TESTING SPECT MOTION CORRECTION ALGORITHMS

by

Andrey Sklyar

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

Computer Science

April 2010

APPROVED

Professor Michael A. Gennert, Thesis Advisor and Department Head

Professor Mathew O. Ward, Thesis Reader

Abstract

Frequently, testing of Single Photon Emission Computed Tomography (SPECT) motion correction algorithms is done either by using simplistic deformations that do not accurately simulate true patient motion or by applying the algorithms directly to data acquired from a real patient, where the true internal motion is unknown. In this work, we describe a way to combine these two approaches by using imaging data acquired from real volunteers to simulate the data that the motion correction algorithms would normally observe.

The goal is to provide an assessment framework which can both: simulate realistic SPECT acquisitions that incorporate realistic body deformations and provide a ground truth volume to compare against. Every part of the motion correction algorithm needs to be exercised – from parameter estimation of the motion model, to the final reconstruction results.

In order to build the ground truth anthropomorphic numerical phantoms, we acquire high resolution MRI scans and motion observation data of a volunteer in multiple different configurations. We then extract the organ boundaries using thresholding, active contours, and morphology. Phantoms of radioactivity uptake and density inside the body can be generated from these boundaries to be used to simulate SPECT acquisitions.

We present results on extraction of the ribs, lungs, heart, spine, and the rest of the soft tissue in the thorax using our segmentation approach. In general, extracting the lungs, heart, and ribs in images that do not contain the spine works well, but the spine could be better extracted using other methods that we discuss.

We also go in depth into the software development component of this work, describing the C++ coding framework we used and the High Level Interactive GUI Language (HLING). HLING solved a lot of problems but introduced a fair bit of its own. We include a set of requirements to provide a foundation for the next attempt at developing a declarative and minimally restrictive methodology for writing interactive image processing applications in C++ based on lessons learned during the development of HLING.

Acknowledgements

I would like to thank Michael Gennert for his patience and mentoring support through the entire time of this work. I would like to thank Michael King and the UMass Medical School Nuclear Science Research Group for the opportunity to be a part of their group and research efforts. I would also like to thank Jonathan Word for his help in reviewing and polishing this report.

This work was supported in part by the National Institute of Biomedical Imaging and Bioengineering Grant No. R01 EB001457. The contents are solely the responsibility of the authors and do not necessarily represent the official views of the National Institutes of Health.

Contents

Table of Figures

Table of Figures

1 Background

Since all the work described in this project is motivated by and pertains to motion correction in SPECT, this introductory section provides a brief overview of Single Photon Emission Computed Tomography (SPECT) and SPECT motion correction.

1.1 SPECT Acquisition

Cardiac SPECT is a minimally invasive way to diagnose the severity of Coronary Artery Disease (CAD) (1) (2). In the procedure, a patient is injected with a radioisotope which travels through the blood stream and gets absorbed in various concentrations by the patient's tissue. Regions of reduced or blocked blood flow will diminish the amount of radioisotope that reaches and gets absorbed by the tissue; a doctor can use images of the radiation to indirectly locate these damaged areas and assess the severity of CAD (1) (3) (4) .

The imaging portion of the SPECT procedure takes between 15 and 30 minutes (3) (4) (5), during which the patient should lie as still as possible on a table in the imaging apparatus. Images of the radiation are taken from different angles by gamma cameras (as shown in Figure 1), with each exposure taking several seconds to accumulate enough data. The cameras' locations and orientations are known very precisely for each exposure, making it possible to reconstruct a 3D rendition of the radiation in the patient's body (6) (7) given that the patient's organs did not move a significant amount during the acquisition.

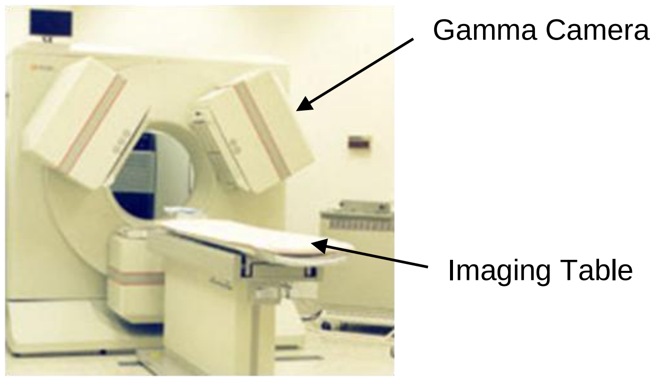


Figure 1 SPECT system with three Gamma Cameras

Typical SPECT reconstruction algorithms function under the assumption that the structures in the imaging volume do not deform during the acquisition procedure. Since the procedure is performed on living people, it is impossible to have all of their organs stay perfectly still. At the very least, the heart will beat and the lungs will expand and contract during the course of the acquisition. Their periodic nature, however, makes them less problematic than other types of motions, since the organs keep coming back to approximately the same location in each cardiac or respiratory cycle (8).

