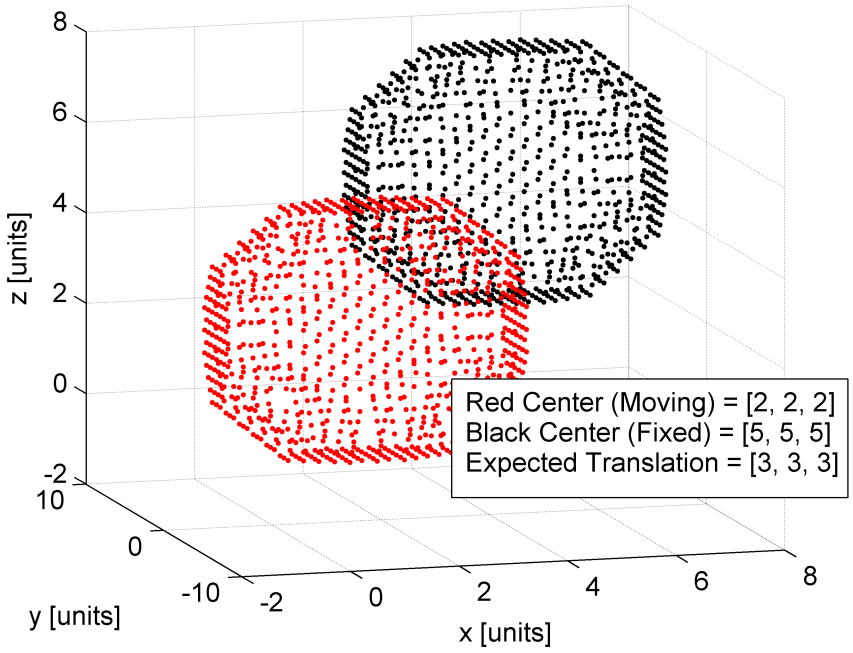

Figure 6.7: *Iterative Closest Point algorithm example. Red points indicate the moving point set (segmentation result), and black points represent the fixed (ground truth) set. The two sets are identical but with a translation applied. The goal of the ICP is to align the sets (optimally, finding the inverse of the transformation that was originally applied).*

Chapter 7

Implementation Tools

7.1 Development Tools

The Insight Toolkit (ITK) [20] was used to implement the various preprocessing, segmentation, and accuracy calculations [61],[21]. ITK is a free open-source, cross-platform C++ toolkit for image segmentation and registration, and is supported by the National Library of Medicine [54]. ITK has not been, and under current guidelines cannot be, approved by the Food and Drug Administration, as software cannot be approved as a medical device by the FDA. The toolkit is deemed an "off the shelf product", meaning the developer handles the responsibility of verifying functionality [12]. Once this software is put into hardware (in our case, the mobile ultrasound scanner), that can be approved as a medical device. The guidelines for software validation can be found in [13], and ITK implements many of these guidelines, including continuous testing via dashboard (CTest-Dart), version control (CVS), configuration standardization (CMake), and bug tracking (phpBugTracker) [14].

ITK versions 3.0.0, 3.2.0 and 3.4.0 were used for the development of this research, and the final routines all re-compiled with ITK 3.6.0. Versions 3.4.0 and earlier versions were copywritten under a modified version of the tradi-

tional BSD (Berkeley Software Distribution) license for open-source software [1]. ITK versions 3.6.0 and above are distributed with a new license that was approved by the Open Source Initiative [37]. Essentially, the license states that redistribution is allowed in both source or binary forms, for commercial or other purposes, without any restrictions, given that the BSD license is distributed with the software, and neither the Insight Software Consortium nor the contributors are to be used to promote or endorse the final products derived from the software [23]. The only change from versions 3.4.0 and earlier to 3.6.0 is the removal of one condition: "Modified source versions must be plainly marked as such, and must not be misrepresented as being the original software". There are a few patented algorithms contained within the ITK toolkit that are exempt from the open-source license, and are plainly marked as such. In fact, the patent on the ICP algorithm used in this research for phantom alignment (see Section 6.4) recently expired (USPTO # 5,715,166 expired on Feb. 3, 2006) [2].

As the ITK toolkit is cross-platform, CMake (Cross Platform Make) 2.4.8 was used to generate the platform-appropriate project configuration files [6]. This used ITK-supplied CMake configuration files to create the appropriate project files, such as those for a Unix environment, Borland, Visual Studio, or MinGW. Initially, Visual Studio .NET 2003 was used, but numerous problems were encountered trying to build ITK using Visual Studio (configuration errors, compiler errors). As a result, the move was made to Cygwin [8], a Linux-like environment for windows. CMake was used to generate Unix-style makefiles (which dictate the compling, assembling, and linker stages of building the executable), and ITK was built in Cygwin using the standard GNU compiler, GCC [17]. The GNU compiler was used to create 32-bit windows executables (release compilation mode). However, as will be detailed in Section 7.3, there were some memory issues with WinXP 32-bit SP2 and ITK, so Linux was used for processing large image volumes. Using CMake and the GNU compiler, no changes were needed in either the source code or

CMake configuration files to compile Unix versions of the programs (which is indeed the point of CMake). Eclipse 3.3.0, for 32-bit WinXP was used as the development environment to write code and for debugging purposes [11]. Ubuntu "Feisty Fawn" 7.04 was used for the Linux operating system [52]. In addition to ITK, Matlab 7.0.4 [33] was used for a small number of filtering operations - the IBS calculation and the mean curvature speckle reduction (these had been developed before the move was made to ITK).

7.2 Visualization Software

ITK only provides image processing functionality - not image visualization. KitWare, which develops ITK, also develops The Visualization Toolkit (VTK) [55]. VTK was not actually used in any of this work; however, it is mentioned since it would be the natural choice for developing any custom GUIs for future work. For the purpose of viewing the 3D image volumes, VolSuite 3.3t was used [56]. VolSuite is a basic visualization/processing tool built on FLTK [16] and OpenGL [38] that was useful for monitoring processing routines and creating snapshots for this document, but unfortunately is no longer being actively developed or supported. While not used for this work, future work would be advised to take a look at ParaView (a powerful visualization application developed by Kitware), in lieu of VolSuite's demise [39].

7.3 Computer Memory Issues

Due to the enormous amount of memory required for some segmentations, the WinXP 32-bit programs consistently crashed due to a failure to allocate enough memory. As a result, the large segmentations (simulated volumes and the real volumes) were done in 64-bit Linux, with 8 gigabytes (GB) of RAM (random access memory). To understand why so much memory is

used, consider the amount of memory required for the segmentation of a pre-processed image volume that is 100 megabytes (MB) (8-bit). All of the level set calculations need to be done in 32-bit floating point to guarantee valid results. This 100 MB 8-bit image is four times as large in floating point, or 400 MB. The following items shown in Table 7.1 are calculated as part of the segmentation procedure and are kept in memory:

Table 7.1: *Memory used for segmentation, using a 100 MB 8-bit example image.*

	Object	Data Type	Memory
Inputs	Input Image	8-bit	100 MB
	Initialization Volume	8-bit	100 MB
Internal	$\|\nabla(in)\|$ (For Speed Image)	32-bit	400 MB
	Speed Image	32-bit	400 MB
	Φ_i (Level Set Field)	32-bit	400 MB
	Φ_{i-1} (Convergence check)	32-bit	400 MB
	$\frac{\partial \Phi_i}{\partial x}, \frac{\partial \Phi_i}{\partial y}, \frac{\partial \Phi_i}{\partial z}$ (For evolution)	3x32-bit	1200 MB
	$\frac{\partial^2 \Phi_i}{\partial x^2}, \frac{\partial^2 \Phi_i}{\partial y^2}, \frac{\partial^2 \Phi_i}{\partial z^2}$ (For curvature)	3x32-bit	1200 MB
	$\kappa(\Phi_i)$ (Curvature force)	32-bit	400 MB
	$\vec{A}(\Phi_i)$ (Advection force)	3x32-bit	1200 MB
Outputs	Output	8-bit	100 MB
	Masked Output	8-bit	100 MB
		Total:	**6 GB**

For a 100 MB image, 6 GB are used during the segmentation just for image and calculation data. This doesn't include libraries and code for image readers/writers, type casters, or the filtering objects performing the required tasks. For the largest image, the 121 MB real prostate volume, the segmentation required just under 8 GB of RAM. Once the segmentations were performed in Linux, the segmentation ran without crashing and no problems were encountered. Results of smaller volumes in Linux were identical to segmentations performed in Windows, as would be expected from ITK's continuous testing across many platforms using CTest-Dart.

Chapter 8

Image Segmenation Results

The results of the segmentation work are presented in this Chapter. We have used three classes of test images to evaluate the performance of segmentations, given different pre- and post-image formation processing techniques and using a level set based segmentation technique. As image source material, we have used three simulated images of cysts, six tissue-mimicking phantom images, and three clinical scans of prostates as our test image set (see Chapter 3). To these images, we have applied the integrated backscatter calculation (IBS), and to both the IBS and non-IBS processed images, we have performed four different speckle reduction filtering schemes. These speckle reduction methods include median filtering, anisotropic diffusion, mean curvature evolution and curvature flow filtering (see chapter 4).

The processed images were then segmented using manually placed seed points and the level set segmentation method (see Chapter 5). The accuracy of these segmentation results were then compared to ground truth models using a volume error metric and a surface error metric (see Chapter 6). Using these results, the goal is to determine which preprocessing methods improved segmentation accuracy. In order to compare these speckle reduction methods on level ground using the same segmentation method, several parameters in the level set evolution equation were held constant, but these can alter-

natively be fine tuned to achieve the best results (see Section 5.2.4) for a given category of images. Results will be presented with these parameters held constant for all image types in order to evalute the effectiveness of the preprocessing schemes. Presented later will be a few segmentation examples where these parameters have been fine tuned for a few specific images, using the best performing preprocessing routine.

Presented first, however, are the results from the beginning of the research. A few processing methods were investigated using a 2D segmentation method that was later abandoned. The histogram based preprocessing methods were found to reduce image quality and were not effective at reducing speckle type noise. The results presented in Section 8.1 were evaluated using the Nearest Neighbor segmentation algorithm (see Section 4.3), and using the accuracy metric given by (4.5), as these were investigated before the switch was made to a level set-based active contours segmentation routine, and before newer performance metrics were created. Subsequent sections give segmentation results using IBS and the various speckle reduction schemes with respect to the level set method and the volume/surface error metrics.

8.1 Segmenation with Histogram Modification Pre-processing

In Section 4.5, various approaches to histogram modification were presented. The basic concept is that by redistributing the image intensities, we can improve the image quality and make it easier to determine the object boundaries. It was found that most of the histogram techniques did not improve boundary detection accuracy without necessary changes to the Nearest Neighbor algorithm (see Section 4.3). The Nearest Neighbor method relies on the fact that the target objects have very low intensity values as compared to the rest of the image. However, the histogram equalization techniques shift these low values to higher gray level intensities. To compensate for this, the

thresholding parameter in the Nearest Neighbor algorithm must be increased to accurately discern the cysts from the surrounding tissue. Typically, this threshold would be computed as a value around 4 (of 255 levels). Most of the histogram operations move the cyst intensities to the 40-60 range (and above), so the threshold was multiplied by 16.

The Nearest Neighbor method (as it was implemented with the original thresholding value) did not respond well to Gaussian-type noise. Many times a single known object was returned as a "swiss cheese" boundary with many artifacts/holes in the middle. To combat this, the disk (structuring element) size used in the morphological opening/closing operation was increased from a size of 3 pixels to 9 pixels. This increased the accuracy noticeably, and to keep comparisons legitimate, the disk size of 9 pixels was used for all boundary detection cases.

The standard method (see Section 4.5.1) for histogram equalization performed the most reliably of the histogram modification methods. Most test cases had boundary detection accuracies in the 70-80% range (where 100 % represents perfect object segmentation), except for the non-IBS (integrated backscatter) images, which had comparable accuracies when no noise was present. The addition of noise (Gaussian, speckle, and salt-and-pepper) greatly decreased the BD accuracy, with some values in the 10-20% range.

As will be described in the next section, none of the desired histograms produced good results with the Matlab histeq() function (see Section 4.5.2). Almost all accuracies were in the 20-40% range, with a few better performing outliers in the 70-80% range. Many of the cases did not find any objects in the image, for an accuracy of 0%. The outliers were mostly for the cases of Gaussian-type noise, and this may be explained by the fact that typically the Gaussian noise spreads out the detected boundaries. Very small detected objects may be smeared out into a larger region, which may coincide with the actual cyst locations and inflate the accuracy measurement.

The performance of the perfectly flat histogram (see Section 4.5.3) equal-

ization technique fell in between that of the standard equalization and the Matlab histogram specifications. However, Gaussian-type noise greatly reduced boundary detection accuracy, some cases as low as 5-10% (and a few for which no objects were found, 0%). Without any added noise, performance was comparable to the cases for no histogram processing, but quickly degrades with the addition of noise.

8.1.1 Histogram Modification Performance

Shown in Figure 8.1 are 2D segmentation results using the Nearest Neighbor algorithm after histogram modifications have been performed. Figure 8.1(a) shows the ground truth image, Figure 8.1(b) shows segmetations after the standard histogram equalization, Figure 8.1(c) after the prefectly flat histogram equalization, and Figure 8.1(d) after histogram specification using the bandstop function.

A total of 560 boundary detection runs were computed using the Nearest Neighbor method (8 images, 10 noise types, 7 histogram processing techniques). Table 8.1 defines the histogram operations shown in Figure 8.2 (see Figure 4.11 for histeq definitions). The average performance of the results are shown below in Figures 8.2 and 8.3. These accuracy metrics are not the volume or surface error metrics presented previously. The metric is the one used in the Nearest Neighbor metric and was presented in Equation (4.5), which is a value between 0 and 100%, 100% representing perfect segmentation accuracy. Figure 8.2 shows the mean accuracy performance for each of the histogram operations. For each histogram operation, the value shown is the average accuracy performance across all 8 image with the 10 noise types added, meaning each data point is the average of 80 Nearest Neighbor segmentations. Likewise, for Figure 8.3, each data point is the average performance across 8 images with the 7 histogram operations, or a total of 56 images each. The histogram and noise types for each figure are given in Table 8.1.

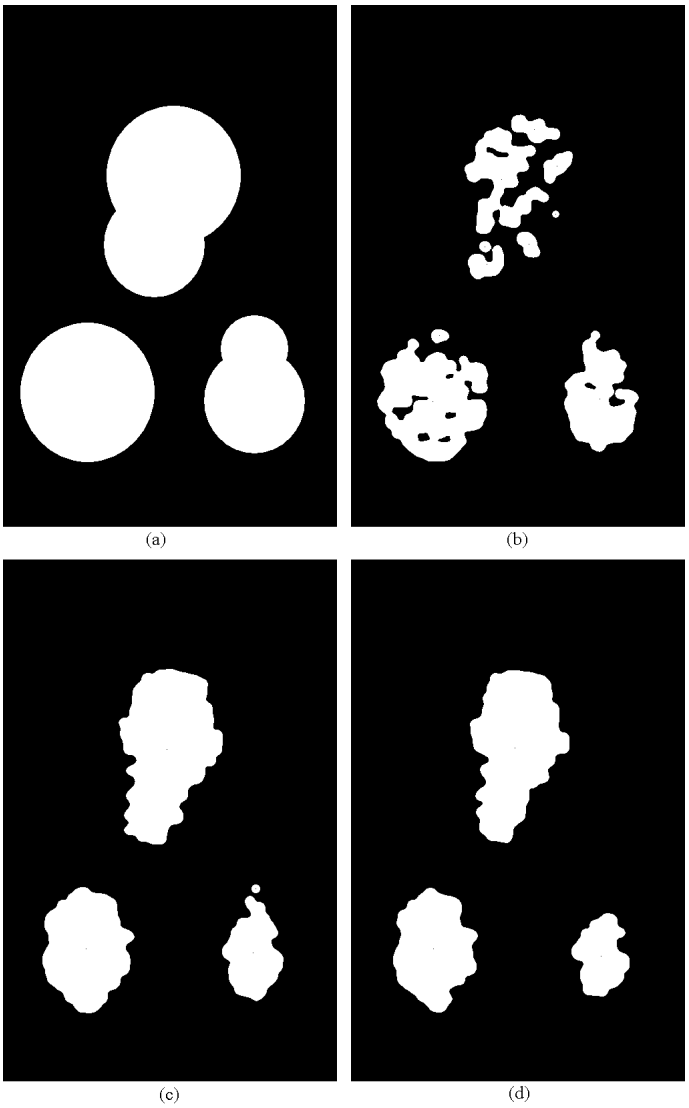

Figure 8.1: *2D segmentation examples using Nearest Neighbor algorithm for different histogram modification methods (a) - Ground truth; (b) - standard histogram equalization; (c) - perfectly flat histogram equalization; (d) - histogram specification using bandstop function.*

Table 8.1: *Histogram modification methods and noise types applied to test images before Nearest Neighbor segmentations, for results presented in Figures 8.2 and 8.3.*

	Histogram Types, Figure 8.2	Noise Types, Figure 8.3
1	Default (Nearest Neighbor)	None
2	Standard CDF	Gaussian, $\sigma^2 = 0.1$
3	Perfectly Flat	Gaussian, $\sigma^2 = 0.4$
4	*histeq()* - rampdown	Gaussian, $\sigma^2 = 0.6$
5	*histeq()* - 2rampsdown	SnP, %pixels affected = 10
6	*histeq()* - bandstop	SnP, %pixels affected = 40
7	*histeq()* - parabolic	SnP, %pixels affected = 60
8		Speckle, $\sigma^2 = 0.5$
9		Speckle, $\sigma^2 = 1.0$
10		Speckle, $\sigma^2 = 1.5$

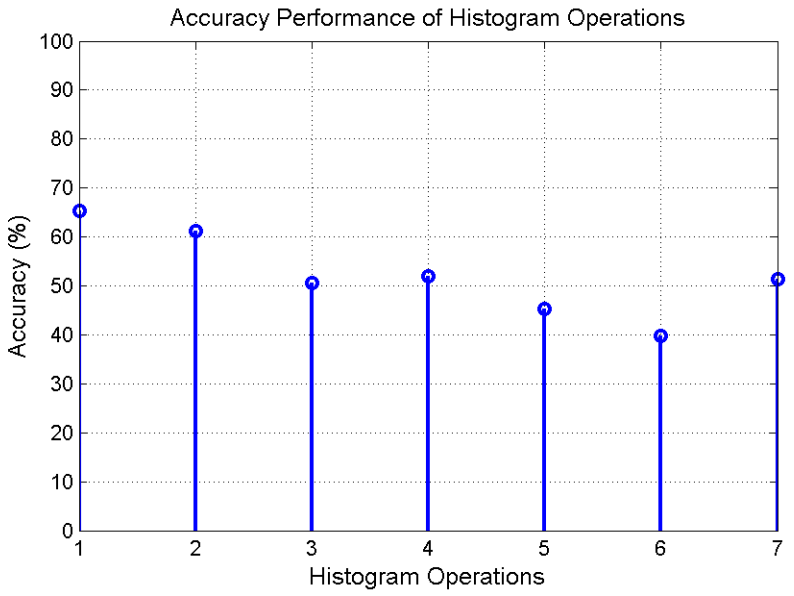

Figure 8.2: *Average Nearest Neighbor performance for various histogram operations (listed in Table 8.1).*

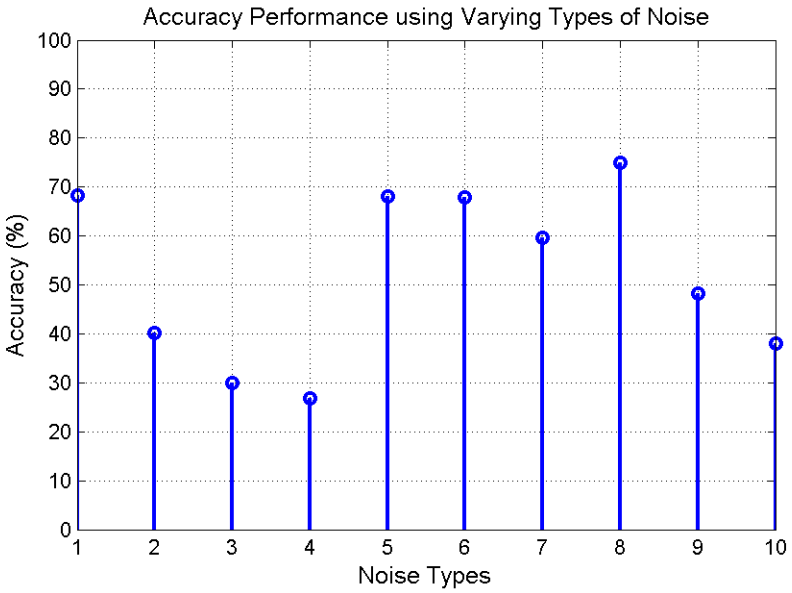

Figure 8.3: *Average Nearest Neighbor performance for various noise types (listed in Table 8.1).*

It should be noted that since the Nearest Neighbor parameters were tweaked to give reasonable performance across many histogram operations, the default (hist op = 1) is much lower than if the parameters were set to their default values. However, from the results shown in Figure 8.2, the histogram operations did not improve the Nearest Neighbor accuracy significantly. This may be attributed to the fact that the Nearest Neighbor algorithm assumes that the objects of interest have very low intensity values compared to the rest of the image. The simulated cysts did have this characteristic, but the histogram operations changed many of these low values to higher ones. As a result, the histogram operations actually distorted the cyst boundaries.