Motions such as bending, twisting, or any other type of the patient readjustment on the table cause difficulty since the organs being imaged are no longer in the same location between the different exposures (9). These types of motions can lead to the acquired data being completely unusable, in which case another acquisition would be required. Even worse, these artifacts may produce misleading reconstructions resulting in a misdiagnosis (10).

1.2 Motion Correction

SPECT Reconstruction algorithms calculate an estimate of the radiation density in the 3D volume where the patient was lying using the projection images acquired by the Gamma Cameras. In an ideal SPECT acquisition an infinite number of projections would be acquired instantaneously. The reconstruction algorithm would then use these projections combined with metadata regarding their acquisition (such as the position and imaging properties of the cameras) to reconstruct the 3D volume that was imaged (6).

Obviously, we cannot take an infinite number of views of a volume instantaneously. Instead, we have to use a small number of cameras (one, two (11) or three (12)) to acquire a limited number of views (up to 64). It takes between 15 and 30 minute to acquire these projection images, as each view requires time to place the camera and to take the exposures (10 to 30 seconds). The patient is asked to remain still during this lengthy procedure.

While the positioning of the cameras and the dosage of radiation can be controlled very precisely, the patient frequently does not remain still enough throughout the procedure to obtain a high quality reconstruction. Motion correction algorithms attempt to improve the quality of the reconstructed volumes by accounting for the patient's motion during the acquisition (9) (14) (15) (8) (16) (17). The issue we begin to address in this work is how to assess the quality of this improvement and how to help algorithm developers pinpoint the specific errors that their algorithms make, which they can then use to make refinements to their approaches.

2 SPECT

We motivate this section by describing one of the most important applications of SPECT: diagnosing the severity of Coronary Artery Disease. Then we go into the details of regular SPECT reconstructions, finishing with an overview of why they are inherently unable to account for patient motion.

2.1 Cardiac Stress Test

One of the simplest methods to assess the heart's ability to supply blood to the myocardium is the cardiac stress test. This test frequently referred to as the "treadmill test," as walking on treadmill is a common way to stimulate the heart (18). In situations where the patient's heart is suspected to be weakened or damaged, performing physical exercise may be too stressful, possibly leading to death (18). In these cases the patient will have their heart "exercised" by the injection of chemicals that will make the heart beat faster without requiring physical exercise (19).

The blood vessels supplying the heart may get obstructed due to the accumulation of plaque or blocked when a clot or vulnerable plaque breaks free and lodges itself in the vessel (20). One can assess the presence and/or severity of such blood flow restrictions by observing a patient's vital signs as the heart is made to work more heavily. This will require more blood to flow to the myocardium. Under normal conditions, the heart will beat faster and stronger to meet the body's needs. However, if the heart muscle is not receiving enough oxygenated blood due to an obstruction or blockage, vital signs will not go up as expected and the patient may feel some discomfort or chest pain (18).

2.2 Nuclear Stress Test

The cardiac stress test provides an understanding of the heart's overall condition. Nuclear imaging enables doctors to determine the specific parts of the heart which are damaged.

In the Nuclear Stress Test, the patient's heart is exercised similarly to how it is during the treadmill test. The radiation is injected at the peak of the stress period when blood is flowing the most rapidly through all the heart (except for in the parts that are blocked). Several minutes after the injection the regions which are not receiving blood will have observably less radiation than the healthy regions making it possible to locate arterial obstructions from images of the radiation (1) (23).

Nuclear imaging of the heart can also be performed as a reset study, without having the patient undergo stress. However, due to the decreased volume of blood flowing to the vessels the contrast between obstructed (as compared to blocked) blood flow and regular blood flow will be small, possibly even imperceptible. Since the stress acquisition does not suffer from this contrast reduction, the stress test results are more informative; if there are no obstructions during the stress study, then it is very unlikely that any will appear during the rest study. On the other hand if obstructions were found during a stress study, then a follow-up rest study can further explain their severity (24).

2.3 SPECT

There are several ways in which images of the radiation in a nuclear study can be acquired and used. With planar imaging (22), for example, the doctor examines the images of the radiation directly. In SPECT, a true 3D rendition of the radiation is generated by incorporating data from multiple views. As compared to planar imaging, slices extracted from SPECT reconstructions provide better localization of the radiation in the heart allowing doctors to better discriminate among the different types of coronary problems (25). In the rest of this section we describe the SPECT acquisition and reconstruction procedures in as much detail as is necessary to understand the rest of this report. For deeper understanding of SPECT, refer to (3) or (4).

2.3.1 Computed Tomography

SPECT is a type of Computed Tomography (CT) or Tomographic Reconstruction. Tomographic reconstruction is a process by which to generate a 3D rendition of a volume from 2D projections of that volume at known angles (25). X-ray CT (the modality commonly associated with CT) uses images of transmitted X-rays. Since denser tissue blocks more X-rays, X-ray CT reconstructions represent the spatially varying density inside the imaging volume. The projection images in SPECT come from observing photons emitted by the radionuclides that were injected into the patient's body. Consequently, the resulting reconstructions represent the radionuclide absorption densities by the patient's organs.