8.2 Segmentation Results with Integrated Backscatter and Speckle Reduction Pre-processing

The speckle reduction types referred to in the tables and figures in Sections 8.2 and 8.3 are as follows:

1. None
2. Median
3. Anisotropic Diffusion
4. Mean Curvature Evolution
5. Curvature Flow

In order to evaluate the speckle reduction performance using the exact same segmentation method across all images, a global set of level set evolution parameters were used ($\alpha = 1.0$, $\beta = 0.3$, $\gamma = 1.5$). These parameters were discussed in Section 5.2.4 and were used for the results presented in Sections 8.2 and 8.3. In Section 8.4, results are presented where the optimal preprocessing routine is used with evolution parameters that have been fine-tuned for each image type to show more accurate segmentations.

8.2.1 Segmentation of Simulated Field-II Images

Volume error results (see Section 6.2) for the three simulated images using various speckle reduction methods with and without the IBS calculation are given in Table 8.2 and shown in Figure 8.4. The bolded and underlined values show the speckle reduction method with the best accuracy for each image. It should be noted that the segmentation results for image 1 are much better than those for image 2 and 3. Image 1 was of a single spherical cyst that resulted in relatively good contrast. Images 2 and 3 consisted of more elaborate shapes, constructed from 3 overlapping spheres and situated at a lower scan depth. As a result, images 2 and 3 had less contrast and the curvature forces prevented some of the segmentations from fully recovering the smaller, more detailed shapes.

Surface error results (see Section 6.3) for the three simulated images using

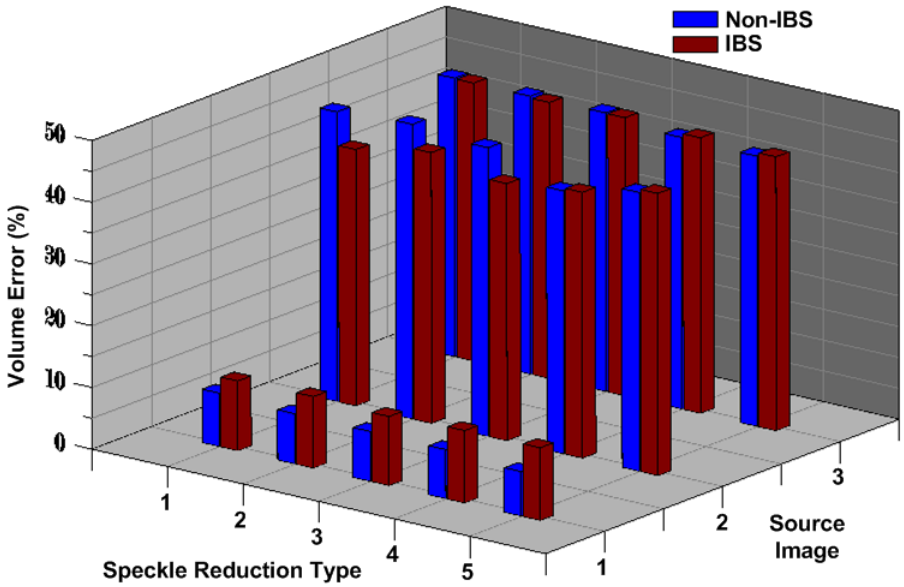

Figure 8.4: *Results, volume error, simulated images (%). Three original source images. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering.*

Table 8.2: *Volume error results, Field II simulated images (%). Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering. The bolded and underlined values show the speckle reduction method with the best accuracy for each image.*

IBS	non-IBS					IBS				
Speck. Red. Meth.	1	2	3	4	**5**	1	2	3	4	5
sim1	8.65	8.14	8.08	7.97	**7.21**	11.31	11.61	11.22	11.75	11.80
sim2	46.94	47.76	46.80	42.61	45.19	**41.55**	43.88	41.70	43.09	45.75
sim3	45.14	45.05	45.17	44.02	**43.80**	45.05	44.75	45.10	44.67	44.40

various speckle reduction methods with and without the IBS calculation are given in Table 8.3 and shown in Figure 8.5.

Table 8.3: *Surface error results, simulated images (RMS, mm). Three original source images. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering. The bolded and underlined values show the speckle reduction method with the best accuracy for each image.*

IBS	non-IBS					IBS				
Speck. Red. Meth.	1	2	3	4	5	1	2	3	4	5
sim1	1.24	1.28	**1.23**	1.26	1.26	1.53	1.54	1.53	1.53	1.55
sim2	1.81	1.89	1.81	1.57	1.69	**1.42**	1.52	1.43	1.49	1.61
sim3	1.89	1.89	1.89	**1.73**	1.78	1.76	1.80	1.77	1.77	1.80

An example of several segmetations are shown in Figure 8.6. Figure 8.6(a) shows one of the original Field-II simulated image volumes, and the ground truth volume in Figure 8.6(b) which has one sphere of radius 12mm. Figures 8.6(c-f) show the level set segmentation results after different speckle reduction methods have been applied.

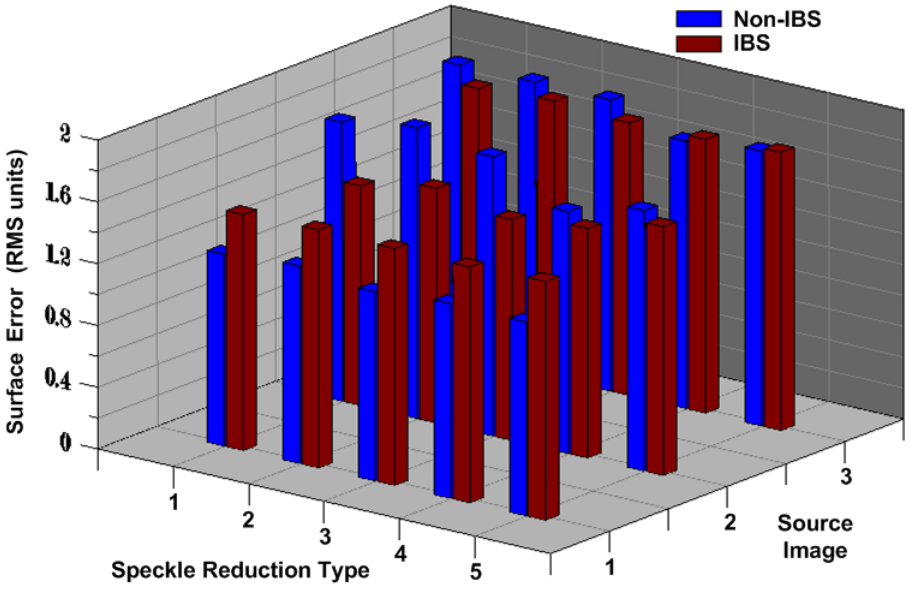

Figure 8.5: *Results, surface error, Field II simulated images (RMS, mm). Three original source images. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering.*

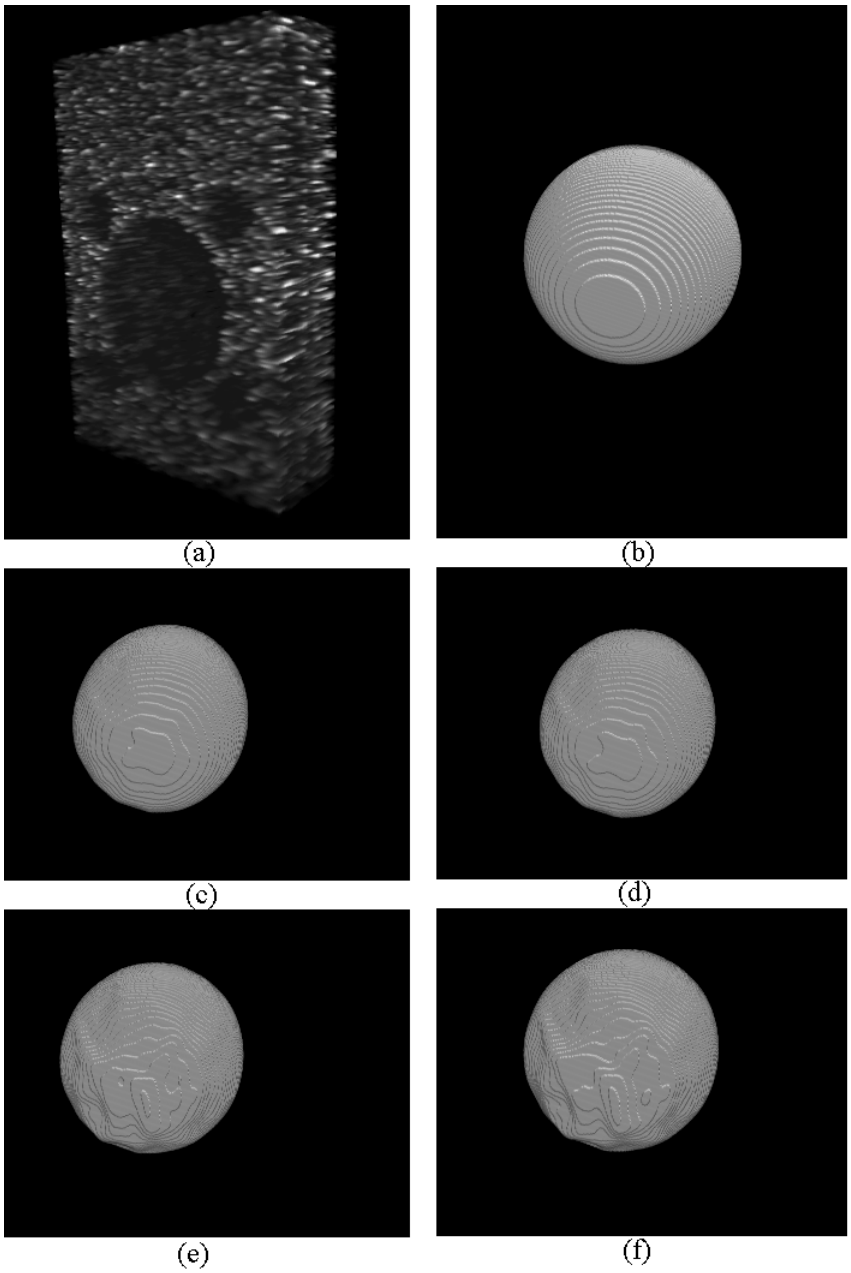

Figure 8.6: *Segmentation examples. (a) Original Field-II simulated image 1; (b) ground truth volume; (c) median, IBS; (d) anisotropic diffusion, IBS; (e) median, no IBS; (f) anisotropic diffusion, no IBS.*

8.2.2 Segmentation of Images from Ultrasound Phantoms

Volume error results for a set of six images from ultrasound phantoms using various speckle reduction methods with and without the IBS calculation are given in Table 8.4 and shown in Figures 8.7 and 8.8. Occassionally, a segmentation would not converge and end up leaking out far beyond the target boundary. In these cases, the volume error would end up being very large because the segmenation volume was much larger than the ground truth volume. These cases are marked with an asterisk in Table 8.4 and have been removed from the graphs in Figures 8.7 and 8.8. With those values left in, the graph axis is wildly inflated and the results of the segmentations that did not leak excessively cannot be discerned from one another clearly.

Table 8.4: *Volume error results, ultrasound phantom images (%). Six original source images. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering. The bolded and underlined values show the speckle reduction method with the best accuracy for each image. The bolded and underlined speckle reduction method indicates the speckle reduction method that had the best accuracy across the most images.*

IBS	non-IBS					IBS				
Speck. Red. Meth.	1	2	**3**	4	5	1	2	3	4	5
box35	18.79	20.02	**14.51**	18.64	18.04	240.7*	244.6*	20.32	226.4*	19.72
box45	54.54	54.53	54.44	**4.49**	54.42	54.59	154.2	54.60	54.60	54.57
box55	28.43	28.09	**26.00**	212.1*	26.40	28.80	296.6*	28.13	276.5*	28.40
cyl35	26.43	26.52	22.84	24.48	21.42	27.11	**8.12**	26.09	24.45	23.28
cyl45	138.4*	123.2*	**21.88**	94.16	62.27	29.79	205.7*	29.11	31.33	28.95
cyl55	223.6*	215.5*	**15.03**	195.3	16.99	210.7	208.4*	198.5*	194.6*	52.89

Surface error results for the six ultrasound phantom images using various speckle reduction methods with and without the IBS calculation are given in Table 8.5 and shown in Figures 8.9 and 8.10.

An example of several segmentations are shown in Figure 8.11. Figure 8.11(a) shows one of the original ultrasound box phantom image vol-

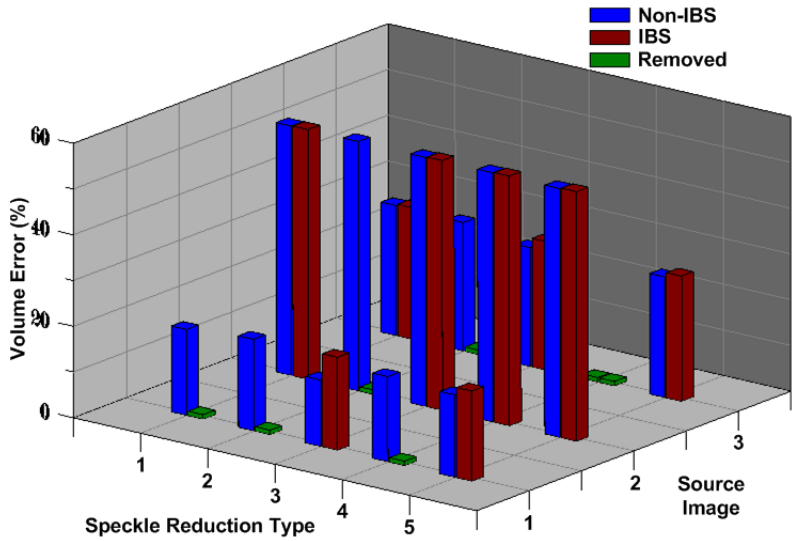

Figure 8.7: *Results, volume error, ultrasound box phantoms (%). Six original source images. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering.*

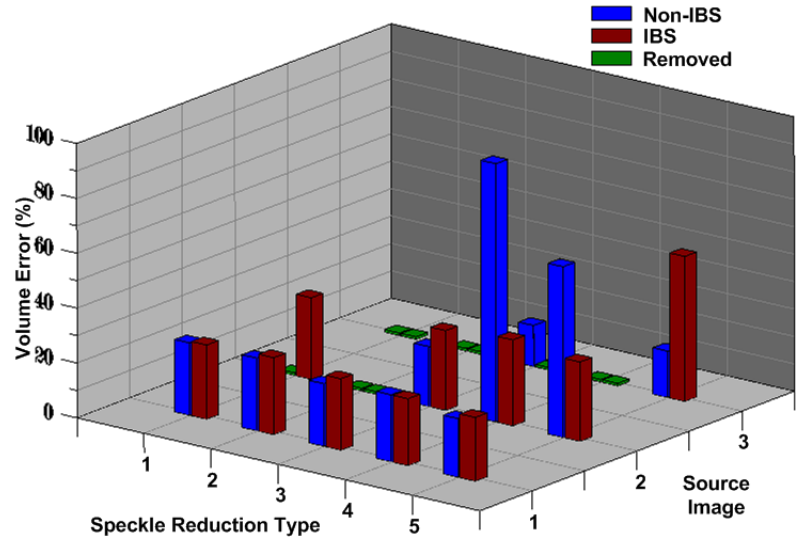

Figure 8.8: *Results, volume error, ultrasound cylinder phantoms (%). Three original source images. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering.*

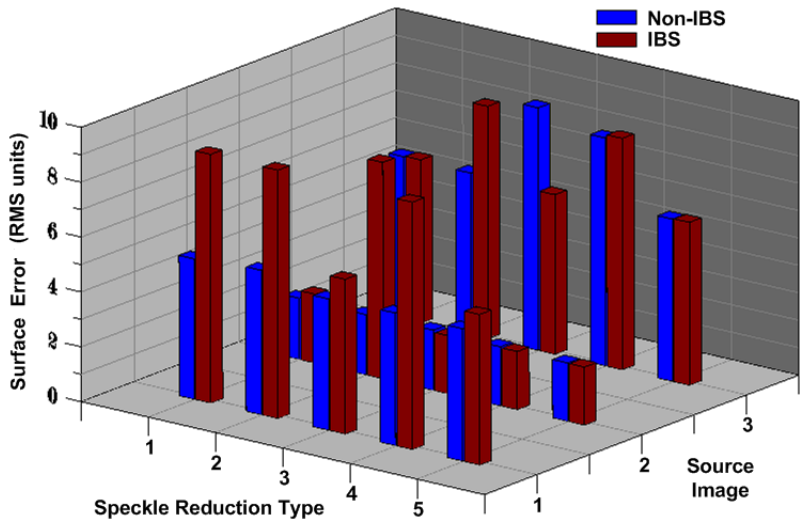

Figure 8.9: *Results, surface error, ultrasound box phantoms (RMS, mm). Three original source images. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering.*

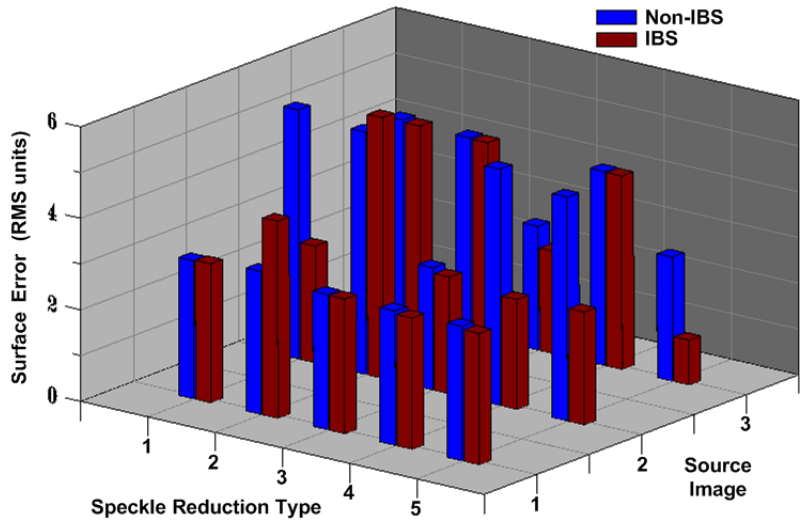

Figure 8.10: *Results, surface error, ultrasound cylinder phantoms (RMS, mm). Three original source images. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering.*

Table 8.5: *Surface error results, ultrasound phantom images (RMS, mm). Six original source images. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering. The bolded and underlined values show the speckle reduction method with the best accuracy for each image. The bolded and underlined speckle reduction methods indicate the methods that had the best accuracy across the most images.*

IBS	non-IBS					IBS				
Speck. Red. Meth.	1	2	3	4	5	1	2	3	**4**	**5**
box35	5.12	5.27	**4.79**	4.84	4.83	9.07	9.06	5.64	9.02	5.49
box45	2.23	2.20	2.20	2.15	2.13	2.51	7.89	**2.12**	**2.12**	**2.12**
box55	5.94	5.90	8.85	8.31	5.92	5.96	8.48	**5.82**	8.44	5.94
cyl35	3.02	3.13	2.96	2.94	2.94	3.04	4.31	2.95	**2.87**	**2.87**
cyl45	5.45	5.30	2.67	5.17	4.90	2.55	5.70	2.56	**2.40**	2.47
cyl55	4.36	4.30	2.71	4.24	2.71	4.30	4.28	2.24	4.23	**0.98**

umes (35% graphite, highest contrast), and the ground truth volume in Figure 8.11(b). Figures 8.11(c-f) show the level set segmentation results after different speckle reduction methods have been applied. The box shape was used as a "tough" case that would probably not be seen in true anatomy because of the sharp corners, but would show how much the curvature force adversely affects the segmentation of object with sharp corners. The curvature force in the segmentation method serves to bridge weak boundaries in the target boundaries, but also prevents the curve from taking on fine, detailed shapes like those found at the corners of the boxes. As a result, even the best case, shown in Figure 8.11(f) does not completely capture the box shape, but is rather smoothed and more cylindrical in shape. Similar results are found in Figure 8.12, which is of a box phantom but with 55% graphite, the lowest contrast investigated in this work. One should also note that these results have not been aligned to the ground truth model yet; the phantom alignment (see Section 6.4) prior to the surface error metric does not occur until the image volumes shown here have been converted into point sets.

Figures 8.13 and 8.14 show similar results to 8.11 and 8.12, but for the

cylindrical phantoms. Most segmetations in Figure 8.13 are accurate, and for this one case of median filtering with the IBS calculation, the segmentation was very good. However, across the most of the other test cases, this preprocessing scheme did not hold up. The segmentations for the cylinder phantoms with 55% graphite, the weakest contrast images, did very poorly except for the one preprocessing case of anisotropic diffusion applied to the non-IBS processed image, shown in Figure 8.14(f).