Though each modality has its own special characteristics that can be exploited during the acquisition and reconstruction stages, Computed Tomography has a very concrete high level structure. The acquisition process takes in the 3D volume of intensities (corresponding to tissue density in X-ray CT or concentration or radionuclides in SPECT) and produces a set of 2D projection images of the 3D volume at known angles. The reconstruction process takes in this set of 2D projections and produces a reconstructed 3D volume that estimates of locations of the intensities in the imaging volume. It is important to note that the imaging volume is only indirectly associated with a patient's organs. The reconstruction process executes irrespective of what was present in the imaging volume, and normal Computed Tomography reconstruction algorithms (including SPECT) assume this volume to be a stationary, rigid object.

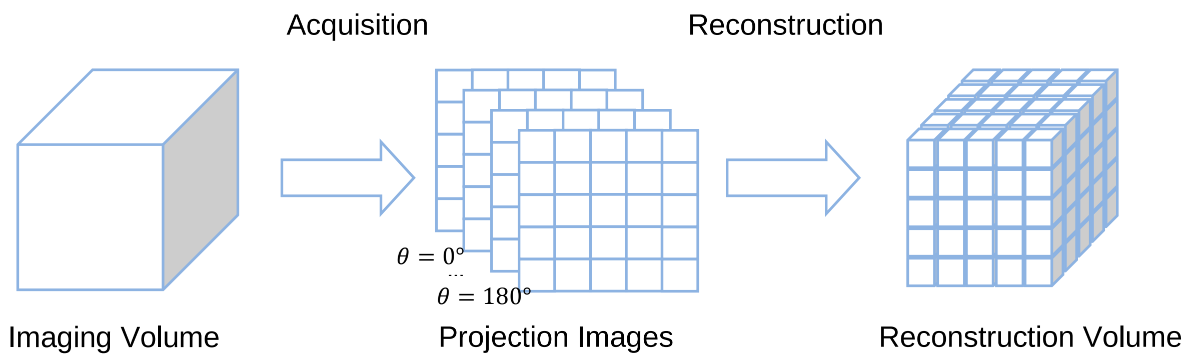


Figure 2 Computed Tomography Structures

2.3.2 SPECT Acquisition

The SPECT imaging process has several sources of degradation. Collimation decreases the number of photons that reach the imaging sensor in exchange for being able to reason as to where the photons came from. Attenuation reduces the number of photons that reach the gamma camera from tissue deeper inside of the patient's body. Scattering deflects photons from their original path, adding additional noise to the acquisition image. Crosstalk blurs the image at the collimator, while distance dependent spatial resolution makes structures further away from the gamma camera fuzzier. Understanding and accounting for the effects of these phenomena is necessary in order to perform the most accurate reconstructions possible.

Collimator

As the radioactive material in the patient's body decays, photons get emitted in random directions. This makes it impossible to tell which part of the body they came from using a photo-sensitive sensor alone. A parallel hole collimator attempts to filter out the photons coming in at non-perpendicular angles to the gamma camera by putting a lead plate with thousands of cylindrical holes in front of the photosensitive locations of the sensor (4) (3). Lead's high atomic weight and correspondingly large nuclei allow it to block photons that are not aligned with the orientation of the collimator holes, providing better localization of where they could have come from at the cost of filtering out more photons.

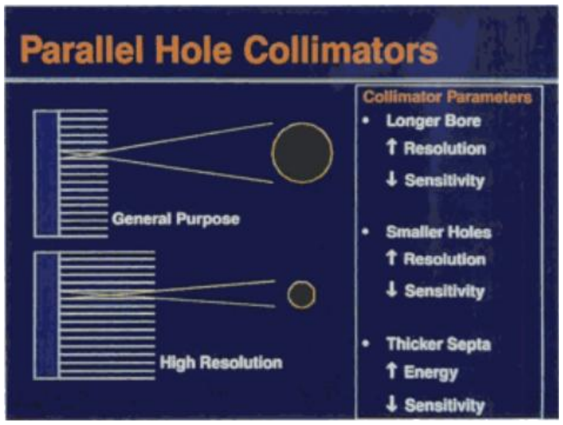


Figure 3 Parallel Hole Collimators from (4)

The photons that make it through the collimator do not, however, all originate from the same line perpendicular to the camera's surface. Since the cylinders have a finite radius and a finite height, photons slightly off axis from the ray going through the center of the cylinder also get observed by the same region of the imaging sensor (**Error! Reference source not found.**). ecreasing the radius or increasing the height of the cylinder narrows the spread of this cone and gives more localized information at the cost of filtering out more photons. Increasing the radius or decreasing the height of the cylinder increases the number of photons that can reach the imaging sensor but also increases the spread of the contributing cone and the probability that incoming photons came in off axis.

Crosstalk and Distance Dependent Spatial Resolution

Sometimes, photons that should have been detected in one bin get detected in a nearby bin instead. This situation is referred to as crosstalk. There are two primary ways that crosstalk can occur. In the first, a photon coming in off-axis through the collimator can get detected by an adjacent bin if the distance between the end of the collimator and the spacing of the bins is just right. The second contributions to crosstalk are the few photons that manage to penetrate the lead of the collimator without getting attenuated. Crosstalk causes blurring in the acquired images.