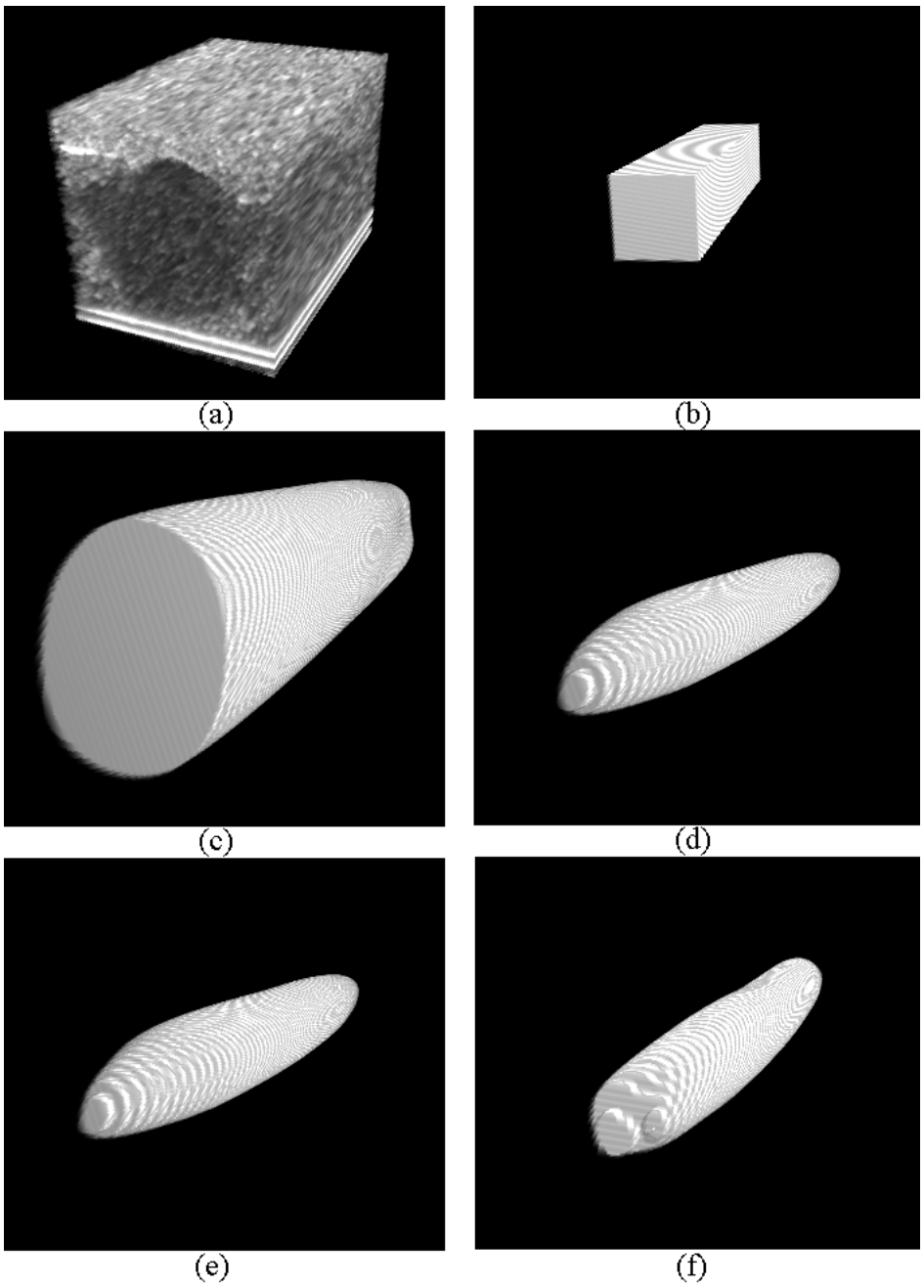

Figure 8.11: *Segmentation examples. (a) Original ultrasound box phantom image, 35% graphite; (b) ground truth volume; (c) median, IBS; (d) anisotropic diffusion, IBS; (e) median, no IBS; (f) anisotropic diffusion, no IBS.*

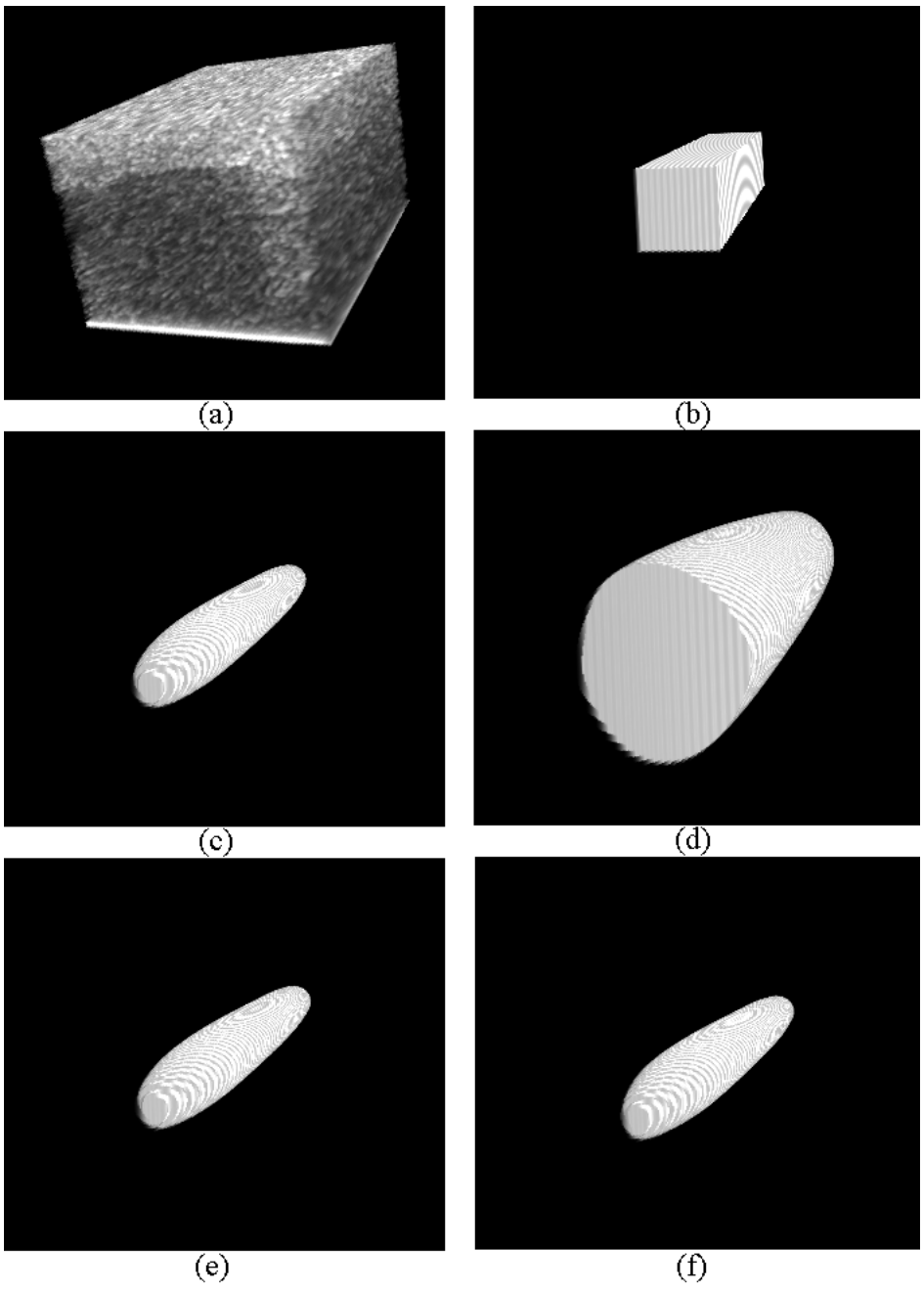

Figure 8.12: *Segmentation examples. (a) Original ultrasound box phantom image, 55% graphite; (b) ground truth volume; (c) median, IBS; (d) anisotropic diffusion, IBS; (e) median, no IBS; (f) anisotropic diffusion, no IBS.*

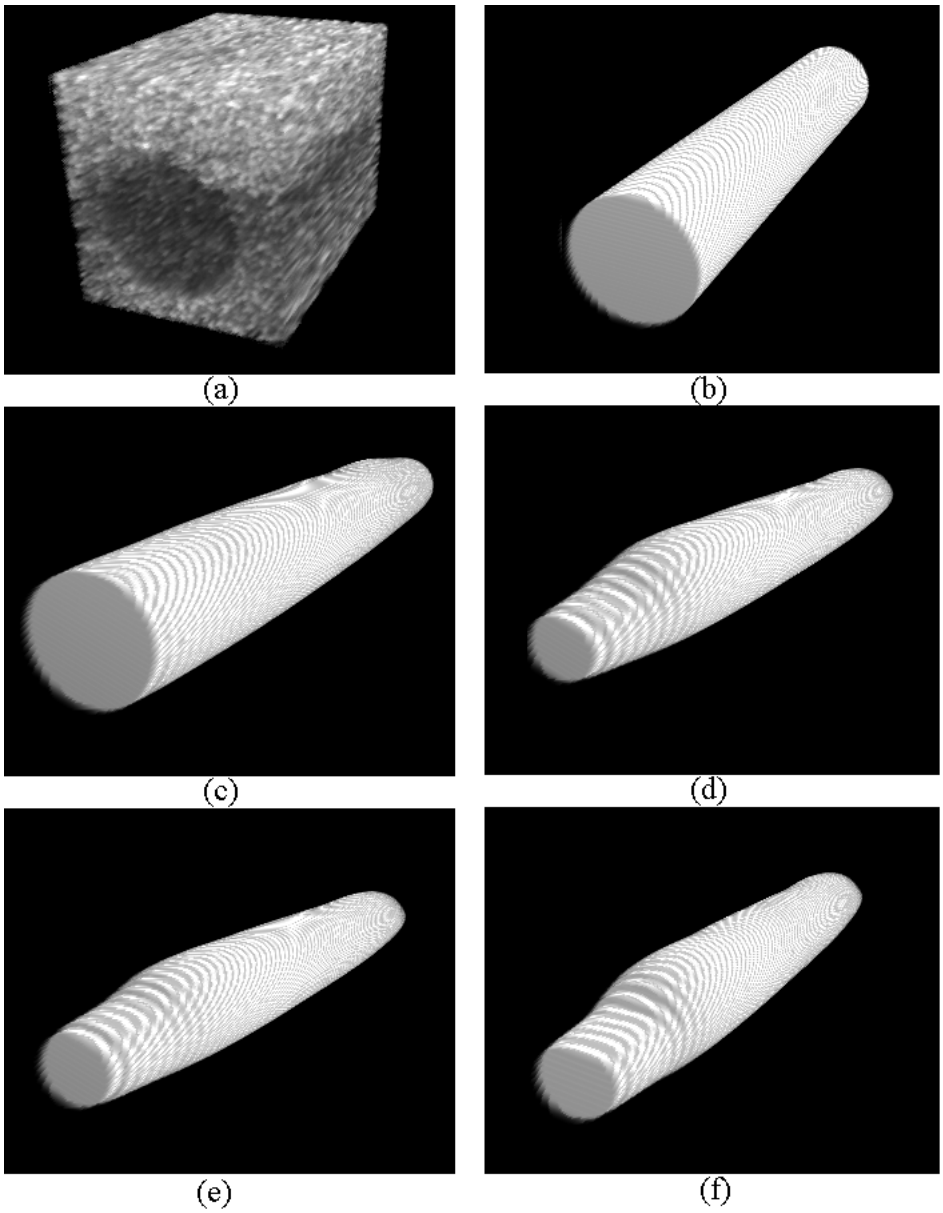

Figure 8.13: *Segmentation examples. (a) Original ultrasound cylinder phantom image, 35% graphite; (b) ground truth volume; (c) median, IBS; (d) anisotropic diffusion, IBS; (e) median, no IBS; (f) anisotropic diffusion, no IBS.*

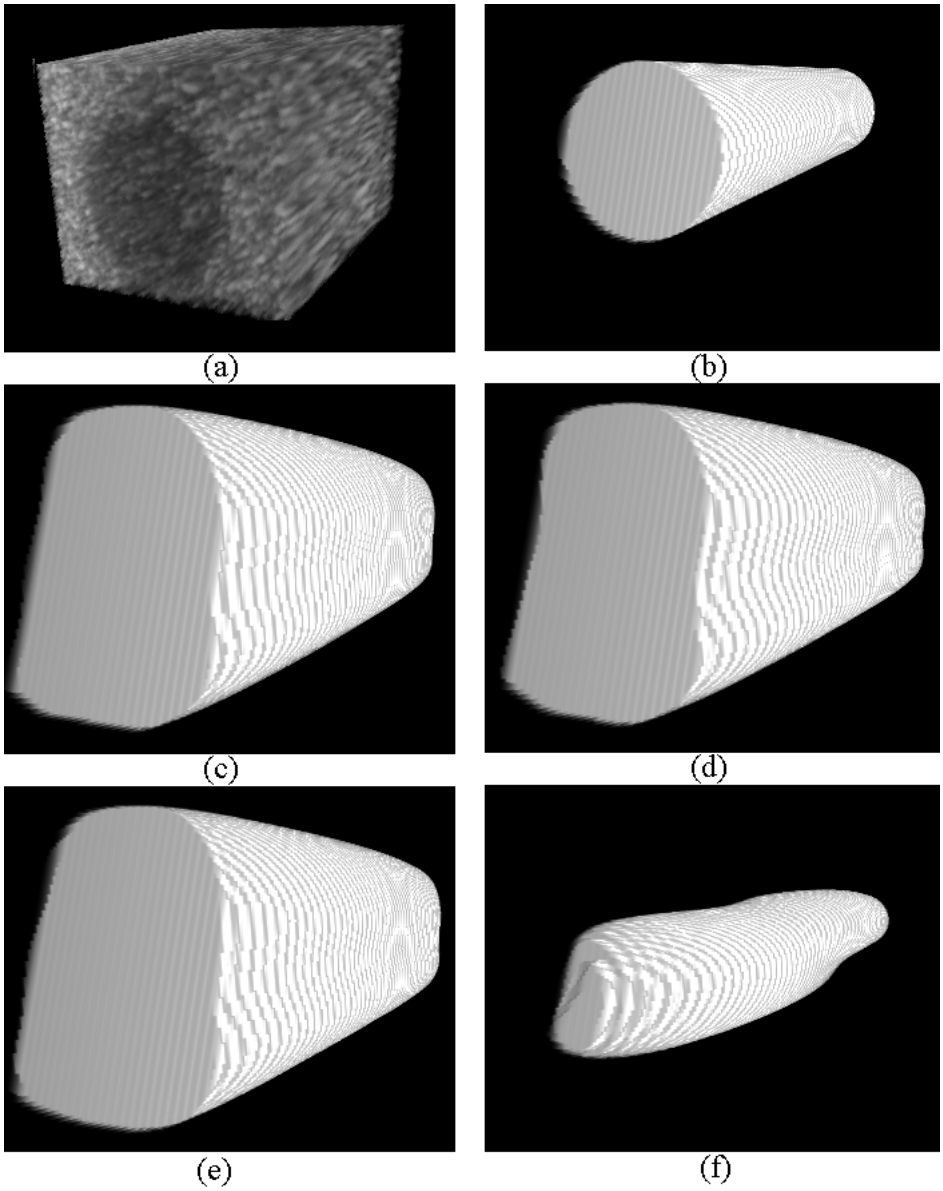

Figure 8.14: *Segmentation examples. (a) Original ultrasound cylinder phantom image, 55% graphite; (b) ground truth volume; (c) median, IBS; (d) anisotropic diffusion, IBS; (e) median, no IBS; (f) anisotropic diffusion, no IBS.*

8.2.3 Segmenation of Clinical Images

As the original RF data was not available for the three clinical prostate images (only the image intensity information), the IBS calculation could not be completed and hence no IBS results are presented for this data. Shown in Figure 8.15 are one original image of a prostate scan, with the two original hand outlined volumes from two doctors shown in Figures 8.15(b) and (c). Figures 8.15(d) and (e) show the smoothed version of the ground truth volumes, which were used to calculate the performance metrics. Shown in Table 8.6 are the volume error metrics for the clinical segmentations. The results are presented as compared to the two different ground truth models, obtained from two different doctors. The smoothed ground truth models were used for these performance metrics (see Section 6.1.1). Figure 8.16 shows this data for the three images, five speckle reduction methods and the two ground truth volumes. Also, as the two hand-outlined ground truth volumes do not perfectly agree (even after smoothing), the doctor-to-doctor volume errors can give an idea of an error margin and were calculated as: *real1*, doc vs. doc: 10.68%; *real2*: 0.384%; *real3*: 0.046% (Prior to smoothing: 9.28%, 3.16%, 3.32%). Note that the disrepancies for *real2* and *real3* were larger prior to smoothing.

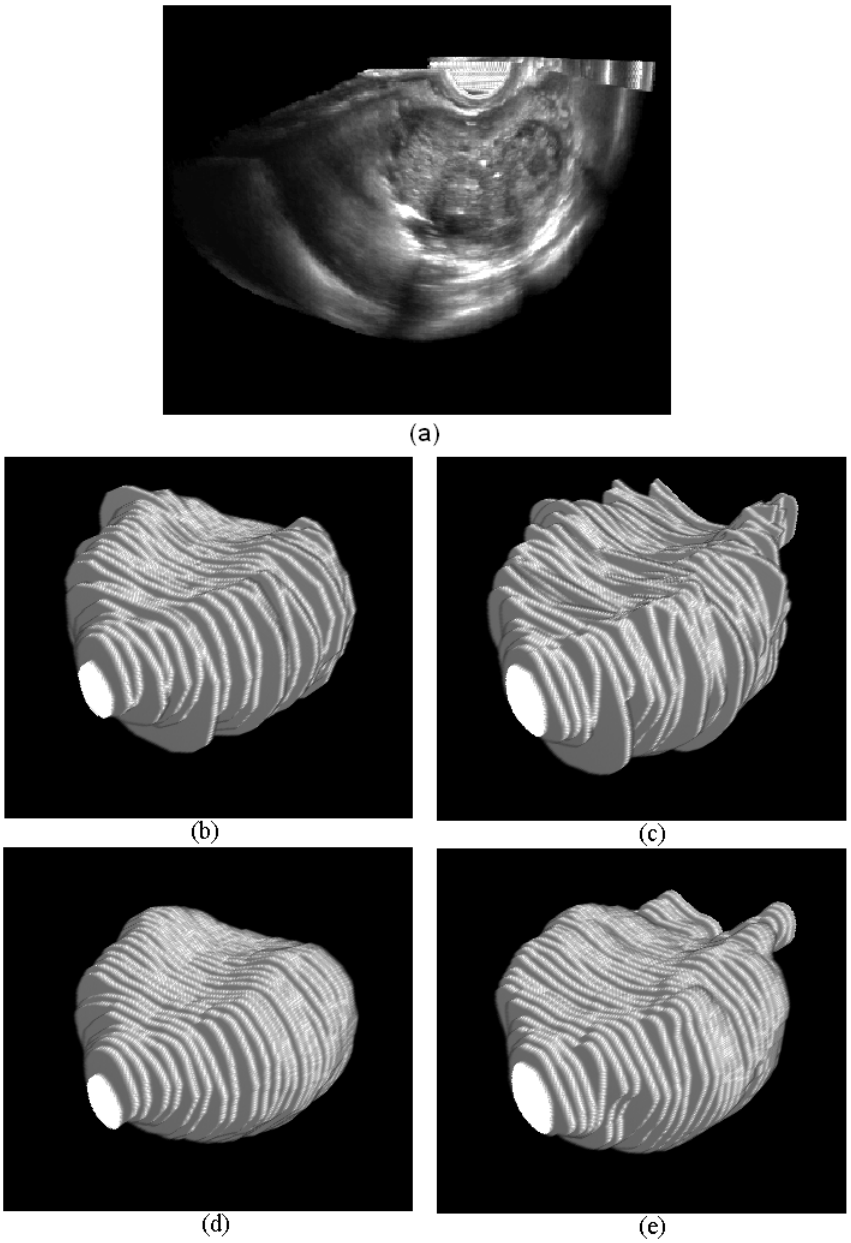

Figure 8.15: *Ground truth examples. (a) Original prostate scan, image 3; (b) ground truth volume, doctor 1; (c) ground truth volume, doctor 2; (d) smoothed ground truth volume, doctor 1; (e) smoothed ground truth volume, doctor 2.*

Table 8.6: *Volume error results, clinical images (%). Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering. The bolded and underlined values show the speckle reduction method with the best accuracy for each image. The bolded and underlined speckle reduction method indicate the method that had the best accuracy across the most images.*

Speck. Red. Meth.	1	2	**3**	4	5
real1_g1	15.52	15.98	**15.05**	15.38	33.51
real1_g2	29.33	29.84	**28.81**	29.18	49.48
real2_g1	86.14	19.92	**18.76**	19.08	84.18
real2_g2	85.42	19.46	**18.30**	18.63	83.47
real3_g1	32.14	32.66	32.00	32.96	**22.93**
real3_g2	32.20	32.73	32.06	33.03	**22.99**

Table 8.7: *Surface error results, clinical images (RMS, mm). Three original source images. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering. The bolded and underlined values show the speckle reduction method with the best accuracy for each image. The bolded and underlined speckle reduction method indicate the method that had the best accuracy across the most images.*

Speck. Red. Meth.	1	2	**3**	4	5
real1_g1	2.89	2.91	**2.87**	2.88	3.35
real1_g2	2.97	3.00	**2.95**	2.96	3.78
real2_g1	6.40	3.17	**3.09**	3.11	6.34
real2_g2	6.62	3.32	**3.25**	3.27	6.55
real3_g1	3.31	3.33	3.30	3.35	**2.77**
real3_g2	3.77	3.79	3.77	3.80	**3.37**

Table 8.7 gives the surface error metrics for the three clinical images with various speckle reduction methods, as compared to the two smoothed hand-outlined ground truth volumes for each image. Figure 8.17 displays this data. As with the volume error metric results discussed above, the two

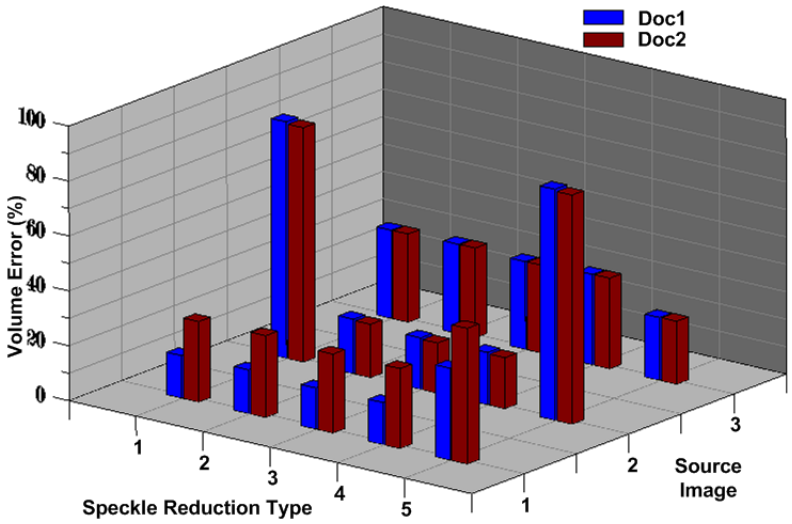

Figure 8.16: *Results, volume error, real images (%). Three original source images. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering.*

hand-outlined ground truth volumes do not perfectly agree. The doctor-to-doctor surface accuracies were calculated as: *real1*, doc vs. doc: 1.93 mm; *real2*: 2.11 mm; *real3*: 2.81 mm (Prior to smoothing: 1.67 mm, 2.88 mm, 2.78 mm).