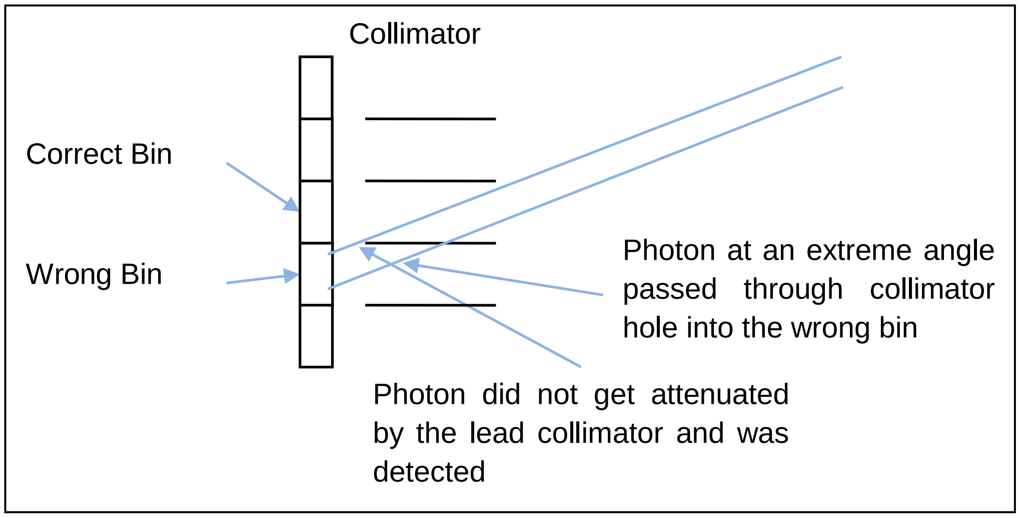

Collimator
Correct Bin
Wrong Bin
Photon at an extreme angle passed through collimator hole into the wrong bin
Photon did not get attenuated by the lead collimator and was detected

Figure 4 Crosstalk

Another consequence of the cone-shaped contribution volume permitted by collimation is that spatial resolution decreases for sources further away from the collimator. The distance dependent spatial resolution can be modeled by blurring the reconstruction volume with a Gaussian kernel of increasing σ at distances further away from the gamma camera. It also complicates reconstruction of the heart because it may be a different distance away from the camera at different angles, requiring the reconstruction algorithm to take these different resolutions into account.

Attenuation and Scattering

As a photon travels through imaging volume, it may interact with the nuclei of the other atoms that are in its path. Upon colliding with a nucleus, a photon may get absorbed (attenuated) and possibly re-emitted (scattered) at a longer wavelength (lower energy) and a different angle. Attenuation reduces the number of photons that reach the gamma camera. Scatter can either increase or decrease the counts depending on whether the re-emitted photon’s orientation lines up with the collimator.

The amount of attenuation that occurs is directly proportional to the number and density of nuclei on the path to the gamma camera. Thus, an X-ray CT scan of a patient can be used to construct an attenuation map of the imaging volume. This attenuation map can then be used by reconstruction algorithms to account for this type of degradation.

One can account for scatter by using an imaging sensor that can distinguish between energy levels of primary (non-scattered) and scattered photons (3).

2.3.3 SPECT Reconstruction

Reconstruction uses the projection images combined with models of the phenomena described in the Acquisition section to generate an estimate of the radioactivity distribution that was imaged. There are many reconstruction algorithms that have been developed and used in both research and the clinical setting (7) (6). Analytical, single pass algorithms are inherently fast, but they are limited in how much they can model and account for image degradation effects. Iterative algorithms can model and account for these factors, but require more computation. This section will review two of the most common reconstruction algorithms used in both research and the clinical setting - filtered back projection (single pass) and Maximum Likelihood Expectation Maximization (MLEM, iterative).

Filtered back projection has traditionally been used in clinical settings due to its fast reconstruction times. MLEM, however, provides better reconstruction results and can account for the degradation factors discussed in the Acquisition section. Even so, it has taken some time for MLEM to become prevalent in the clinical setting, despite formal recommendations and definitive evidence that doing attenuation correction significantly increases reconstruction quality (25).

Filtered Back Projection

Filtered Back Projection provides an exact analytical solution to the problem of reconstructing the imaging volume from ideal projection images (no attenuation, no scattering, etc.). We will describe at a high level the mathematical considerations involved in reconstructing a single 2D slice of the imaging volume. Refer to (28) for a full treatment of the subject. Reconstruction of the full volume may be seen as repeating this process for the remaining slices in the volume.

The value of a point (bin) ρ in a 1D projection image taken at a specific angle θ can be seen as a 1D integral taken through the 2D imaging slice $f(x,y)$ (Figure 5 illustrates why ρ can be thought of as the location of a bin). By repeating this integration for all bins at all angles, we get the full projection image $g(\rho,\theta)$, also known as the Radon Transform or Sinogram. as follows.

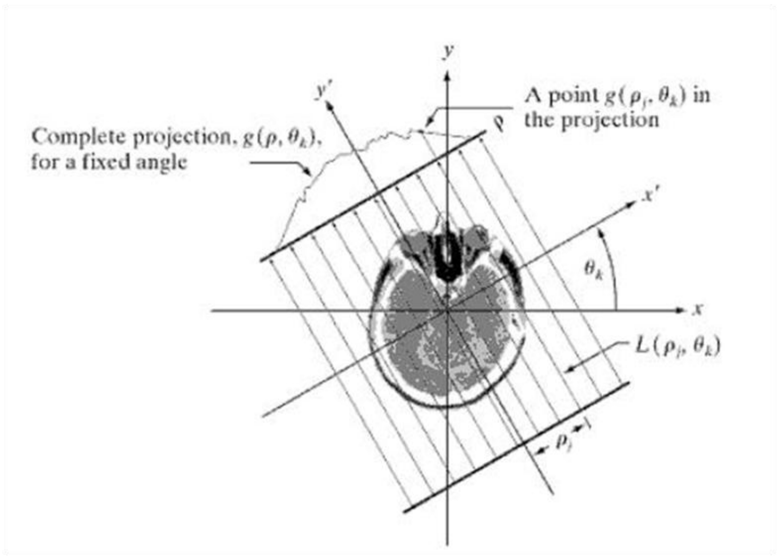

y
y'
A point $g(\rho_j, \theta_k)$ in the projection
Complete projection, $g(\rho, \theta_k)$, for a fixed angle
x'
θ_k
x
$L(\rho_j, \theta_k)$
ρ_j

Figure 5 Complete Projection for a Fixed Angle (Figure 5.37 from (28)

The Radon transform provides a mapping from f to g. The task at hand is derive the inverse Radon transform, mapping g back onto f. The key property which leads to the Filtered Back Projection reconstruction algorithm comes from the Fourier Slice Theorem. We refer the reader to (28) for a full derivation and treatment of Fourier Slice Theorem and only use the results here.

The Fourier Slice Theorem states that the 1D Fourier Transform of the projection image $g(\rho, \theta)$ at angle θ, $G(\omega, \theta)$, is exactly equal to the line of intensities passing through the origin at that same exact θ in the 2D Fourier Transform of $f(x, y)$, $F(u, v)$. We can use this relationship to come up with a mapping from $g(\rho, \theta)$ onto $F(u, v)$. We can then reconstruct $f(x, y)$ from $F(u, v)$ by applying the inverse Fourier Transform,

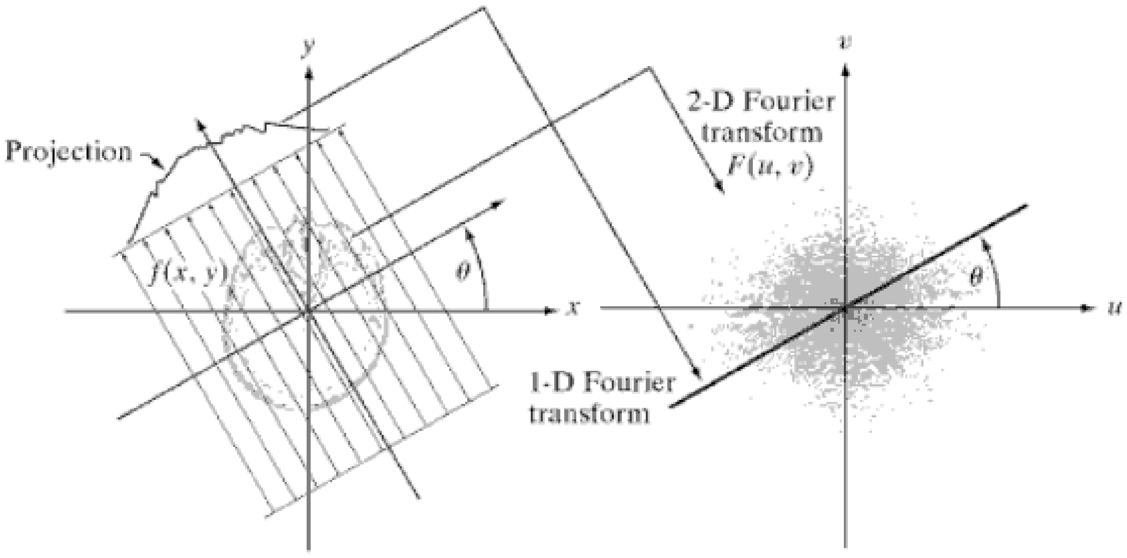


Figure 6 Fourier Slice Theorem

Let $F'_{\theta}(u,v)$ be the function defined by the thick black line in Figure 6

$$F'_{\theta}(u,v) = \begin{cases} G(\omega,\theta) \ \ if\ u\sin\theta + \ v\cos\theta = 0 \\ \qquad 0\ otherwise \end{cases}$$

Equation 1 $F'_{\theta}(u,v)$ A line from $F(u,v)$ oriented at angle θ through the origin