Shown in Figures 8.18 - 8.20 are segmentation examples for each of the three clinical images, using the different speckle reduction methods. Figure 8.18(a) shows clinical image 1, Figure 8.18(b) shows one of the original hand-outlined ground truth volumes (doc1), and Figure 8.18(c) the original doc2 ground truth volume. Note that the ground truth volumes and segmentation results are zoomed in, and not shown at the same scale as the original images shown in Figure 8.18(a). Figure 8.18(d) shows the segmentation after median filtering. The median filtering resulted in the surface expanding out beyond the true prostate boundary and was not very accurate. Figures 8.18(e) and (f) show the smoothed versions of the original ground truth volumes (shown above them in (b) and (c)). Figure 8.18(g) shows the result after the application of anisotropic diffusion processing, (h) after mean cur-

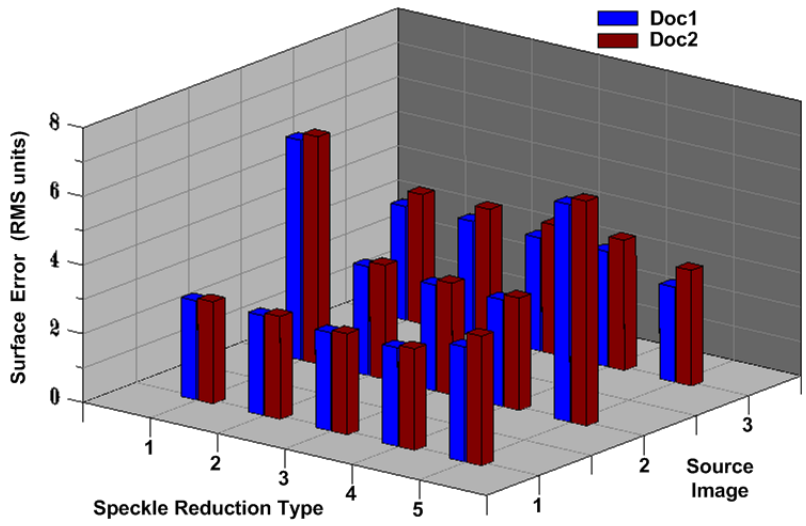

Figure 8.17: *Results, surface error, real images (RMS, mm). Three original source images. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering.*

vature evolution filtering, and (i) after the image volume was processed with curvature flow filtering. Similar results are shown in Figures 8.19 and 8.20 for clinical images 2 and 3.

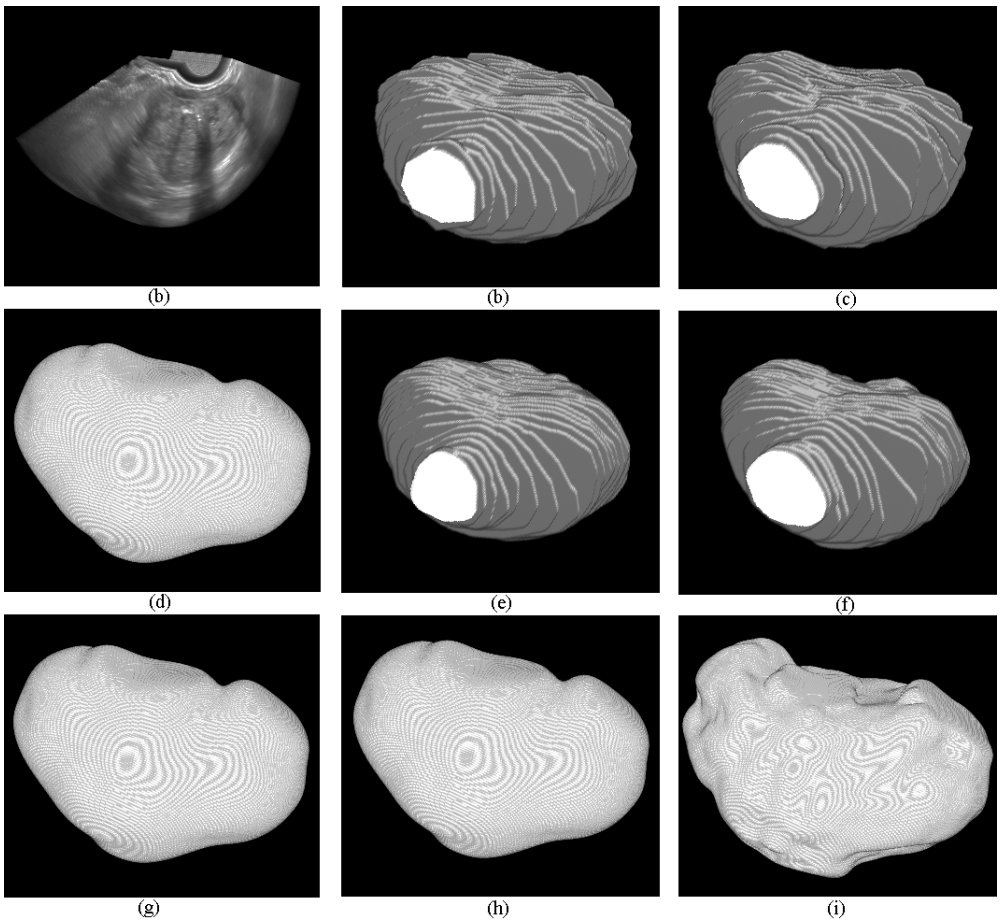

Figure 8.18: *Segmentation examples - clinical prostate scan 1. (a) Original clinical image 1 with cutway; (b) ground truth volume, doctor 1; (c) ground truth volume, doctor 2; (d) median filtering; (e) smoothed ground truth volume, doctor 1; (f) smoothed ground truth volume, doctor 2; (g) anisotropic diffusion filtering; (h) mean curvature evolution filtering; (i) curvature flow filtering.*

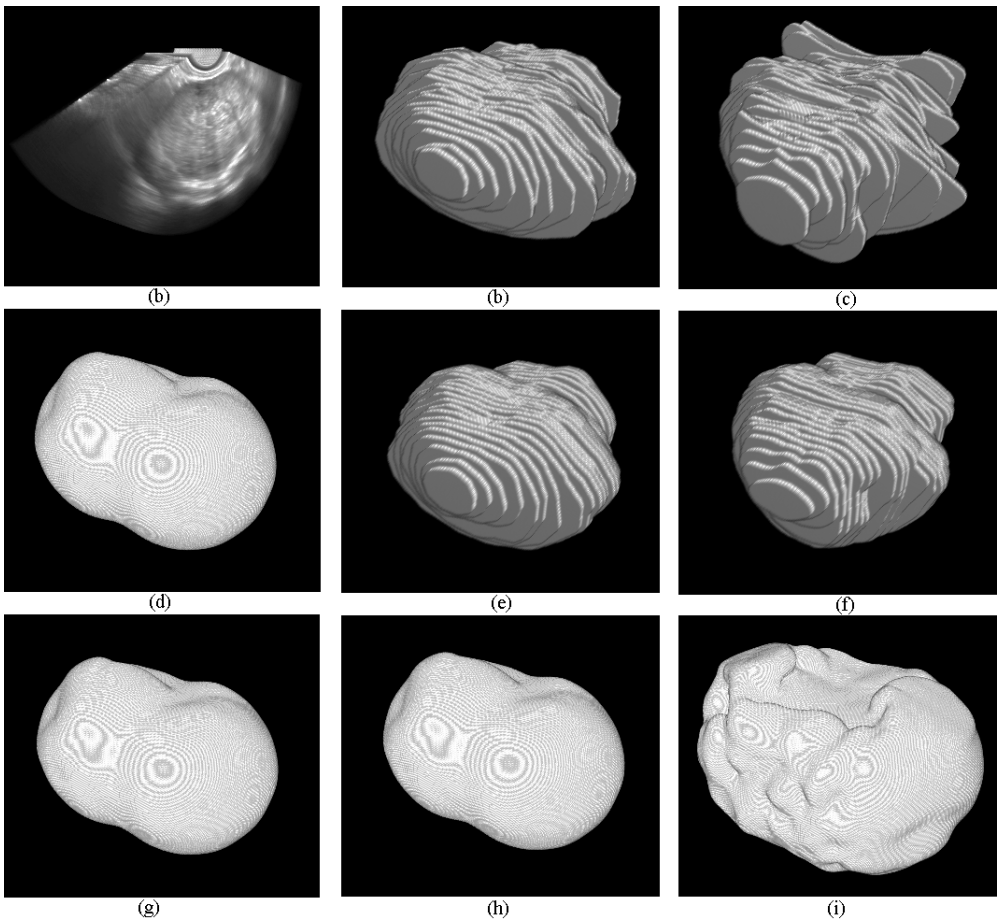

Figure 8.19: *Segmentation examples - clinical prostate scan 2. (a) Original clinical image 1 with cutway; (b) ground truth volume, doctor 1; (c) ground truth volume, doctor 2; (d) median filtering; (e) smoothed ground truth volume, doctor 1; (f) smoothed ground truth volume, doctor 2; (g) anisotropic diffusion filtering; (h) mean curvature evolution filtering; (i) curvature flow filtering.*

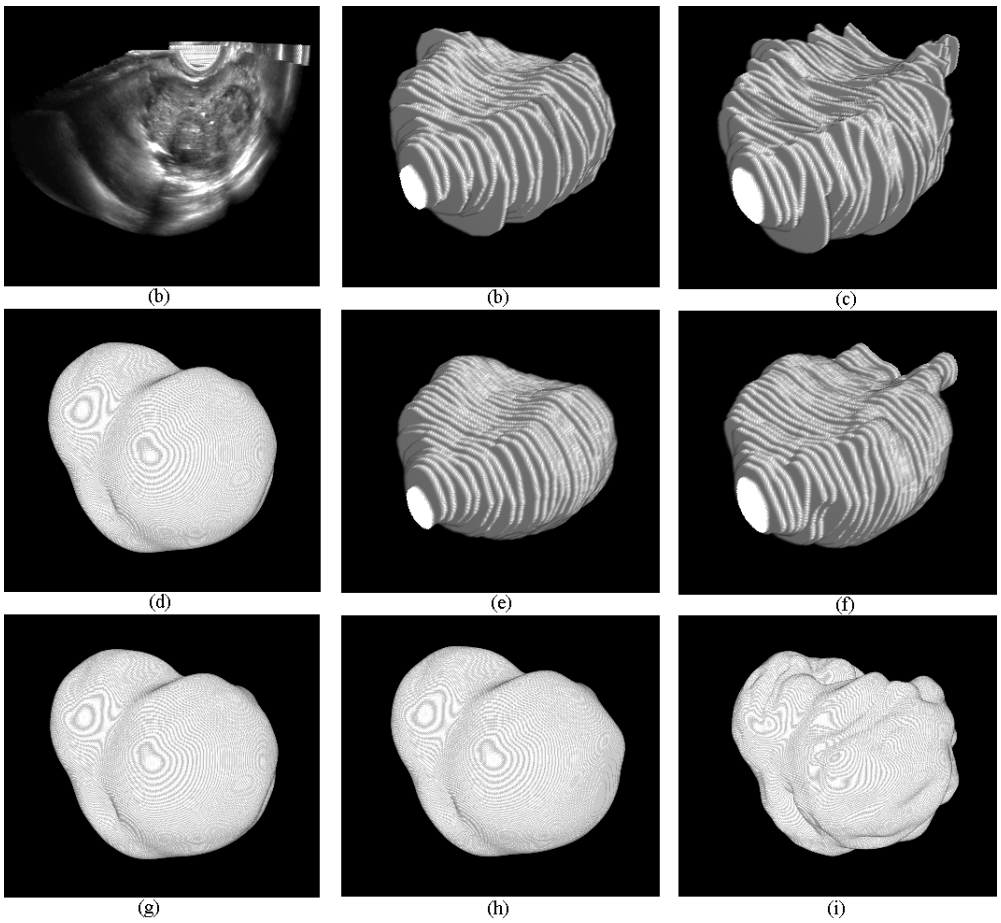

Figure 8.20: *Segmentation examples - clinical prostate scan 3. (a) Original clinical image 1 with cutway; (b) ground truth volume, doctor 1; (c) ground truth volume, doctor 2; (d) median filtering; (e) smoothed ground truth volume, doctor 1; (f) smoothed ground truth volume, doctor 2; (g) anisotropic diffusion filtering; (h) mean curvature evolution filtering; (i) curvature flow filtering.*

8.3 Average of Performance Metrics

To decide which preprocessing methods to use for the final segmentation procedure, the averages of the performance metrics are presented in this section. It is important to stress that these averages are not a *mean* in the strict mathematical sense. These are averages of performance across different images, which contain objects of varying sizes and hence are not independent measurements subject to some stochastic phenomena. Table 8.8 gives the averages of both the volume error and surface error metrics, with respect to the various speckle reduction methods. Each value was calculated from all of the images, including both IBS and non-IBS processed images. It should be noted that both metrics return purely positive values. From this data, it can be seen that the anisotropic diffusion (#3) and curvature flow filtering (#5) perform similarly. This is not the end of the story, however - next will be discussed how IBS performs vs. non-IBS. It was found that IBS hampered instead of increased performance accuracy, and once the IBS processed images are removed from this data, the anisotropic diffusion filtering performs the best. This will be presented shortly.

Table 8.8: *Average performance metrics across all images, for various speckle reduction methods. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering.*

Speck. Red. Meth.	Vol. Err. Avg. (%)	Surf. Err. Avg. (RMS, mm)
1	64.98	3.70
2	81.63	3.96
3	35.53	3.08
4	72.77	3.54
5	37.66	3.29

Table 8.9 shows the performance metrics for the non-IBS processed images verses the one with IBS processing (for all speckle reduction methods). Note that all of the images (simulated, phantoms, clinical) were used to calculate these values for the non-IBS case, although only the simulated and phantom

images were used for the IBS cases since the IBS data was not available for the clinical image volumes. From this data, it was apparent that the segmentation accuracy was better for the non-IBS processed images, and that IBS should not be used in the final segmentation routine for the best performance.

Table 8.9: *Average metric performance with IBS and without IBS calculations, using all available images* and for all speckle reduction methods. *Non-IBS images used for these values: simulated, phantom, clinical. IBS images used: simulated, phantoms.*

	Vol. Err. Avg. (%)	Surf. Err. Avg. (RMS, mm)
non-IBS	45.75	3.49
IBS	79.78	3.56

Table 8.10 shows the data from Table 8.8 without including the IBS processed images. Interestingly, without IBS, the anisotropic diffusion filtering method clearly demonstrates performance superior to the rest of the speckle reduction schemes, notably within the volume error metric, whereas performance was similar to the curvature flow filtering method when the IBS images were included.

Table 8.10: *Average performance metrics across all non-IBS processed images, for various speckle reduction methods. Speckle reduction types - (1) none; (2) median filtering; (3) anisotropic diffusion; (4) mean curvature evolution; (5) curvature flow filtering.*

Speck. Red. Meth.	Vol. Err. Avg. (%)	Surf. Err. Avg. [RMS mm]
1	58.24	3.80
2	48.14	3.38
3	26.76	3.22
4	56.04	3.43
5	39.58	3.62

8.4 Modified Evolution Parameters

As was mentioned previously, a global set of evolution parameters were used to evaluate the speckle reduction methods using a constant segmentation method across all images. By modifying these parameters for specific sets of images, segmentation accuracy can be improved. This section presents a few cases where segmentation accuracy has been improved by modifying (through trial and error) the strength coefficients (balloon force, curvature force) from the values used for evaluating the best preprocessing method.

Figure 8.21 shows a segmentation of simulated image 1. By increasing β from 0.30 to 0.40, the curve expands with greater force and results in a slightly better segmentation. Volume error = 7.66%, surface error = 1.32 mm. Original volume error = 7.21%, original surface error = 1.26 mm. In this case, the volume error has improved, although the surface error has increased slightly. This case shows arguably marginal improvement, although the segmentation was rather accurate to begin with. The ultrasound phantom and clinical image results show much better improvement.

Figure 8.22 shows a segmentation of the 35% cylinder phantom with $\beta = 0.40$, modified from 0.30, and $\gamma = 1.25$, modified from 1.50 and using anisotropic diffusion for preprocessing. By increasing the balloon force, the curve expansion increases and more closely approaches the true boundary. Also, by lowering the curvature strength coefficient, and the curve is able to expand into the charp corners. This image was processed with the anisotropic diffusion speckle reduction methods. Volume error = 10.09%, and surface error = 4.00 mm. Original volume error = 21.42%, original surface error = 4.90 mm.

Figure 8.23 shows a segmentation of clinical prostate image 1 with the balloon force (β) reduced from 0.3 to 0.2, preprocessed using anisotropic diffusion. The object boundaries are relatively weak in the clinical images as compared to the simulated and phantom images, and most of the clinical segmentations using the original parameters expanded beyond the actual

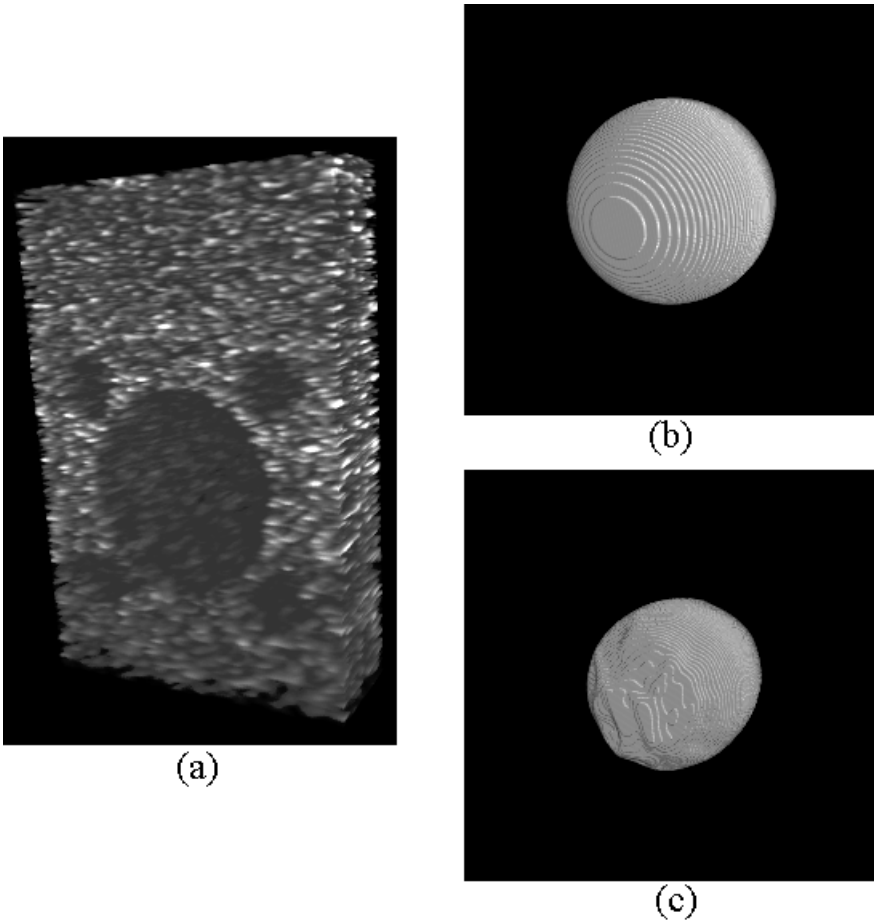

Figure 8.21: *Segmentation, simulated image 1, modified parameters. (a) Original simulated image 1 (with cutaway); (b) ground truth volume; (c) segmentation using anisotropic diffusion, with level set parameter* $\beta = 0.40$*, modified from* 0.30*. Volume error = 7.66%, surface error = 1.32 mm. Original volume error = 7.21%, original surface error = 1.26 mm*

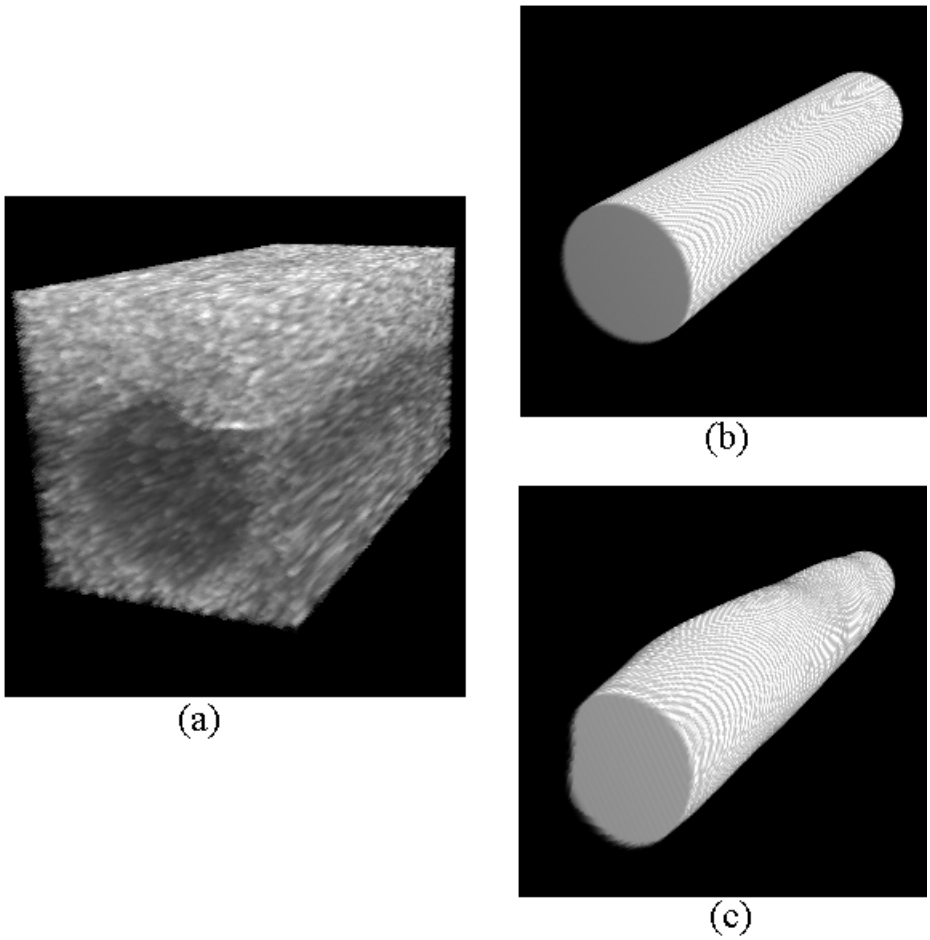

Figure 8.22: *Segmentation, cylinder phantom, 35% graphite, modified parameters. (a) Original cylinder phantom, 35% graphite (with cutaway); (b) ground truth volume; (c) segmentation using anisotropic diffusion, with* $\beta = 0.40$*, modified from* 0.30*, and* $\gamma = 1.25$*, modified from* 1.50*. Volume error = 10.09%, and surface error = 4.00 mm. Original volume error = 21.42%, original surface error = 4.90 mm.*

boundaries. By weakening the force expanding the curve, the segmentations settled closer to the actual boundaries and did not expand out beyond them. Volume error = 23.46% and 15.64% (vs. Doc1 and Doc2), surface error = 2.62 mm and 1.95 mm (vs. Doc1 and Doc2). Original volume error = 33.51% and 49.48% (vs. Doc1 and Doc2), original surface error = 3.35 mm and 3.78 mm (vs. Doc1 and Doc2).