If we integrate $F^{'}_{\theta}(u,v)$ with respect to θ, we end up with $F^{'}(u,v)$ – a version of $F(u,v)$ where intensities closer to the origin (the lower frequency components) are over emphasized.

$$F^{'}(u,v) = \int_{0}^{2\pi} F'_{\theta}(u,v)\, d\theta$$

Equation 2 $F^{'}(u,v)$ Fourier Transform of Back Projected Reconstruction

We can correct for this over-emphasis by applying an infinite ramp filter in the Fourier domain to obtain the exact Fourier Transform. Taking the inverse Fourier Transform of this filtered version of $F^{'}(u,v)$ produces the original image $f(x,y)$. Alternatively, one can take the inverse Fourier Transform of $F^{'}(u,v)$ to get $f'(x,y)$ and convolve it with the inverse Fourier Transform of the ramp filter to obtain the same results since multiplication in the Fourier domain is a convolution in the spatial domain. The intermediate $f'(x,y)$ will be a blurry version of $f(x,y)$ because of the overemphasis of the lower frequency components.

In reality, we cannot multiply a function by an infinite ramp filter. We similarly cannot take the inverse Fourier Transform of an infinite ramp filter. Thus, we have to use a clipped version of the ramp filter. In order to remove the ringing artifact that comes from taking a Fourier Transform of a non-smooth function, a smoothed out version of the ram filter, as illustrated in Figure 7, is used. However, using these non-ideal filters inherently blurs the resulting reconstruction.

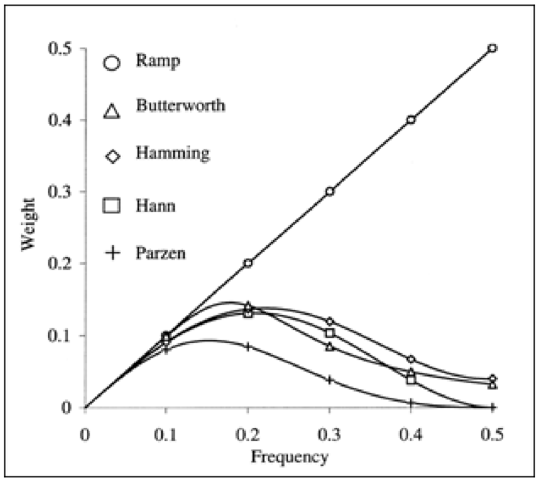


Figure 7 Ramp Functions from (6)

Generating F' and taking its inverse Fourier Transform as described above is a computationally intensive procedure. Fortunately, there exists an alternative procedure to generate F' from the projection images by back projecting them into the reconstruction volume. As illustrated in Figure 8 B, back projection copies the intensity values from one angle of the Sinogram into the reconstruction generating a smeared version of the one projection image. All the projections are incorporated into the reconstruction by adding their streaked images. If too few projections are used, a star artifact may appear (Figure 8 E). These smeared reconstructions approach F' as the number of projections goes to infinity. Convolving with the inverse Fourier Transform of the smoothed ramp filter takes care of the excessive blurring seen in these images.

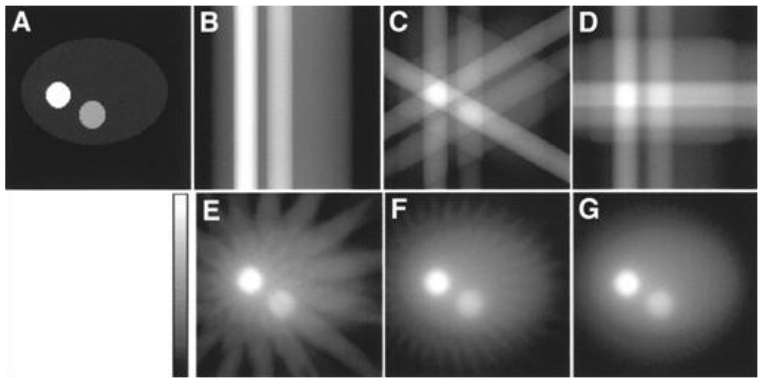


Illustration of star (or streak) artifact. (A) Slice used to create projections. (B–G) 1, 3, 4, 16, 32, and 64 projections equally distributed over 360° are used to reconstruct slice using backprojection algorithm. Activity in reconstructed image is not located exclusively in original source location, but part of it is also present along each line of backprojection. As number of projections increases, star artifact decreases.

Figure 8 Illustration of Filtered Back Projection from (6)

Figure 9 illustrates the Filtered Back Projection process, from acquisition through reconstruction. In a real acquisition $f(x, y)$ would never actually be available, with the reconstructed $\widehat{f}'(x, y)$ being the only view into the patient's body. When no degradation sources are present, FBP performs as expected, generating a very adequate reconstruction with some blurring.