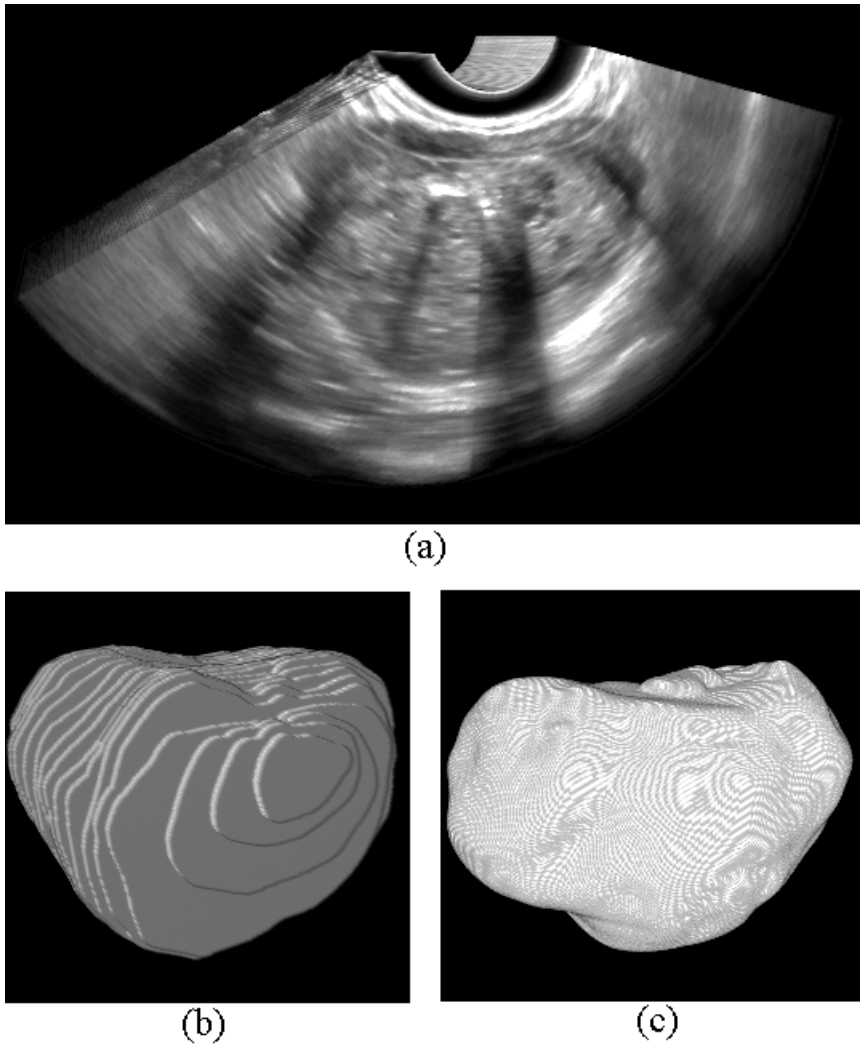

Figure 8.23: *Segmentation, clinical prostate scan 1, modified parameters. (a) Original clinical prostate image 1 (with cutaway); (b) smoothed ground truth volume, doctor 1; (c) segmentation using anisotropic diffusion, with level set parameter* $\beta = 0.20$*, modified from* 0.30*. Volume error = 23.46% and 15.64% (vs. Doc1 and Doc2), surface error = 2.62 mm and 1.95 mm (vs. Doc1 and Doc2). Original volume error = 33.51% and 49.48% (vs. Doc1 and Doc2), original surface error = 3.35 mm and 3.78 mm (vs. Doc1 and Doc2).*

8.5 Conclusions & Discussion

The final preprocessing scheme chosen to give the best segmentation performance was to perform anisotropic diffusion filtering on the non-IBS processed images. For objects with sizes on the order of 3 cm^3, the surface error metric gives a value of about 3.22 mm across the non-IBS processed images examined in this work. This is only slightly larger than the doctor-to-doctor discrepancies (prior to smoothing) in surface accuracies for the clinical images, which can be seen in Table 6.1. The volume error for the preprocessing scheme averages to about 27%. While at first glance this may seem a bit high, it must be kept in mind that the level set evolution parameters (see Section 5.2.4) were chosen to give decent performance across all these images, "decent" in that the segmentations would inflate with sufficient force, stop near the target boundaries, and not regularly leak throughout the entire image volumes. These parameters could be adjusted depending on the type of object being examined, to give the best segmentation performance in specific cases.

For example, the curvature force coefficient, β, which controls the amount of the surface tension-like force, has a certain tradeoff. With higher values, the evolving surface more easily bridges weak boundaries in the target boundaries and prevents leakage, but prevents the surface from taking on smaller, detailed shapes. This can be seen in the box phantom segmentations (see Figures 8.11 and 8.12), where the globally chosen value of β prevents the surface from expanding into the small corners of the boxes. This value does however prevent the surface from leaking through some of the very weak boundaries found in the clinical images. For mostly smooth-shaped organs, the current value is most likely sufficient. If a doctor were investigating a fluid volume in between organs (such as internal bleeding), which could take on shapes with very sharp corners, the doctor might select a "fluid volume" segmentation mode. This could relax the β term - which would probably work well, since the fluid boundaries would most likely have a high contrast

against the surrounding organs. Further work might even build classification algorithms into the system to use textural-based information to decide which type of target is being examined, and select an appropriate segmentation mode. By looking at the results presented in Section 8.4, it can be shown that for specific types of images (target objects), segmentation accuracy can be increased by fine tuning the level set parameters. These values did not have be changed very much in order to improve segmentation accuracy.

Chapter 9

Conclusions

In this work we have examined segmentation accuracy using various preprocessing methods on several test images. Three simulated 3D images, six 3D phantom images, and three 3D clinical prostate scans were used as the test image set. The first class simulated cysts on the order of 1-2 cm. They were of interest because the true target boundaries were known exactly. However, they were somewhat unrealistic in the sense that the cyst objects were very dark as compared the surrounding tissue, and that high amount of contrast is not typical of ultrasound images. The phantom images were created with varying levels of contrast to more closely mimic true conditions. Phantom cyst-mimicking objects were created with box and cylinder shapes, on the order of 1-3 cm. The box shapes were not particularly realistic in shape, as most anatomical shapes do not have sharp corners. However, when internal bleeding occurs and the fluid collects between the organs, this can take on shapes with very sharp corners and edges so the box phantoms had definite value in evaluating the segmentation routine and the speckle reduction techniques. An extra step needed to be taken when evaluating the surface error metric for the phantoms, as the exact locations of the objects were not known. An alignment was performed between the phantom segmentations and their respective ground truth models. The clinical images were arguably of the

most value, as they were actual ultrasound scans of real prostate images.

We have investigated histogram equalization methods (standard equalization, several specified histogram functions, perfectly flat equalization) and IBS with the nearest neighbor segmentation routine. It was found that the histogram equalization methods did not improve segmentation accuracy, although IBS was inconclusive. After this work was done, the nearest neighbor segmentation was abandoned in favor of a more advanced method (active contours using the level set method) that could be easily implemented directly in 3D. Using level set based segmentation, we have investigated the use of IBS and four different speckle reduction methods (median filtering, anisotropic diffusion filtering, mean curvature evolution and curvature flow filtering). It was found that IBS did not improve segmentation accuracy, and without IBS, anisotropic diffusion resulted in the best segmentations.

The preprocessing and segmentation methods took between 20 and 40 minutes for each image, depending on the size of the image volume and the number of iterations that the segmentation took to converge. Most segmentations converged after 200-600 iterations. However, these programs are in a prototype phase and there are a few optimizations that could be implemented that would reduce the required computation by at least an order of magnitude. Using multi-core CPUs or GPUs could reduce that time by another magnitude of order. This will be discussed a bit more in Chapter 10.

The level set parameters were held constant so that the preprocessing methods could be evaluated using the same segmentation method. Using these non-optimal parameters, segmentations were presented that were very close to the true boundaries, and in the case of the clinical images, on par with the doctor-to-doctor error for the same image volumes. Also presented were a few examples of segmentations where the parameters were modified for the specific type of target object, which showed improved segmentation accuracy. By changing these parameters depending on the type of target object, segmentation accuracy can be further improved.

In this work we have shown that active contours using the level set method are useful for 3D medical ultrasound segmentation and developed a prototype image processing and segmentation system for the mobile ultrasound project, that with increased optimization and dedicated hardware, could be brought down to a clinically useful processing time of 30 seconds to 1 minute. The segmentation routine is general enough that only a few parameters must be changed to segment many different types of anatomical target objects. We have explored the accuracy obtained after segmentation using four different popular filtering schemes for reducing speckle type noise, and found that anisotropic diffusion filtering resulted in the best segmentations. In addition to using a volume error metric to gauge segmentation accuracy, we have presented a surface error metric as a second quantitative measure. This segmentation module will be useful for the mobile ultrasound device by giving the clinician the ability to view 3D models of target objects and track volume and shape over time. This semi-automated segmenation procedure turns what would take hours by hand into a process that currently takes 20-40 minutes, and can be even further reduced with increased optimization, mentioned in Chapter 10. This takes the valuable time of doctors away from a tedious task and allows them to spend more time diagnosing and treating patients.

Chapter 10

Future Work

To fully automate the segmentation routine, a method of automatically placing seed points msut be implemented. Right now, one must manually place these seed points inside the target objects. One would probably look at textural methods to generate these seed points, and would benefit from using apriori information about the target objects under consideration (see [18], [47] and [48]).

Also, the segmentation routine is in a prototype phase and does not have any optimizations implemented to reduce computational load. The Narrowband Method, and the newer Parallel Sparse Field Method [46] are optimizations that drastically reduce the number of voxels within the level set fields that need to be updated for each iteration (requiring periodic reinitialization). The Narrowband determines which voxels are within a certain distance (band) of the current zero-level set, and only updates those, rather than every voxel in the entire field. The Parallel Sparse Field Method does the same thing, but implements a linked list for better efficiency and speed.

The preprocessing steps are also computationally cumbersome. Most ITK functions are multithreaded, so multi-core CPUs (central processing units) can benefit from running multiple threads at one. However, with the decreasing cost of GPU (graphics processing unit) units, which are essentially linear

algebra machines, it would be productive to look into offloading the computations from the CPU to a GPU. Another major advantage of GPUs is that they are massively parallel, so that many of the calculations can be done at once with many more than just 2 or 4 cores. NVIDIA has recently released CUDA [7], a package that allows one to program directly for the GPU (or wrap existing functions). In recent months, some ITK functions have even been ported to take advanatage of the parallel architecture of GPUs using CUDA.

Finally, a global set of level set evolution parameters were used across all of these images. This is not ideal, as each type of object has its own caveats, such as higher or lower contrast than other targets and different shapes (sharp corners or smooth boundaries). Some examples of improved segmentations using modified parameters were presented in Section 8.4. In the future, it would probably be benficial to determine exactly *which* objects (organs) this system should be able to segment, and come up with sets of parameters for each object type. Using this, the clinican would select "cardiac" mode, for example, and the segmentation would load those parameters for the cardiac segmentation once the image has been scanned (captured).

Bibliography

[1] Berkeley Software Distribution Certification Group, The BSD Certification Group Inc., http://www.bsdcertification.org, Accessed Apr. 11, 2008.

[2] Besl, P., McKay, N., "Apparatus for the registration of three-dimensional shapes", http://www.patentstorm.us/patents/5715166.html, Feb. 3, 1998, Accessed Apr. 11, 2008.

[3] Buchholz, Brooke, "Measuring acoustic properties of tissue mimicking phantoms using pulse-echo FIX" ultrasound", Boston University, Slide Presentation, http://www.censsis.neu.edu/Education/LSAMPREU/2003/presentations/bbuchholz.pdf#search=%22brook%20buchholz%2C%20%22phantom%22%22, Accessed 11/26/07.

[4] Caselles, V., *et al*, "*Geodesic Active Contours*", International Journal on Computer Vision, Vol. 22, No. 1, p. 61-69, 1997.

[5] Caselles, V., Catte, F., Coll, T., Dibos, F., "*A geometric model for active contours*", Numer. Math, Vol. 66, No. 1, 1999, pp. 1-31.

[6] Cross Platform Make, http://www.cmake.org/HTML/index.html, Kitware, Inc., Accessed Apr. 11, 2008.

[7] CUDA, NVIDIA Corporation, http://www.nvidia.com/object/cuda_home.html, Accessed Sep 13, 2008.

[8] Cygwin, http://www.cygwin.com/, Accessed Apr. 11, 2008.

[9] Dey, J., et al., "*Prior-shape-based segmentation of various objects in ultrasound images after speckle-reduction using level-set based curvature evolution*", Proceedings of the 2006 SPIE, Mar. 15, 2006.

[10] Electrical and Computer Engineering Department, Fred J. Looft, Department Head, Worcester Polytechnic Institute, http://www.ece.wpi.edu/, Accessed Sep. 12, 2008.

[11] "About the Eclipse Foundation", http://www.eclipse.org/org/, Eclipse Foundation, Accessed Apr. 11, 2008.

[12] "Off-The-Shelf Software Use in Medical Devices", http://www.fda.gov/cdrh/ode/guidance/585.html, Food and Drug Administration, Sep. 9, 1999, Accessed Apr. 11, 2008.

[13] "General Principles of Software Validation; Final Guidance for Industry and FDA Staff", http://www.fda.gov/cdrh/comp/guidance/938.html, Food and Drug Administration, Jan. 11, 2002, Accessed Apr. 11, 2008.

[14] FDA Guidelines for Software Development, http://www.itk.org/Wiki/FDA_Guidelines_for_Software_Developement, Jun. 11, 2007, Accessed Apr. 11, 2008.

[15] Field II, http://server.oersted.dtu.dk/personal/jaj/field/, Accessed 2/7/08.

[16] Fast Light Toolkit (FLTK), http://www.fltk.org/, Accessed April 14, 2008.

[17] GNU and the GCC compiler, http://gcc.gnu.org/, Accessed April 14, 2008.

[18] haralick, M. H., "*Statistical and Structural Approaches to Texture*", Proceedings of the IEEE, Vol. 76, N0. 5, May 1979, pp. 786.

[19] Holman, J.P., *Heat Transfer*, Ninth ed., McGraw-Hill, 2002, pp. 4.

[20] Insight Toolkit, http://www.itk.org/, Insight Software Consortium, Accessed Apr. 11, 2008.

[21] Ibanez, L., Schroeder, W., Ng, L., Cates, J., "The ITK Software Guide, 2nd ed., Updated for ITK version 2.4", Insight Software Consortium, Nov. 21, 2005.

[22] ITK Doxygen-generated documentation, Kitware, Inc., http://www.itk.org/Doxygen34/html/index.html, Accessed 2/7/08.

[23] ITK Copyright Notice, http://www.itk.org/HTML/Copyright.htm, Insight Software Consortium, Accessed Apr. 11, 2008.

[24] Jensen, Jrgen Arendt, Svendsen, Niels Bruun, "Calculation of Pressure Fields from Arbitrarily Shape, Apodized, and Excited Ultrasound Transducers", IEEE Trans. On Ultrasonics, Ferroelectrics and Frequency Control, Vol, 29, No. 2, March 1992, pp. 262-267.

[25] Jensen, Jrgen Arendt, Users guide for the Field II program, Release 2.86, August 17, 2001, pp. 7.

[26] Kichenassamy, S., *et al*, "*Conformal curvature flows: From phase transitions to active vision*", Arch. Rational Mech. Anal., Vol 134, No. 3, 1996, pp. 275-301.

[27] Kimmel, R., *Numerical Geometry of Images, Theory, Algorithms and Applications*, Springer, 2003.

[28] Kinsler, L., "Fundamentals of Acoustics", 4th ed., John Wiley and Sons, 2000, pp. 152.

[29] Kuntimad, G., Ranganath, H.S., "*Perfect image segmentation using pulse coupled neural networks*", IEEE Transactions on Neural Networks, Vol. 10, No. 3, May 1999, pp. 591-198.

[30] Levman, J., Alirezaie, J., Khan, G., "*Perfectly Flat Histogram Equalization*", Proceedings of the IASTED International Conference; Signal Processing, Pattern Recognition & Applications, pp. 38-41, June-July 2003.Gonzalez, R. C., Digital Image Processing, Addison-Wesley Publishing Company, 1993, pp. 161-182.

[31] Malladi, R. Sethian, J.A., "*Image Processing via level set curvature flow*", Proc. Natl. Acad. Sci. USA, July 1995, pp. 7046-7050.

[32] Malladi, P., Sethian, J.A., Vemuri, B.C., "*Shape modeling with front propagation*", IEEE Trans. Pattern Anal. Machine Intell., Vol. 17, Feb 1995, pp. 158-175.

[33] The Mathworks, Matlab 7.0.4, http://www.mathworks.com/, Accessed April 14, 2008.

[34] Matlab Documentation, *histeq.m*.

[35] More, J. J., "*Levenberg-Marquardt algorithm: Implementation and Theory*", Conference on Numerical Analysis, Dundee, U.K., June 19707, pp. 105-116.

[36] Osher, S., Sethian, J., "Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations", Journal of Computational Physics, 79, pp.12-49, 1988.

[37] Open Source Initiative, http://www.opensource.org/docs/osd, Jul, 7, 2006, Accessed Apr. 11, 2008.

[38] OpenGL, http://www.opengl.org/, Khronos Group, Accessed April 14, 2008.

[39] ParaView, Parallel Visualization Application, http://www.paraview.org, Kitware Inc., Accessed April 14, 2008.

[40] Pedersen, P.C., Mitra, M., Dey, J., "*Boundary detection in 3D ultrasound reconstruction using nearest neighbor map,*" Medical Imaging 2006: Ultrasonic Imaging and Signal Processing. Edited by Emelianov, Stanislav; Walker, William F. Proceedings of the SPIE, Volume 6147, San Diego, CA, Feb. 11 - 16, 2006, pp. 57-67, 2006.

[41] Penttinen, A., Luukkala, M., "The impulse response and pressure nearfield of a curved ultrasonic radiator", J. Phys. D., vol. 9, pp. 1547-1557, 1976.

[42] Perona, P., Malik, J., "*Scale-space and edge detection using anisotropic diffusion*", IEEE Transactions on Pattern Analysis Machine Intelligence, vol. 12, pp. 629-639, 1990.

[43] Phillips http://www.medical.philips.com/main/products/ultrasound/transducers/bptrt9_5.html, Accessed 2/7/08.

[44] Rowan, Matthew I., "An Injury-Mimicking Ultrasound Phantom as a Training Tool for Diagnosis of Internal Trauma", M.S. Thesis, Worcester Polytechnic Institute, Dec. 18, 2006.

[45] Sethian, J.A., *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Materials Science*, Cambridge University Press, 1996.

[46] Sethian, J.A., Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science, Cambridge University Press, 1999.

[47] Shapiro, L. G., Stockman, G. C., *Computer Vision*, Prentice Hall, 2001, pp. 63-73.

[48] Shen, D., Zhan, Y., Davatzikos, C., "*Segmentation of Prostate Boundaries From Ultrasound Images Using Statistical Shape Model*", IEEE Transactions on Medical Imaging, Vol. 22, No. 4, April 2003, pp. 539-551.

[49] Suri, J.S., Liu, K., Singh, S., Laxminarayan, S.N., Zeng, X., Reden, L., I., "Shape Recovery Algorithms using Level Sets in 2-D/3-D Medical Imagery: A State-of-the-Art Review", IEEE Transactions on Information Technology in Biomedicine, Vol. 6, No. 1, March 2002, pp. 8-28.

[50] Szabo, T. L., *Diagnostic Ultrasound Imaging, Inside Out*, Elsevier Academic Press, 2004, pp. 262-263.

[51] Telemedicine and Advanced Technology Research Center (TATRC), U.S. Army Medical Research & Materiel Command's (USAMRMC), http://www.tatrc.org/, Accessed Sep. 12, 2008.

[52] Ubuntu, http://www.ubuntu.com/, Canonical Ltd., Accessed Apr. 11, 2008.

[53] Ultrasound Research Laboratory, Worcester Polytechnic Institute, Department of Electrical and Computer Engineering, http://www.ece.wpi.edu/Research/Ultrasound/, Accessed Sep. 12, 2008.

[54] United States National Library of Medicine, National Institutes of Health, http://www.nlm.nih.gov/, accessed Apr. 11, 2008.

[55] The Visualization Toolkit (VTK), http://www.vtk.org/, Kitware, Inc., Accessed April 14, 2008.

[56] VolSuite, http://www.osc.edu/archive/VolSuite/, Accessed April 14, 2008.

[57] Weight, J.P., Hayman, A.J., "Observations of the propagation of very short ultrasonic pulses and their reflection by small targets", The Journal of the Acoustical Society of America, vol. 63, No. 2, pp. 396 404, February 1978.

[58] Worcester Polytechnic Institute, Worcester, Massachusetts, USA, http://www.wpi.edu/, Accessed Sep. 12, 2008.

[59] Xu, C., Yezzi, A., Prince, J., "On the Relationship between Parametric and Geometric Active Contours", Proc. of 34th Asilomar Conference on Signals, Systems, and Computers, pp. 483-489, October 2000.

[60] Yezzi, A., *et al*, "*A geometric snake model for segmentation of medical imagery*", IEEE Trans. Med. Imag., Vol 16, 1997, pp. 199-209.

[61] Yoo, T.S., Ackerman, M.J., Lorensen, W.E., Schroeder, W., Chalana, V., Aylward, S., Metaxes, D., Whitaker, R. Engineering and Algorithm Design for an Image Processing API: A Technical Report on ITK - The Insight Toolkit. In *Proc. of Medicine Meets Virtual Reality*, J. Westwood, ed., IOS Press Amsterdam 2002, pp. 586-592.

[62] Yushkevich, Paul A., Piven, J., Hazlett, H.C., Smith, R.G., Ho, S., Gee, J.C., and Gerig, G.. "User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability." Neuroimage. 2006 Jul 1; 31(3):1116-28.

[63] Zhang, Zhengyou, "*Iterative Point Matching for Registration of Free-Form Curves and Surfaces*", International Journal of Computer Vision, 13, 2, 1994, pp. 119-152.

Appendix A: Source Code

Median Filter, medfilt3d.cxx:

```
#if defined(_MSC_VER)
#pragma warning ( disable : 4786 )
#endif

#ifdef __BORLANDC__
#define ITK_LEAN_AND_MEAN
#endif


#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkMedianImageFilter.h"


int main( int argc, char * argv[] )
{
  if( argc < 3 )
    {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << "  inputImageFile   outputImageFile" << std::endl;
    return EXIT_FAILURE;
    }


  typedef   unsigned char  InputPixelType;
  typedef   unsigned char  OutputPixelType;


  typedef itk::Image< InputPixelType,  3 >   InputImageType;
  typedef itk::Image< OutputPixelType, 3 >   OutputImageType;
  typedef itk::ImageFileReader< InputImageType  >  ReaderType;
  typedef itk::ImageFileWriter< OutputImageType >  WriterType;


  ReaderType::Pointer reader = ReaderType::New();
  WriterType::Pointer writer = WriterType::New();
  reader->SetFileName( argv[1] );
  writer->SetFileName( argv[2] );


  typedef itk::MedianImageFilter<
              InputImageType, OutputImageType >  FilterType;
  FilterType::Pointer filter = FilterType::New();
```

```
45
46
47    InputImageType::SizeType indexRadius;
48    indexRadius[0] = 1; // radius along x
49    indexRadius[1] = 1; // radius along y
50    indexRadius[2] = 1; // radius along z
51
52    filter->SetRadius( indexRadius );
53    filter->SetInput( reader->GetOutput() );
54    writer->SetInput( filter->GetOutput() );
55    writer->Update();
56
57
58    return EXIT_SUCCESS;
59  }
```

Anisotropic Diffusion Filter, andiff3d.cxx:

```
1   #if defined(_MSC_VER)
2   #pragma warning ( disable : 4786 )
3   #endif
4
5   #ifdef __BORLANDC__
6   #define ITK_LEAN_AND_MEAN
7   #endif
8
9   #include "itkImage.h"
10  #include "itkImageFileReader.h"
11  #include "itkImageFileWriter.h"
12  #include "itkRescaleIntensityImageFilter.h"
13  #include "itkCurvatureAnisotropicDiffusionImageFilter.h"
14
15
16  int main( int argc, char * argv[] )
17  {
18    if( argc < 6 )
19      {
20      std::cerr << "Usage: " << std::endl;
21      std::cerr << argv[0] << "  inputImageFile  outputImageFile ";
22      std::cerr << "numberOfIterations  timeStep  conductance useImageSpacingon/off" <<
            std::endl;
23      return EXIT_FAILURE;
24      }
25
26
27    typedef     float     InputPixelType;
28    typedef     float     OutputPixelType;
29
30    typedef itk::Image< InputPixelType,  2 >   InputImageType;
31    typedef itk::Image< OutputPixelType, 2 >   OutputImageType;
32    typedef itk::ImageFileReader< InputImageType >  ReaderType;
33    typedef itk::CurvatureAnisotropicDiffusionImageFilter<
34                 InputImageType, OutputImageType >  FilterType;
35
36    FilterType::Pointer filter = FilterType::New();
37
38
39    ReaderType::Pointer reader = ReaderType::New();
40    reader->SetFileName( argv[1] );
41
42
43    filter->SetInput( reader->GetOutput() );
```

```
44
45
46    const unsigned int numberOfIterations = atoi( argv[3] );
47    const double       timeStep = atof( argv[4] );
48    const double       conductance = atof( argv[5] );
49    const bool         useImageSpacing = (argc != 6);
50    filter->SetNumberOfIterations( numberOfIterations );
51    filter->SetTimeStep( timeStep );
52    filter->SetConductanceParameter( conductance );
53    if (useImageSpacing)
54      {
55      filter->UseImageSpacingOn();
56      }
57    filter->Update();
58
59
60    typedef unsigned char WritePixelType;
61    typedef itk::Image< WritePixelType, 3 > WriteImageType;
62    typedef itk::RescaleIntensityImageFilter<
63                 OutputImageType, WriteImageType > RescaleFilterType;
64
65
66    RescaleFilterType::Pointer rescaler = RescaleFilterType::New();
67    rescaler->SetOutputMinimum(   0 );
68    rescaler->SetOutputMaximum( 255 );
69
70
71    typedef itk::ImageFileWriter< WriteImageType >  WriterType;
72    WriterType::Pointer writer = WriterType::New();
73    writer->SetFileName( argv[2] );
74    rescaler->SetInput( filter->GetOutput() );
75    writer->SetInput( rescaler->GetOutput() );
76    writer->Update();
77
78
79    return EXIT_SUCCESS;
80  }
```

Curvature Flow Filter, curvfilt3d.cxx:

```
1   #if defined(_MSC_VER)
2   #pragma warning ( disable : 4786 )
3   #endif
4
5   #ifdef __BORLANDC__
6   #define ITK_LEAN_AND_MEAN
7   #endif
8
9
10  #include "itkImage.h"
11  #include "itkImageFileReader.h"
12  #include "itkImageFileWriter.h"
13  #include "itkRescaleIntensityImageFilter.h"
14  #include "itkCurvatureFlowImageFilter.h"
15
16
17  int main( int argc, char * argv[] )
18  {
19    if( argc < 5 )
20      {
21      std::cerr << "Usage: " << std::endl;
```

```
22      std::cerr << argv[0] << "  inputImageFile  outputImageFile  numberOfIterations
            timeStep" << std::endl;
23      return EXIT_FAILURE;
24      }
25
26
27    typedef     float     InputPixelType;
28    typedef     float     OutputPixelType;
29    const unsigned int Dimension = 3;
30
31    typedef itk::Image< InputPixelType,  Dimension >   InputImageType;
32    typedef itk::Image< OutputPixelType, Dimension >   OutputImageType;
33    typedef itk::ImageFileReader< InputImageType >  ReaderType;
34    typedef itk::CurvatureFlowImageFilter<
35                InputImageType, OutputImageType >  FilterType;
36
37
38    ReaderType::Pointer reader = ReaderType::New();
39    reader->SetFileName( argv[1] );
40
41
42    FilterType::Pointer filter = FilterType::New();
43    filter->SetInput( reader->GetOutput() );
44
45
46    const unsigned int numberOfIterations = atoi( argv[3] );
47    const double       timeStep = atof( argv[4] );
48    filter->SetNumberOfIterations( numberOfIterations );
49    filter->SetTimeStep( timeStep );
50    filter->Update();
51
52
53    typedef unsigned char WritePixelType;
54    typedef itk::Image< WritePixelType, Dimension > WriteImageType;
55    typedef itk::RescaleIntensityImageFilter<
56                OutputImageType, WriteImageType > RescaleFilterType;
57    RescaleFilterType::Pointer rescaler = RescaleFilterType::New();
58
59
60    rescaler->SetOutputMinimum(   0 );
61    rescaler->SetOutputMaximum( 255 );
62
63
64    typedef itk::ImageFileWriter< WriteImageType >  WriterType;
65    WriterType::Pointer writer = WriterType::New();
66    writer->SetFileName( argv[2] );
67
68
69    rescaler->SetInput( filter->GetOutput() );
70    writer->SetInput( rescaler->GetOutput() );
71    writer->Update();
72
73
74    return EXIT_SUCCESS;
75  }
```

Subampling, SubsampleVolumeNoFilter.cxx:

```
1   #if defined(_MSC_VER)
2   #pragma warning ( disable : 4786 )
3   #endif
4
```

```
#ifdef __BORLANDC__
#define ITK_LEAN_AND_MEAN
#endif

#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkResampleImageFilter.h"
#include "itkIdentityTransform.h"
#include "itkNearestNeighborInterpolateImageFunction.h"
#include "itkCastImageFilter.h"
#include "itkRegionOfInterestImageFilter.h"


int main( int argc, char * argv[] )
{
  if( argc < 12 )
    {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0]
      << "  inputImageFile  outputImageFile factorX factorY factorZ startROIx startROIy
          startROIz sizeROIx sizeROIy sizeROIz"
      << std::endl;
    return EXIT_FAILURE;
    }


  const     unsigned int    Dimension = 3;

  typedef   unsigned char   InputPixelType;
  typedef   float           InternalPixelType;
  typedef   unsigned char   OutputPixelType;

  typedef itk::Image< InputPixelType,    Dimension >   InputImageType;
  typedef itk::Image< InternalPixelType, Dimension >   InternalImageType;
  typedef itk::Image< OutputPixelType,   Dimension >   OutputImageType;
  typedef itk::RegionOfInterestImageFilter< InputImageType,
                                            OutputImageType > FilterType;


  FilterType::Pointer roifilter = FilterType::New();
  OutputImageType::IndexType startroi;
  startroi[0] = atoi( argv[6] );
  startroi[1] = atoi( argv[7] );
  startroi[2] = atoi( argv[8] );
  OutputImageType::SizeType sizeroi;
  sizeroi[0] = atoi( argv[9] );
  sizeroi[1] = atoi( argv[10] );
  sizeroi[2] = atoi( argv[11] );
  OutputImageType::RegionType desiredRegion;
  desiredRegion.SetSize(  sizeroi  );
  desiredRegion.SetIndex( startroi );
  roifilter->SetRegionOfInterest( desiredRegion );


  typedef itk::ImageFileReader< InputImageType  >  ReaderType;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName( argv[1] );


  const double factorX = atof( argv[3] );
  const double factorY = atof( argv[4] );
```

```
66  const double factorZ = atof( argv[5] );
67
68
69  try
70    {
71    reader->Update();
72    }
73  catch( itk::ExceptionObject & excep )
74    {
75    std::cerr << "Exception catched !" << std::endl;
76    std::cerr << excep << std::endl;
77    }
78
79
80  InputImageType::ConstPointer inputImage = reader->GetOutput();
81
82
83  typedef itk::CastImageFilter< InputImageType,
84                                InternalImageType >   CastFilterType;
85  CastFilterType::Pointer caster = CastFilterType::New();
86
87  roifilter->SetInput( reader->GetOutput() );
88  caster->SetInput( roifilter->GetOutput() );
89
90
91  const InputImageType::SpacingType& inputSpacing = inputImage->GetSpacing();
92
93  const double sigmaX = inputSpacing[0] * factorX;
94  const double sigmaY = inputSpacing[1] * factorY;
95  const double sigmaZ = inputSpacing[2] * factorZ;
96
97
98  typedef itk::ResampleImageFilter<
99                  InternalImageType, OutputImageType >  ResampleFilterType;
100
101 ResampleFilterType::Pointer resampler = ResampleFilterType::New();
102 typedef itk::IdentityTransform< double, Dimension >  TransformType;
103
104 TransformType::Pointer transform = TransformType::New();
105 transform->SetIdentity();
106 resampler->SetTransform( transform );
107
108
109 typedef itk::NearestNeighborInterpolateImageFunction<
110                              InternalImageType, double > InterpolatorType;
111 InterpolatorType::Pointer interpolator = InterpolatorType::New();
112
113
114 resampler->SetInterpolator( interpolator );
115 resampler->SetDefaultPixelValue( 0 ); // value for regions without source
116
117
118 OutputImageType::SpacingType spacing;
119 spacing[0] = inputSpacing[0] * factorX;
120 spacing[1] = inputSpacing[1] * factorY;
121 spacing[2] = inputSpacing[2] * factorZ;
122 resampler->SetOutputSpacing( spacing );
123 resampler->SetOutputOrigin( inputImage->GetOrigin() );
124 InputImageType::SizeType   inputSize =
125             inputImage->GetLargestPossibleRegion().GetSize();
126
127
```

```
  typedef InputImageType::SizeType::SizeValueType SizeValueType;
  InputImageType::SizeType    size;
  size[0] = static_cast< SizeValueType >( inputSize[0] / factorX );
  size[1] = static_cast< SizeValueType >( inputSize[1] / factorY );
  size[2] = static_cast< SizeValueType >( inputSize[2] / factorZ );


  resampler->SetSize( size );
  resampler->SetInput( caster->GetOutput() );


  typedef itk::ImageFileWriter< OutputImageType >  WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput( resampler->GetOutput() );
  writer->SetFileName( argv[2] );


  try
    {
    writer->Update();
    }
  catch( itk::ExceptionObject & excep )
    {
    std::cerr << "Exception caught !" << std::endl;
    std::cerr << excep << std::endl;
    }

  std::cout << "Resampling Done" << std::endl;


  return EXIT_SUCCESS;
}
```

Binary Thresholder, BinaryThresholdImageFilter.cxx:

```
#if defined(_MSC_VER)
#pragma warning ( disable : 4786 )
#endif

#ifdef __BORLANDC__
#define ITK_LEAN_AND_MEAN
#endif

#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkBinaryThresholdImageFilter.h"


int main( int argc, char * argv[] )
{
  if( argc < 7 )
    {
    std::cerr << "Usage: " << argv[0];
    std::cerr << " inputImageFile outputImageFile ";
    std::cerr << " lowerThreshold upperThreshold ";
    std::cerr << " outsideValue insideValue   "  << std::endl;
    return EXIT_FAILURE;
    }


  typedef  unsigned char  InputPixelType;
```

```
  typedef  unsigned char  OutputPixelType;
  typedef  itk::Image< InputPixelType,  3 >        InputImageType;
  typedef  itk::Image< OutputPixelType, 3 >        OutputImageType;
  typedef  itk::BinaryThresholdImageFilter<
              InputImageType, OutputImageType >   FilterType;
  typedef  itk::ImageFileReader< InputImageType >  ReaderType;
  typedef  itk::ImageFileWriter< OutputImageType > WriterType;


  ReaderType::Pointer reader = ReaderType::New();
  FilterType::Pointer filter = FilterType::New();


  WriterType::Pointer writer = WriterType::New();
  writer->SetInput( filter->GetOutput() );
  reader->SetFileName( argv[1] );
  filter->SetInput( reader->GetOutput() );


  const OutputPixelType outsideValue = atoi( argv[5] );
  const OutputPixelType insideValue  = atoi( argv[6] );
  filter->SetOutsideValue( outsideValue );
  filter->SetInsideValue(  insideValue  );


  const InputPixelType lowerThreshold = atoi( argv[3] );
  const InputPixelType upperThreshold = atoi( argv[4] );


  filter->SetLowerThreshold( lowerThreshold );
  filter->SetUpperThreshold( upperThreshold );
  filter->Update();


  writer->SetFileName( argv[2] );
  writer->Update();

  return EXIT_SUCCESS;
}
```

Morphological Opening, morph.cxx:

```
#if defined(_MSC_VER)
#pragma warning ( disable : 4786 )
#endif

#ifdef __BORLANDC__
#define ITK_LEAN_AND_MEAN
#endif

#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkBinaryBallStructuringElement.h"
#include "itkBinaryMorphologicalClosingImageFilter.h"
#include "itkBinaryMorphologicalOpeningImageFilter.h"


int main( int argc, char * argv[] )
{
  if( argc < 5 )
```

```
21      {
22      std::cerr << "Usage: " << std::endl;
23      std::cerr << argv[0] << "  inputImageFile  outputImageFile  radiusX(int)  radiusY(
            int)  radiusZ(int)" << std::endl;
24      return EXIT_FAILURE;
25      }
26
27    typedef     unsigned char     InputPixelType;
28    typedef     unsigned char     OutputPixelType;
29
30    const unsigned int Dimension = 3;
31    typedef itk::Image< InputPixelType,  Dimension > InputImageType;
32    typedef itk::Image< OutputPixelType, Dimension > OutputImageType;
33    typedef itk::ImageFileReader< InputImageType >   ReaderType;
34    typedef itk::BinaryBallStructuringElement<
35                        InputPixelType,
36                        Dimension  >                 KernelType;
37    typedef itk::BinaryMorphologicalOpeningImageFilter<
38                InputImageType, OutputImageType,
39                KernelType >                         FilterType;
40    typedef itk::RescaleIntensityImageFilter<
41                OutputImageType, OutputImageType >  RescaleFilterType;
42    typedef itk::ImageFileWriter< OutputImageType >  WriterType;
43
44
45    // Read image
46    ReaderType::Pointer reader = ReaderType::New();
47    reader->SetFileName( argv[1] );
48
49
50    // Create kernel (structuring element)
51    KernelType::RadiusType radius;
52    radius[0] = atoi( argv[3] );
53    radius[1] = atoi( argv[4] );
54    radius[2] = atoi( argv[5] );
55    KernelType kernel;
56    kernel.SetRadius( radius );
57    kernel.CreateStructuringElement();
58
59
60    // Create morphological filter
61    FilterType::Pointer filter = FilterType::New();
62    filter->SetInput( reader->GetOutput() );
63    filter->SetKernel( kernel );
64
65
66    // Rescale for output
67    RescaleFilterType::Pointer rescaler = RescaleFilterType::New();
68    rescaler->SetOutputMinimum(   0 );
69    rescaler->SetOutputMaximum( 255 );
70    rescaler->SetInput( filter->GetOutput() );
71
72
73    // Write output
74    WriterType::Pointer writer = WriterType::New();
75    writer->SetFileName( argv[2] );
76    writer->SetInput( rescaler->GetOutput() );
77    writer->Update();
78
79
80    return EXIT_SUCCESS;
81  }
```