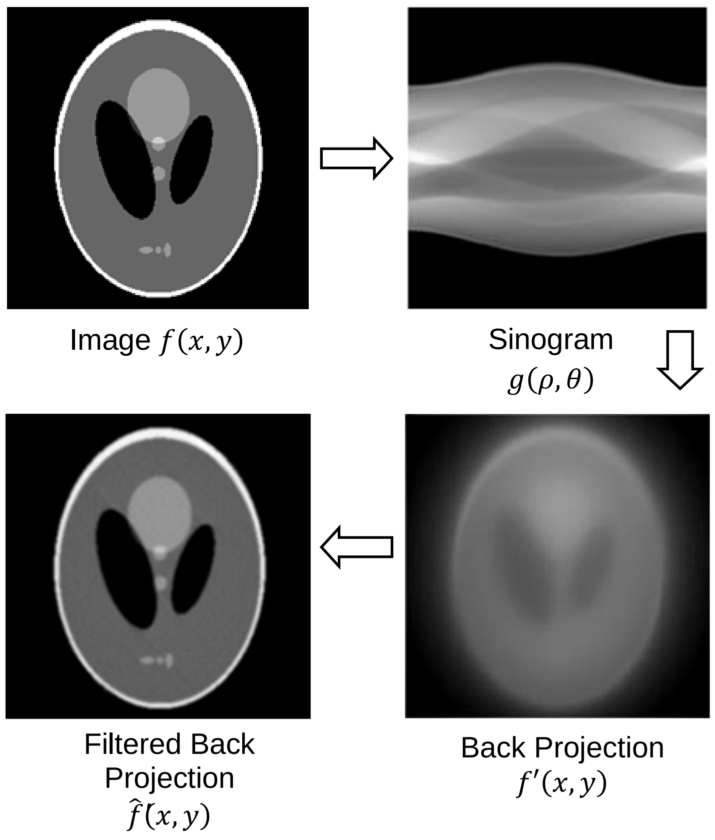


Image $f(x,y)$
Sinogram $g(\rho,\theta)$
Filtered Back Projection $\hat{f}(x,y)$
Back Projection $f'(x,y)$

Figure 9 Filtered Back Projection of a Phantom

Maximum Likelihood Expectation Maximization

Using all the understanding of the physics of the projection process, one can develop an accurate forward-projection matrix mapping the radioactivity in the voxels onto intensity values in the projection bins (3). Iterative algorithms use this projection system information to improve the estimate of the activity distribution in the imaging volume.

To model attenuation, one can weigh the contributions of each voxel based on the amount and density of tissue that is in the way from that voxel to the gamma camera. This attenuation map can be estimated from an X-Ray CT scan of the patient. The expected effects of scattering and distance dependent resolution can also be incorporated into the system matrix using this weighting approach.

Since all of the output of imaging process (the two dimensional images) can be modeled as weighted sums of the input (the three dimensional voxels), the entire process is linear and can be thought of as a matrix multiplication.

$$\vec{p} = A\vec{v}$$

Equation 3 Forward Projection

Where $\vec{v}$ is the vector of n voxel intensities, $\vec{p}$ is the vector of m bin intensities, and A is the $m \times n$ system matrix. Each element a_{ij} represents the probability that a photon emitted from voxel j is observed by bin i. The MELM algorithm uses the system matrix to generate the simulated projections.

The MLEM algorithm iteratively improves the estimate of the reconstructed volume (6). If $\vec{v}^k$ represents the estimate of the intensities in the voxels at iteration k of the MLEM algorithm, $\vec{p}^k = A\vec{v}^k$ is the simulated projection image at iteration k. If we call the bin values observed during the physical scan of the patient $\vec{p}$, then the value of each voxel in the next iteration computed by:

$$v_j^{k+1} = v_j^k \frac{\sum_{i=1}^{m} a_{ij} \frac{p_i}{p_i^k}}{\sum_{i=1}^{m} a_{ij}}$$

Equation 4 MLEM Update Step

This formula maximizes the log of the likelihood when modeling the radioactive decay process as a Poisson distribution (6) (7). It makes intuitive sense too: this can be interpreted as updating the value of the voxel by the amount that all the bins that it contributes to were off by from the true observation. In the case that there is no error, $\frac{p_i}{p_i^k} = 1$ and the entire expression simplifies to $v_j^{k+1} = v_j^k$. In reality, this update step is performed several times, but must be halted before convergence, as those reconstructions tend to incorporate too much of the noise present in the true projection operation back into the reconstruction volume (6). Instead a predefined stopping criterion, such as a number of iteration, is used.

2.3.4 Effects of Patient Motion

The system matrix describes what stationary voxels contribute to which collector bins in the projections. If the patient were to remain perfectly still during the acquisition, then the system matrix could be used directly to perform an accurate reconstruction. If the patient were to reposition him or herself during the acquisition, then photons that were emitted from the same relative location in the patient's body would have originated from different points in 3D space. The end result of this discrepancy is that the reconstructed volume does not accurately approximate the imaging volume.

Depending on the severity, direction, and duration of motion, false features called artifacts can appear in the 3D reconstruction. In some cases, especially related to large motions, the shape and size of the artifact is easily identifiable as being caused by motion (10). In those cases, a repeats study would be performed and the new set of projection images would be acquired. When the displacement is less pronounced, however, the artifacts can look like a valid organ structures and lead to a misdiagnosis (26). If the patient were actually healthy but was diagnosed as sick, then unneeded medical expenses would be incurred. On the other hand, a sick patient being diagnosed as healthy would delay treatment and allow a serious problem to be unattended.