Make Cylinder, makecylinder.cxx:

```
#if defined(_MSC_VER)
#pragma warning ( disable : 4786 )
#endif

#include "itkSpatialObject.h"
#include "itkCylinderSpatialObject.h"
#include "itkSpatialObjectToImageFilter.h"
#include "itkImageFileWriter.h"


int main( int argc, char *argv[] )
{
  if( argc < 12 )
    {
    std::cerr << "Missing Parameters " << std::endl;
    std::cerr << "Usage: " << argv[0];
    std::cerr << " outputFile radius height sx sy sz tx ty tz spx spy spz [radius(mm),
        height(mm), size(numVox), translation(mm) voxelSpacing[mm]]";
    return EXIT_FAILURE;
    }


  typedef itk::CylinderSpatialObject CylinderType;
  CylinderType::Pointer cylinder = CylinderType::New();

  cylinder->SetRadius(  atof( argv[2] )  );
  cylinder->SetHeight(  atof( argv[3] )  );


  typedef CylinderType::TransformType  TransformType;
  TransformType::Pointer transform = TransformType::New();


  typedef itk::Vector<float,3> VectorType;
  VectorType axis;
  axis[0]=1;
  axis[1]=0;
  axis[2]=0;

  transform->Rotate3D(axis,1.57079632679490,1);
  cylinder->SetObjectToParentTransform( transform );
  cylinder->ComputeObjectToWorldTransform();


  VectorType translation;
  translation[0] = atof( argv[7] );
  translation[1] = atof( argv[8] );
  translation[2] = atof( argv[9] );


  transform->SetTranslation( translation );
  cylinder->SetObjectToParentTransform( transform );
  cylinder->ComputeObjectToWorldTransform();


  typedef itk::Image<unsigned char,3> ImageType;
  typedef itk::ImageFileWriter<  ImageType  > WriterType;
  typedef itk::SpatialObjectToImageFilter< CylinderType, ImageType >
                                                    SpatialObjectToImageFilterType;

  WriterType::Pointer writer = WriterType::New();
```

```
61
62
63    SpatialObjectToImageFilterType::Pointer imageFilter = SpatialObjectToImageFilterType::
          New();
64
65    ImageType::SpacingType spacing;
66    spacing[0] = atof( argv[10]  );
67    spacing[1] = atof( argv[11]  );
68    spacing[2] = atof( argv[12]  );
69    imageFilter->SetSpacing( spacing );
70
71    ImageType::SizeType size;
72    size[0] = atoi( argv[4] );
73    size[1] = atoi( argv[5] );
74    size[2] = atoi( argv[6] );
75    imageFilter->SetSize( size );
76
77
78    imageFilter->SetInput(  cylinder  );
79    imageFilter->SetInsideValue( 255 );
80    imageFilter->Update();
81
82
83    ImageType::Pointer binaryMeshImage = imageFilter->GetOutput();
84    const char * outputFileName = argv[1];
85    writer->SetInput( binaryMeshImage );
86    writer->SetFileName(  outputFileName  );
87
88
89    try
90      {
91      writer->Update();
92      }
93    catch( itk::ExceptionObject & excep )
94      {
95      std::cerr << "Exception caught !" << std::endl;
96      std::cerr << excep << std::endl;
97      return EXIT_FAILURE;
98      }
99
100
101   return EXIT_SUCCESS;
102 }
```

Make Ellipse, makeellipse.cxx:

```
1  #if defined(_MSC_VER)
2  #pragma warning ( disable : 4786 )
3  #endif
4
5  #include "itkSpatialObject.h"
6  #include "itkEllipseSpatialObject.h"
7  #include "itkSpatialObjectToImageFilter.h"
8  #include "itkImageFileWriter.h"
9
10
11 int main( int argc, char *argv[] )
12 {
13   if( argc < 13 )
14     {
15     std::cerr << "Missing Parameters " << std::endl;
16     std::cerr << "Usage: " << argv[0];
```

```
17      std::cerr << " outputFile rx ry rz sx sy sz tx ty tz spx spy spz [radius(units),
            size(vox), translation(units), voxel spacing(units) ]";
18      return EXIT_FAILURE;
19      }
20
21
22    typedef itk::EllipseSpatialObject<3> EllipseType;
23    EllipseType::Pointer ellipse = EllipseType::New();
24
25    typedef itk::FixedArray<float,3> ArrayType;
26    ArrayType radius;
27    radius[0] = atof( argv[2] );
28    radius[1] = atof( argv[3] );
29    radius[2] = atof( argv[4] );
30    ellipse->SetRadius(  radius  );
31
32    typedef EllipseType::TransformType  TransformType;
33    TransformType::Pointer transform = TransformType::New();
34
35
36    typedef itk::Vector<float,3> VectorType;
37    VectorType translation;
38    translation[0] = atof( argv[8]  );
39    translation[1] = atof( argv[9]  );
40    translation[2] = atof( argv[10] );
41
42    transform->SetTranslation( translation );
43    ellipse->SetObjectToParentTransform( transform );
44    ellipse->ComputeObjectToWorldTransform();
45
46
47    typedef itk::Image<unsigned char,3> ImageType;
48    typedef itk::ImageFileWriter<  ImageType  > WriterType;
49    typedef itk::SpatialObjectToImageFilter< EllipseType, ImageType >
50                                                  SpatialObjectToImageFilterType;
51
52
53    WriterType::Pointer writer = WriterType::New();
54
55
56    SpatialObjectToImageFilterType::Pointer imageFilter = SpatialObjectToImageFilterType::
          New();
57
58
59    ImageType::SpacingType spacing;
60    spacing[0] = atof( argv[11]  );
61    spacing[1] = atof( argv[12]  );
62    spacing[2] = atof( argv[13]  );
63    imageFilter->SetSpacing( spacing );
64
65
66    ImageType::SizeType size;
67    size[0] = atoi( argv[5] );
68    size[1] = atoi( argv[6] );
69    size[2] = atoi( argv[7] );
70    imageFilter->SetSize( size );
71
72
73    imageFilter->SetInput(  ellipse  );
74    imageFilter->SetInsideValue( 255 );
75    imageFilter->Update();
76
```

```
77
78    ImageType::Pointer binaryMeshImage = imageFilter->GetOutput();
79    const char * outputFileName = argv[1];
80    writer->SetInput( binaryMeshImage );
81    writer->SetFileName(  outputFileName  );
82
83
84    try
85      {
86      writer->Update();
87      }
88    catch( itk::ExceptionObject & excep )
89      {
90      std::cerr << "Exception caught !" << std::endl;
91      std::cerr << excep << std::endl;
92          return EXIT_FAILURE;
93      }
94
95
96    return EXIT_SUCCESS;
97  }
```

Level Set Segmentation, levelsetGeodesic3Dinit.cxx:

```
1   #if defined(_MSC_VER)
2   #pragma warning ( disable : 4786 )
3   #endif
4
5   #ifdef __BORLANDC__
6   #define ITK_LEAN_AND_MEAN
7   #endif
8
9   #include "itkImage.h"
10  #include "itkGeodesicActiveContourLevelSetImageFilter.h"
11  #include "itkGradientMagnitudeRecursiveGaussianImageFilter.h"
12  #include "itkSigmoidImageFilter.h"
13  #include "itkRescaleIntensityImageFilter.h"
14  #include "itkBinaryThresholdImageFilter.h"
15  #include "itkImageFileReader.h"
16  #include "itkImageFileWriter.h"
17  #include "itkMinimumMaximumImageCalculator.h"
18  #include "itkSignedDanielssonDistanceMapImageFilter.h"
19  #include "itkAndImageFilter.h"
20
21
22  int main( int argc, char *argv[] )
23  {
24    if( argc < 11 )
25      {
26      std::cerr << "Missing Parameters " << std::endl;
27      std::cerr << "Usage: " << argv[0];
28      std::cerr << " inputImage initImage outputBinary outputImage";
29      std::cerr << " Sigma SigmoidAlpha SigmoidBeta";
30      std::cerr << " PropagationScaling CurvatureScaling AdvectionScaling NumIterations"
            << std::endl;
31      return 1;
32      }
33
34
35    typedef    float           InternalPixelType;
36    const      unsigned int    Dimension = 3;
37    typedef itk::Image< InternalPixelType, Dimension >  InternalImageType;
```

```
38
39
40  typedef unsigned char OutputPixelType;
41  typedef itk::Image< OutputPixelType, Dimension > OutputImageType;
42  typedef itk::BinaryThresholdImageFilter<
43                        InternalImageType,
44                        OutputImageType    >    ThresholdingFilterType;
45
46  ThresholdingFilterType::Pointer thresholder = ThresholdingFilterType::New();
47
48  thresholder->SetLowerThreshold( -1000.0 );
49  thresholder->SetUpperThreshold(     0.0 );
50
51  thresholder->SetOutsideValue(  0  );
52  thresholder->SetInsideValue(  255 );
53
54
55  typedef  itk::ImageFileReader< InternalImageType > ReaderType;
56  typedef  itk::ImageFileWriter<  OutputImageType  > WriterType;
57
58  ReaderType::Pointer reader1 = ReaderType::New();
59  ReaderType::Pointer reader2 = ReaderType::New();
60  WriterType::Pointer writer = WriterType::New();
61
62  reader1->SetFileName( argv[1] );
63  reader2->SetFileName( argv[2] );
64  writer->SetFileName( argv[3] );
65
66
67  typedef itk::RescaleIntensityImageFilter<
68                              InternalImageType,
69                              OutputImageType >   CastFilterType;
70  typedef   itk::GradientMagnitudeRecursiveGaussianImageFilter<
71                              InternalImageType,
72                              InternalImageType >  GradientFilterType;
73  typedef   itk::SigmoidImageFilter<
74                              InternalImageType,
75                              InternalImageType >  SigmoidFilterType;
76  typedef   itk::AndImageFilter< OutputImageType, OutputImageType,
77                              OutputImageType   >  AndImageFilterType;
78  typedef   itk::SignedDanielssonDistanceMapImageFilter< InternalImageType,
79                             InternalImageType  >  DanielsonFilterType;
80  typedef   itk::GeodesicActiveContourLevelSetImageFilter< InternalImageType,
81                             InternalImageType >   GeodesicActiveContourFilterType;
82  typedef   itk::MinimumMaximumImageCalculator< InternalImageType >
        MinMaxCalculatorType;
83
84
85  GradientFilterType::Pointer  gradientMagnitude = GradientFilterType::New();
86  SigmoidFilterType::Pointer sigmoid = SigmoidFilterType::New();
87  AndImageFilterType::Pointer andfilter = AndImageFilterType::New();
88  SparseFieldType::Pointer sparsefield = SparseFieldType::New();
89
90
91  sigmoid->SetOutputMinimum(  1.0  );
92  sigmoid->SetOutputMaximum(  0.0  );
93
94  DanielsonFilterType::Pointer danielsonFilter = DanielsonFilterType::New();
95  GeodesicActiveContourFilterType::Pointer geodesicActiveContour =
96                                    GeodesicActiveContourFilterType::New();
97
98
```

```
99  const double propagationScaling  = atof( argv[8] );
100 const double curvatureScaling    = atof( argv[9] );
101 const double advectionScaling    = atof( argv[10] );
102
103
104 geodesicActiveContour->SetPropagationScaling( propagationScaling );
105 geodesicActiveContour->SetCurvatureScaling( curvatureScaling );
106 geodesicActiveContour->SetAdvectionScaling( advectionScaling );
107
108
109 const unsigned int numIterations = atoi( argv[11] );
110 geodesicActiveContour->SetMaximumRMSError( 0.02 );
111 geodesicActiveContour->SetNumberOfIterations( numIterations );
112
113
114 gradientMagnitude->SetInput( reader1->GetOutput() );
115 sigmoid->SetInput( gradientMagnitude->GetOutput() );
116
117
118 geodesicActiveContour->SetInput(  danielsonFilter->GetOutput() );
119 geodesicActiveContour->SetFeatureImage( sigmoid->GetOutput() );
120
121
122 thresholder->SetInput( geodesicActiveContour->GetOutput() );
123 writer->SetInput( thresholder->GetOutput() );
124
125
126 const double sigma = atof( argv[5] );
127 gradientMagnitude->SetSigma(  sigma  );
128
129
130 const double alpha =  atof( argv[6] );
131 const double beta  =  atof( argv[7] );
132
133
134 sigmoid->SetAlpha( alpha );
135 sigmoid->SetBeta(  beta  );
136 std::cout << "Alpha = " << alpha << std::endl;
137 std::cout << "Beta = " << beta << std::endl;
138
139
140 danielsonFilter->SetInput( reader2->GetOutput() );
141
142
143 CastFilterType::Pointer caster1 = CastFilterType::New();
144 CastFilterType::Pointer caster2 = CastFilterType::New();
145 CastFilterType::Pointer caster3 = CastFilterType::New();
146 CastFilterType::Pointer caster4 = CastFilterType::New();
147 CastFilterType::Pointer caster5 = CastFilterType::New();
148
149
150 WriterType::Pointer writer1 = WriterType::New();
151 WriterType::Pointer writer2 = WriterType::New();
152 WriterType::Pointer writer3 = WriterType::New();
153 WriterType::Pointer writer4 = WriterType::New();
154 WriterType::Pointer writer5 = WriterType::New();
155
156
157 try
158   {
159   writer->Update();
160   }
```

```
  catch( itk::ExceptionObject & excep )
    {
    std::cerr << "Exception caught !" << std::endl;
    std::cerr << excep << std::endl;
    }


  caster5->SetInput( reader1->GetOutput() );
  andfilter->SetInput1( caster5->GetOutput() );
  andfilter->SetInput2( thresholder->GetOutput() );


  //  segmented result ANDed with original image
  writer5->SetInput( andfilter->GetOutput() );
  writer5->SetFileName( argv[4] );
  caster5->SetOutputMinimum(   0 );
  caster5->SetOutputMaximum( 255 );
  writer5->Update();

  std::cout << "Max. RMS error: " << geodesicActiveContour->GetMaximumRMSError() << std
      ::endl;
  std::cout << std::endl;
  std::cout << "No. elapsed iterations: " << geodesicActiveContour->GetElapsedIterations
      () << std::endl;
  std::cout << "RMS change: " << geodesicActiveContour->GetRMSChange() << std::endl;


  return 0;
}
```

Point Set Extraction, SurfaceExtraction.cxx:

```
//  This program will extract points along the boundary of a binary input image.
//  The input voxel values are {0, objectValue}.  Points are extracted from the
//  center of each voxel face where two adjacent voxels have differing values.
//  The double-precision floating-pt point coordinates are written to the
//  outputPtsFile like so:
//  x1 y1 z1
//  x2 y2 z2
//     ...
//  xN yN zN
//
//     John David Quartararo  February 2007

#if defined(_MSC_VER)
#pragma warning ( disable : 4786 )
#endif

#ifdef __BORLANDC__
#define ITK_LEAN_AND_MEAN
#endif

#include "itkImageFileReader.h"
#include "itkBinaryMask3DMeshSource.h"
#include "itkImage.h"
#include "itkMesh.h"
#include <itkMeshSpatialObject.h>
#include <itkSpatialObjectWriter.h>
#include <itkSpatialObjectReader.h>


int main(int argc, char * argv[] )
```

```
{


  if( argc < 4 )
    {
    std::cerr << "Usage: " << argv[0]
              << "  inputImageFile  outputPtsFile  objectValue " << std::endl;
    return EXIT_FAILURE;
    }


  // Typedefs
  const unsigned int Dimension = 3;
  typedef unsigned char  PixelType;

  typedef itk::Image< PixelType, Dimension > ImageType;
  typedef itk::Mesh<double>                  MeshType;
  typedef itk::MeshSpatialObject< MeshType > MeshSpatialObjectType;
  typedef itk::BinaryMask3DMeshSource< ImageType, MeshType >   MeshSourceType;
  typedef MeshSourceType::PointsContainer::ConstIterator       PtIterator;
  typedef itk::ImageFileReader< ImageType >  ReaderType;


  // Image reader
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName( argv[1] );


  // Attempt to read image
  try
    {
    reader->Update();
    }
  catch( itk::ExceptionObject & exp )
    {
    std::cerr << "Exception thrown while reading the input file " << std::endl;
    std::cerr << exp << std::endl;
    return EXIT_FAILURE;
    }


  //  Create Mesh source
  MeshSourceType::Pointer meshSource = MeshSourceType::New();

  const PixelType objectValue = static_cast<PixelType>( atof( argv[3] ) );

  meshSource->SetObjectValue( objectValue );
  meshSource->SetInput( reader->GetOutput() );


  //  Update the surface extracter
  try
    {
    meshSource->Update();
    }
  catch( itk::ExceptionObject & exp )
    {
    std::cerr << "Exception thrown during meshSource->Update() " << std::endl;
    std::cerr << exp << std::endl;
    return EXIT_FAILURE;
    }

```

```
93
94    //  Get mesh, set up iterator for writing points file
95    MeshSourceType::OutputMeshType::Pointer mesh = meshSource->GetOutput();
96    std::ofstream raw_mesh( argv[2] );
97
98    PtIterator ptIterator = mesh->GetPoints()->Begin();
99    PtIterator ptEnd      = mesh->GetPoints()->End();
100
101
102   //  Write points to file
103   while(ptIterator != ptEnd)
104   {
105         MeshSourceType::OPointType curPoint = ptIterator.Value();
106     raw_mesh << curPoint[0] << " " << curPoint[1] << " " << curPoint[2] << std::endl;
107     ++ptIterator;
108   }
109   raw_mesh.close();
110
111
112   return EXIT_SUCCESS;
113 }
```

Iterative Closest Point Alignment, icp3d.cxx:

```
1  #ifdef _WIN32
2  #pragma warning ( disable : 4786 )
3  #endif
4
5  #include "itkEuler3DTransform.h"
6  #include "itkEuclideanDistancePointMetric.h"
7  #include "itkLevenbergMarquardtOptimizer.h"
8  #include "itkPointSet.h"
9  #include "itkPointSetToPointSetRegistrationMethod.h"
10 #include "itkTransformFileWriter.h"
11 #include "itkTransformFileReader.h"
12 #include <iostream>
13 #include <fstream>
14
15
16 int main(int argc, char * argv[] )
17 {
18
19   if( argc < 5 )
20     {
21     std::cerr << "Arguments Missing. " << std::endl;
22     std::cerr <<
23       "Usage:  icp3D.exe  groundTruthPointsFile  segResultPointsFile  maxIterations
           transformFileName"
24       << std::endl;
25     return 1;
26     }
27
28
29   const unsigned int Dimension = 3;
30   typedef itk::PointSet< float, Dimension >    PointSetType;
31   PointSetType::Pointer groundTruthPointSet  = PointSetType::New();
32   PointSetType::Pointer segResultPointSet    = PointSetType::New();
33
34   typedef PointSetType::PointType         PointType;
35   typedef PointSetType::PointsContainer   PointsContainer;
36
37   PointsContainer::Pointer groundTruthPointContainer  = PointsContainer::New();
```

```
38  PointsContainer::Pointer segResultPointContainer = PointsContainer::New();
39  PointType groundTruthPoint;
40  PointType segResultPoint;
41
42
43  std::ifstream   groundTruthFile;
44  groundTruthFile.open( argv[1] );
45  if( groundTruthFile.fail() )
46    {
47    std::cerr << "Error opening points file with name : " << std::endl;
48    std::cerr << argv[1] << std::endl;
49    return 2;
50    }
51
52  unsigned int pointId = 0;
53  groundTruthFile >> groundTruthPoint;
54  while( !groundTruthFile.eof() )
55    {
56    groundTruthPointContainer->InsertElement( pointId, groundTruthPoint );
57    groundTruthFile >> groundTruthPoint;
58    pointId++;
59    }
60  groundTruthPointSet->SetPoints( groundTruthPointContainer );
61
62  // read point sets
63  std::ifstream   segResultFile;
64  segResultFile.open( argv[2] );
65  if( segResultFile.fail() )
66    {
67    std::cerr << "Error opening points file with name : " << std::endl;
68    std::cerr << argv[2] << std::endl;
69    return 2;
70    }
71
72  pointId = 0;
73  segResultFile >> segResultPoint;
74  while( !segResultFile.eof() )
75    {
76    segResultPointContainer->InsertElement( pointId, segResultPoint );
77    segResultFile >> segResultPoint;
78    pointId++;
79    }
80  segResultPointSet->SetPoints( segResultPointContainer );
81
82
83  // Set up the Metric
84  typedef itk::EuclideanDistancePointMetric<
85                                  PointSetType,
86                                  PointSetType>  MetricType;
87  typedef MetricType::TransformType              TransformBaseType;
88  typedef TransformBaseType::ParametersType      ParametersType;
89  typedef TransformBaseType::JacobianType        JacobianType;
90  MetricType::Pointer  metric = MetricType::New();
91
92
93  // Set up a Transform
94  typedef itk::Euler3DTransform< double >        TransformType;
95  TransformType::Pointer transform = TransformType::New();
96
97
98  // Optimizer Type
99  typedef itk::LevenbergMarquardtOptimizer       OptimizerType;
```

```
100
101   OptimizerType::Pointer optimizer = OptimizerType::New();
102   optimizer->SetUseCostFunctionGradient(false);
103
104
105   // Registration Method
106   typedef itk::PointSetToPointSetRegistrationMethod< PointSetType,
107                                                      PointSetType >
                                                         RegistrationType;
108   RegistrationType::Pointer   registration  = RegistrationType::New();
109
110
111   // Scale the translation components of the Transform in the Optimizer
112   OptimizerType::ScalesType scales( transform->GetNumberOfParameters() );
113   const double translationScale = 1000;   // dynamic range of translations --/ was
          1000.0
114   const double rotationScale    = 1.0;    // dynamic range of rotations --/ was 1.0
115   scales[0] = 1e6 / rotationScale;
116   scales[1] = 1e6 / rotationScale;
117   scales[2] = 1e6 / rotationScale;
118   scales[3] = 1.0 / translationScale;
119   scales[4] = 1.0 / translationScale;
120   scales[5] = 1.0 / translationScale;
121   unsigned long   numberOfIterations =  atoi(argv[3]);   // --/ was 2000
122   double          gradientTolerance  =  1e-8;    // convergence criterion --/ was 1e-4
123   double          valueTolerance     =  1e-10;   // convergence criterion --/ was 1e-4
124   double          epsilonFunction    =  1e-5;    // convergence criterion --/ was 1e-5
125   optimizer->SetScales( scales );
126   optimizer->SetNumberOfIterations( numberOfIterations );
127   optimizer->SetValueTolerance( valueTolerance );
128   optimizer->SetGradientTolerance( gradientTolerance );
129   optimizer->SetEpsilonFunction( epsilonFunction );
130
131   // Start from an Identity transform (in a normal case, the user
132   // can probably provide a better guess than the identity...
133   transform->SetIdentity();
134   registration->SetInitialTransformParameters( transform->GetParameters() );
135
136
137   // Connect all the components required for Registration
138   registration->SetMetric(        metric        );
139   registration->SetOptimizer(     optimizer     );
140   registration->SetTransform(     transform     );
141   registration->SetFixedPointSet( groundTruthPointSet );
142   registration->SetMovingPointSet(   segResultPointSet   );
143
144
145   // Do Registration
146   try
147     {
148     registration->StartRegistration();
149     }
150   catch( itk::ExceptionObject & e )
151     {
152     std::cout << e << std::endl;
153     return EXIT_FAILURE;
154     }
155   std::cout << "Solution = " << transform->GetParameters() << std::endl;
156   std::cout << "  (Phi_x, Phi_y, Phi_Z [Rad], Tx, Ty, Tz [Units]) " << std::endl;
157
158
159   // Write Resulting Transform to File
```

```
  itk::TransformFileWriter::Pointer writer;
  writer = itk::TransformFileWriter::New();
  writer->SetInput( transform );
  writer->SetFileName( argv[4] );
  try
   {
   writer->Update();
   }
  catch( itk::ExceptionObject & excp )
   {
   std::cerr << "Error while saving the transforms" << std::endl;
   std::cerr << excp << std::endl;
   return 0;
   }


  // Read Transform file to confirm write operation
  itk::TransformFileReader::Pointer reader;
  reader = itk::TransformFileReader::New();
  reader->SetFileName( argv[4] );
  try
    {
    reader->Update();
    }
  catch( itk::ExceptionObject & excp )
    {
    std::cerr << "Error while reading the transform file" << std::endl;
    std::cerr << excp << std::endl;
    std::cerr << "[FAILED]" << std::endl;
    return EXIT_FAILURE;
    }

  typedef itk::TransformFileReader::TransformListType * TransformListType;
  TransformListType transforms = reader->GetTransformList();
  itk::TransformFileReader::TransformListType::const_iterator it = transforms->begin();
  if(!strcmp((*it)->GetNameOfClass(),"Euler3DTransform"))
    {
    TransformType::Pointer euler3d_read = static_cast<TransformType*>((*it).GetPointer()
        );
    }
  else
    {
    return EXIT_FAILURE;
    }


  ParametersType parameters = transform->GetParameters();
  itk::Array<double> values = metric->GetValue( parameters );

  unsigned int numval = metric->GetNumberOfValues();
  double sumsq = 0;
  for(unsigned int i=0;i<numval;i++)
  {
        sumsq = sumsq + values[i]*values[i];
  }
  std::cout << "Final RMS = " << sqrt(sumsq/numval) << std::endl;

  return EXIT_SUCCESS;
}
```

Volume Error Metric, metricvol.cxx:

```
1  //     John David Quartararo  April 2008
2
3
4  #ifdef _WIN32
5  #pragma warning ( disable : 4786 )
6  #endif
7
8  #include <cmath>
9  #include "itkImage.h"
10 #include "itkImageFileReader.h"
11 #include "itkImageRegionConstIterator.h"
12
13
14 int main(int argc, char * argv[] )
15 {
16   if( argc < 4 )
17     {
18     std::cerr << "Arguments Missing. " << std::endl;
19     std::cerr <<
20       "Usage:  MetricVol.exe  GroundTruthBinaryImage  SegResultBinaryImage Verbose(1=on
           ,0=off)"
21       << std::endl;
22     return EXIT_FAILURE;
23     }
24
25   const unsigned int Dimension = 3;
26
27   typedef unsigned int                            PixelType;
28   typedef itk::Image< PixelType, Dimension >      ImageType;
29   typedef itk::ImageFileReader< ImageType >       ReaderType;
30   typedef itk::ImageRegionConstIterator< ImageType > IteratorType;
31
32
33   //  Read images
34   ReaderType::Pointer readerGT = ReaderType::New();
35   readerGT->SetFileName( argv[1] );
36
37   ReaderType::Pointer readerSeg = ReaderType::New();
38   readerSeg->SetFileName( argv[2] );
39
40   try
41     {
42     readerGT->Update();
43     readerSeg->Update();
44     }
45   catch ( itk::ExceptionObject &err)
46     {
47     std::cout << "ExceptionObject caught !" << std::endl;
48     std::cout << err << std::endl;
49     return -1;
50     }
51
52
53   //  Read Spacing Information, Calculate volume per voxel
54   const ImageType::SpacingType& spacingGT = readerGT->GetOutput()->GetSpacing();
55   const ImageType::SpacingType& spacingSeg = readerSeg->GetOutput()->GetSpacing();
56   const float volPerVoxGT = spacingGT[0] * spacingGT[1] * spacingGT[2];
57   const float volPerVoxSeg = spacingSeg[0] * spacingSeg[1] * spacingSeg[2];
58
59
60   //  Create Image Iterators
```

```
  IteratorType itGT( readerGT->GetOutput(), readerGT->GetOutput()->
      GetLargestPossibleRegion() );
  IteratorType itSeg( readerSeg->GetOutput(), readerSeg->GetOutput()->
      GetLargestPossibleRegion() );


  //  Iterate through images, count number of non-zero voxels
  double numVoxGT = 0;
  for ( itGT.GoToBegin(); !itGT.IsAtEnd(); ++itGT)
    {
    if (itGT.Get() != 0 )
      {
      numVoxGT++;
      }
    }

  double numVoxSeg = 0;
  for ( itSeg.GoToBegin(); !itSeg.IsAtEnd(); ++itSeg)
    {
    if (itSeg.Get() != 0 )
      {
      numVoxSeg++;
      }
    }


  //  Calculate volumes of Ground Truth and Segmented Results
  const float volGT = volPerVoxGT * numVoxGT;
  const float volSeg = volPerVoxSeg * numVoxSeg;


  //  Calculate volume error metric
  const float volDifference = fabs(volGT-volSeg);
  const float volError = 100*fabs(volGT-volSeg)/volGT;


  //  Display Results
  const int verbose  = atoi( argv[3] );
  if (verbose == 1)
    {
    std::cout << "volPerVoxGT = " << volPerVoxGT << std::endl;
    std::cout << "volPerVoxSeg = " << volPerVoxSeg << std::endl;
    std::cout << "numVoxGT = " << numVoxGT << std::endl;
    std::cout << "numVoxSeg = " << numVoxSeg << std::endl;
    std::cout << "Ground Truth Volume = " << volGT << " [Units^3]" << std::endl;
    std::cout << "Segmented Input Volume = " << volSeg << " [Units^3]" << std::endl;
    std::cout << "Volume Difference = " << volDifference << " [Units^3]" << std::endl;
    }
  std::cout << "Volume Error = " << volError << "%" << std::endl;


  return EXIT_SUCCESS;
}
```

Surface Accuracy Metric, metricsurf.cxx:

```
//  This program will return a distance metric for two given point sets.
//  The first point set is the ground truth. The second is the segmentation
//  result. The second set is traversed and for each point, the closest point
//  in the first set is found.  As such, every point in the 2nd set will be
//  used, although not every point in the first set may be accessed. The
//  distances are stored and the RMS value of those returned as the surface
```

```
7  //  accuracy metric.
8  //  The point set inputs are read in from a data file using the following format:
9  //  x1 y1 z1
10 //  x2 y2 z2
11 //     ...
12 //  xN yN zN
13 //
14 //     John David Quartararo  February 2007
15
16
17 #ifdef _WIN32
18 #pragma warning ( disable : 4786 )
19 #endif
20
21 #include "itkEuclideanDistancePointMetric.h"
22 #include "itkPointSet.h"
23 #include "itkEuler3DTransform.h"
24 #include <iostream>
25 #include <fstream>
26 #include <cmath>
27
28 int main(int argc, char * argv[] )
29 {
30  if( argc < 3 )
31     {
32     std::cerr << "Arguments Missing. " << std::endl;
33     std::cerr <<
34       "Usage:  MetricSurf.exe  GroundTruthPointsFile  SegResultPointsFile"
35       << std::endl;
36     return EXIT_FAILURE;
37     }
38
39   const unsigned int Dimension = 3;
40
41   typedef itk::PointSet< float, Dimension >   PointSetType;
42   PointSetType::Pointer GroundTruthPointSet = PointSetType::New();
43   PointSetType::Pointer SegResultPointSet   = PointSetType::New();
44   typedef PointSetType::PointType             PointType;
45   typedef PointSetType::PointsContainer       PointsContainer;
46   PointsContainer::Pointer GroundTruthPointContainer  = PointsContainer::New();
47   PointsContainer::Pointer SegResultPointContainer    = PointsContainer::New();
48   PointType GroundTruthPoint;
49   PointType SegResultPoint;
50
51
52   // Read pointset data files (GroundTruth/SegResult)
53   // Read the file containing coordinates of GroundTruth points.
54   std::ifstream   GroundTruthFile;
55   GroundTruthFile.open( argv[1] );
56   if( GroundTruthFile.fail() )
57     {
58     std::cerr << "Error opening points file with name : " << std::endl;
59     std::cerr << argv[1] << std::endl;
60     return 2;
61     }
62   unsigned int pointId = 0;
63   GroundTruthFile >> GroundTruthPoint;
64   while( !GroundTruthFile.eof() )
65     {
66     GroundTruthPointContainer->InsertElement( pointId, GroundTruthPoint );
67     GroundTruthFile >> GroundTruthPoint;
68     pointId++;
```

```
69      }
70    GroundTruthPointSet->SetPoints( GroundTruthPointContainer );
71
72
73    // Read the file containing coordinates of SegResult points.
74    std::ifstream   SegResultFile;
75    SegResultFile.open( argv[2] );
76    if( SegResultFile.fail() )
77      {
78      std::cerr << "Error opening points file with name : " << std::endl;
79      std::cerr << argv[2] << std::endl;
80      return 2;
81      }
82    pointId = 0;
83    SegResultFile >> SegResultPoint;
84    while( !SegResultFile.eof() )
85      {
86      SegResultPointContainer->InsertElement( pointId, SegResultPoint );
87      SegResultFile >> SegResultPoint;
88      pointId++;
89      }
90    SegResultPointSet->SetPoints( SegResultPointContainer );
91
92
93    // Set up the Metric
94    typedef itk::EuclideanDistancePointMetric< PointSetType,
95                                    PointSetType> MetricType;
96
97    MetricType::Pointer  metric = MetricType::New();
98
99    metric->SetFixedPointSet( GroundTruthPointSet );
100   metric->SetMovingPointSet( SegResultPointSet );
101
102
103   // Set up a transform
104   typedef itk::Euler3DTransform< double >       TransformType;
105   typedef MetricType::TransformType             TransformBaseType;
106   typedef TransformBaseType::ParametersType     ParametersType;
107
108   TransformType::Pointer transform = TransformType::New();
109
110   transform->SetIdentity();
111   metric->SetTransform( transform );
112   ParametersType parameters = transform->GetParameters();
113   metric->SetTransformParameters( parameters );
114
115
116   // Initialize metric calculation
117   try {
118     metric->Initialize();
119     }
120   catch( itk::ExceptionObject & e )
121     {
122     std::cout << "Metric initialization failed" << std::endl;
123     std::cout << "Reason " << e.GetDescription() << std::endl;
124     return EXIT_FAILURE;
125     }
126
127
128   // Display Results
129   unsigned int numval = metric->GetNumberOfValues();
130   itk::Array<double> values = metric->GetValue( parameters );
```

```
  double sumsq = 0;
  for(unsigned int i=0;i<numval;i++)
  {
        sumsq = sumsq + values[i]*values[i];
  }

  std::cout << "RMS = " << sqrt(sumsq/numval) << std::endl;


  return EXIT_SUCCESS;
}
```

Mean Curvature Evolution Filter, curvatureEvol.m, by Joyoni Dey [9]:

```
function [X] = curvatureEvol(time,dt,X)
% curve(time,dt,X)
% evolve the 0.5 level curve, curvature flow
%       time    evolution time
%       dt      time step (dt<0.5 for stability, empirical)
%       X       grayscale image, range [0,1]

t=1;

while t*dt<time
  Dx2=(DCx(X)).^2;
  Dy2=(DCy(X)).^2;
  deno = Dx2+Dy2;
  dX=(Dxx(X).*Dy2-2*DCx(X).*DCy(X).*Dxy(X)+Dyy(X).*Dx2)./(eps+deno);            % eps
      prevents divsion by 0

  X=X+dt*dX;
  t=t+1;
end


function B=DCx(A)
% DCx   Central derivative in x direction.
%       Periodic boundary conditions.
B=(A([2:end 1],:)-A([end 1:end-1],:))/2;

function B=DCy(A)
% DCy   Central derivative in y direction.
%       Periodic boundary conditions.
B=(A(:,[2:end 1])-A(:,[end 1:end-1]))/2;

function B=Dyy(A)
% Dyy   Second derivative in y direction.
%       Periodic boundary conditions.
B=(A(:,[2:end 1])+A(:,[end 1:end-1])-2*A);

function B=Dxx(A)
% Dxx   Second derivative in x direction.
%       Periodic boundary conditions.
B=(A([2:end 1],:)+A([end 1:end-1],:)-2*A);

function B=Dxy(A)
% Dxy   Mixed second derivative.
%       Periodic boundary conditions.
B=(A([2:end 1],[2:end 1])+A([end 1:end-1],[end 1:end-1])...
   -A([2:end 1],[end 1:end-1])-A([end 1:end-1],[2:end 1]))/4;
```

Point Set Subsample, ptsReduce.m:

```
function ptsReduce(inputFileName,keepRatio,spx,spy,spz)
%  This function takes a 3D point set, randomly selects a subset of points,
%  and writes those to a new file.  The keepRatio defines how many points
%  to keep - a keepRatio of 0.1 would mean 10% of the original points would
%  be written to the new file. spx, spy, and spz define the voxel spacing
%  of the 3D image file (MHD+RAW format) from which the point set was
%  derived. Different random subsets of points are chosen until a set is
%  found whose mean along each dimension is not larger that the smallest
%  voxel spacing. This is to ensure a proper subset has been selected that
%  has the same gross location as the full set.
%    John David Quartararo 2007

datain = load([inputFileName '.pts']);
[n m]  = size(datain);
meanin=mean(datain);
maxTolerance = min([spx, spy, spz]);

it = 0;
tolerance=1/eps;        %  Initialize to high value
dataout=0;
while tolerance>=maxTolerance  % Don't want the mean of the subsampled point set
                               %  along any dimension to change any more than the
                               %  smallest voxel spacing along any dimension
    it=it+1;
    rand('state', sum(100*clock));
    randindex = ceil(n*rand(ceil(n*keepRatio),1));
    dataout = datain(randindex,:);
    meanout=mean(dataout);
    tolerance=abs(meanin-meanout);
end

dlmwrite(['subsample\' inputFileName '_subsample.pts'],dataout,'delimiter',' ','newline'
     ,'pc');

it
tolerance
```

Perfectly Flat Histogram Equalization, hist_flat.m:

```
% Implementation of "Perfectly Flat Histogram Equalization"
%
% J. Levman, J. Alirezaie, G. Khan, "Perfectly Flat Histogram
% Equalization", Proceedings of the IASTED International Conference
% Signal Processing, Pattern Recognition &  Applications, pp. 38-41,
% June-July 2003.
%
% Code written by: John David Quartararo
% January 2006
%
% Phase 1: Histogram spike redistribution (omitted)
% Phase 2: Histogram matching (specification)
% Phase 3: Histogram smoothing
close all;
clear all;

%=========================================================================
% Read image file, set variables
fname   =  ['sim_ibs.bmp'];
in      =  imread(fname);
in      =  uint8(in);
```

```
[m,n]   =  size(in);
[a,b]   =  imhist(in);
target  =  ceil(m*n/256);    %  Target pixels per bin for a flat
     %    histogram
ph1out  =  in;

%=========================================================================
% Phase 2 : Histogram matching (specification)
% Create paper-defined "desired" histogram, use histeq() to "match" to %   this
dhist   = zeros(1,256);
dhist(1:4:256) = 4*target;
ph2out  = histeq(ph1out,dhist);

%=========================================================================
% Phase 3 : Histogram smoothing
ph3out  = phase3(ph2out);

%=========================================================================
% Write output to disk as 8-bit grayscale bitmap, with "flat_" prefix
imwrite(ph3out,['flat_' fname]);

function imout = phase3(imin);
% USAGE: imout = phase3(imin);
%
% Phase 3 of 3: Histogram smoothing using random pixel swapping
% Parent function: hist_flat.m
% Child function:  moverandpix.m
%
% Inputs:  imin:    An 8-bit (uint8) grayscale image, size(imin)=[m,n]
% Output:  imout:   An 8-bit (uint8) grayscale image, size(imout)=[m,n]
%
% Implementation:   Traverses histogram from I=0 -> I=255
% Implementation of "Perfectly Flat Histogram Equalization"
%
% J. Levman, J. Alirezaie, G. Khan, "Perfectly Flat Histogram
% Equalization", Proceedings of the IASTED International Conference
% Signal Processing, Pattern Recognition &  Applications, pp. 38-41,
% June-July 2003.
%
% Code written by: John David Quartararo
% February 2006

[m,n]       = size(imin);
[hgram,I]   = imhist(imin);
imout       = imin;
moved       = zeros(size(imin));
target      = ceil(m*n/256);       %  Target pixels per bin for a flat
                                   %  histogram
for bin = 1:256                 %  Each intensity value denoted a "bin"
    if hgram(bin) > target
        %  Move excess (wrt target) pixels into next bin
        binput = bin + 1;
        [imout,moved] = ...
            moverandpix(bin-1,binput-1,hgram(bin)-target,imout,moved);
        temp           = hgram(bin);
        hgram(bin)     = target;
        hgram(binput) = hgram(binput)+temp-target;
    elseif hgram(bin) < target
        %  Move pixels from (bintake>bin) into bin
        bintake = bin + 1;
        while hgram(bin)<target && bintake<=256
            if hgram(bintake)>=1
```

```
            %  Find next bin with pixels to take (bintake)
                if target - hgram(bin) >= hgram(bintake)
                    %  Move all pixels from bintake, (may need more
                    %  from another bin as well)
                    [imout,moved]  = ...
                        moverandpix(bintake-1,bin- ...
  1,hgram(bintake),imout,moved);
                    hgram(bin)     = hgram(bin) + hgram(bintake);
                    hgram(bintake) = 0;
                else
                    %  Move some pixels out of bintake to set
  %    bin=target
                    [imout,moved]  = moverandpix(bintake-1,bin- ...
1,target-hgram(bin),imout,moved);
                    hgram(bintake) = hgram(bintake) - (target- ...
hgram(bin));
                    hgram(bin)     = target;
                end
            end %if hgram(bintake)>=1
            bintake = bintake + 1;
        end %while hgram(bin)<target && bintake<=255
    end %if
end %for bin = 1:256

function [imout,movedout]=moverandpix(from,to,num,imin,movedin);
% USAGE: [imout,movedout]=moverandpix(from,to,num,imin,movedin)
%
% Inputs:  from:     Intensity value of pixels to be changed
%          to:       Intensity value to change pixels to
%          num:      How many pixels to change
%          imin:     An 8-bit (uint8) grayscale image (input)
%          movedin:  Matrix indicating which pixels have been changed
%                           before {size(movedin)=size(imin)}
% Outputs: imout:    An 8-bit (uint8) grayscale image
%                     with randomly changed pixels
%          movedout: Matrix indicating which pixels have been changed
%                     (cumulative from input, movedin), 1=moved,
%                           0=original
%                     size(movedout)=size(imout)
%
% Implementation of "Perfectly Flat Histogram Equalization"
%
% J. Levman, J. Alirezaie, G. Khan, "Perfectly Flat Histogram
% Equalization", Proceedings of the IASTED International Conference
% Signal Processing, Pattern Recognition &  Applications, pp. 38-41,
% June-July 2003.
%
% Code written by: John David Quartararo
% February 2006

movedout = movedin;
imout    = imin;

% Find indeces of image pixels with "from" grayscale value
fromindmov  = find(imin==from & movedin==1);
  % indeces of pixels with "from" value that have previously been moved
fromindnmov = find(imin==from & movedin==0);
  % indeces of pixels with "from" value that have not been moved before
numnmov = prod(size(fromindnmov)); % How many of each
nummov  = prod(size(fromindmov));

if nummov >= num
```

```
146 % There are enough pixels that have been moved previously
147 %  (Want to change "previously moved" pixels first)
148     rand('state',sum(100*clock));   % Seed random number generator
149     r = randperm(nummov);                    % Random indeces
150     imout(fromindmov(r(1:num)))     = to;
151 % Random pixels -> "to" value
152     movedout(fromindmov(r(1:num))) = 1;
153 % Update movedout matrix
154
155 else
156 % There are not enough pixels that have been moved previously
157 %   So, take some that have not been moved to make up difference
158     if nummov~=0
159     % First, move all available pixels that have been moved before
160     imout(fromindmov) = to;
161     end
162     % Then, move remaining pixels that have not been moved before
163     rand('state',sum(100*clock));   % Seed random number generator
164     r = randperm(numnmov);           % Random indeces
165     imout(fromindnmov(r(1:num-nummov)))=to;
166 % Random pixels -> "to" value
167     movedout(fromindnmov(r(1:num-nummov)))=1;
168 % Update movedout matrix
169 end
```