3 Motion Correction in SPECT

There are three categories of patient motion that are common during SPECT: rigid body motion, periodic local deformations, and non-local deformations. Rigid body motions are those that could be performed if we assume that the patient is made of rigid substance. The only possible motions are translation and rotation in three dimensions (9) (27). Breathing (8) and heart beating (14) are periodic local deformations in the sense that the body deforms, but it follows the same relative path throughout the acquisition. Non-periodic non-rigid body deformations, however, are not constrained to return to the same relative location during the course of the acquisition that cannot be modeled using rigid body motion. An example would be twisting or bending on the table in order to get more comfortable (16).

Roughly speaking, rigid body motion is the easiest to correct because the motion model is so simple and, if the patient really did undergo rigid body motion, the motion parameters can be acquired in a straightforward way. On the other hand, not accounting for rigid body motion during reconstruction can lead to very severe motion artifacts (10). Periodic local deformations are harder to model and correct since they involve the body deforming repeatedly over the course of the acquisition. If uncorrected, their effects on image reconstruction, however, are primarily to blur the boundaries of the organ undergoing periodic motion, making the reconstructions slightly harder to interpret, but still useful in diagnose (8) (14). Finally, non-periodic non-rigid deformations are the hardest to model and have the most varying effects on image reconstruction. In the best case, the motion can occur in another part of the body and not affect the organ of interest, such as can happen if the patient were to move the arms without disturbing the torso during a cardiac SPECT procedure (28). In the worst case, the patient may bend or twist at the waist and readjust the shoulders to get more comfortable (16), disturbing exactly the region of interest.

3.1.1 Motion Observations, Modeling, and Correction

A series of observations over the course of the acquisition are used to derive model parameters that represent the patient's motion. The types of observations performed include: motion evident from the projection data alone or monitoring the output of sensors attached to the patient’s body

An example of motion that is evident from the projection data alone is a translation up or down the table. The projection images from adjacent angles should contain approximately the same structures in the same horizontal locations. If a global translation and horizontal direction leads to a better horizontal alignment between the structures in the two different projections, then the patient has most likely performed a global shift (31).

Some examples of external sensors used to collect motion data are an EKG to monitor heartbeat, which can be used to account for its periodic motion (14). Similarly, an elastic strap that measures chest expansion can be used to track the patient's breathing (8). Retroreflective markers on the patient's surface observed by a visual tracking system can be used to obtain the parameters for rigid body motion, respiration, and deformation (30).

These observations are then used to estimate the parameters of a motion model that the correction algorithm assumes. These include

- translation (31)
- six degree of freedom rigid body motion (translation and rotation) (9)
- affine transformation (12 degrees of freedom) (15)
- parameterized deformation (bending and twisting) (16)
- interpolated between table and patient’s surface (31)
- freeform deformation (32)

Motion data is incorporated into the reconstruction one of two ways: by deforming the projections and perform regular reconstruction (15), or by deforming the reconstruction volume during projection and back projection (9) (16) (33). The end goal is to get a stationary volume that most accurately represents the activity distribution within the patient's stationary body.

In order for a motion correction approach to be useful, its motion model needs to be

- Expressive enough to capture the types of motions that would arise in real studies and
- Have parameters that can be approximated accurately based on the observations

In general, the fewer the number of parameters, the more accurately they can be approximated (e.g. rigid body motion). Conversely, the larger the number of parameters, the more expressive the model can be (e.g. freeform deformations). This might seem to suggest that freeform deformation cannot be usefully corrected based. It could be if appropriate observation data were available. For example, a joint SPECT/MRI of the torso would provide the necessary data to perform non-rigid registration (32) (42) on the organs in the MRI volumes, which could then be applied to correct a cardiac SPECT reconstruction. SPECT/MRI systems have recently been developed for small animal imaging (36) (37), but are not yet available for human use.

3.1.2 Motion Correction Assessment

The assessment tools available to motion correction algorithm developers are physical phantoms (31) (9), numerical phantoms (34) (35) (36), or live patients. A positive feature of phantoms is that they provide a ground truth to compare one's results directly to. Numerical phantoms can model anatomy very accurately (34) (36), but the projection data has to be simulated. In both cases, incorporating realistic motion into the stationary phantoms is difficult. Live patient data provides the most accurate anatomy, projection physics, and motion, but the ground truth is not known.

Physical phantoms tend to be rigid water-filled structures with a simplified anatomical model. The types of motions that they can simulate are similarly limited - one can simulate rigid body motion accurately by moving the phantom (31) (9). Respiratory motion can also be crudely simulated by periodic translation of the phantom up-and-down the imaging table (33) (31). The projection physics, however, are exactly the same as would be expected in real captures, since the real radioactive material is observed using the real imaging apparatus.

Mathematical numerical phantoms such as the Mathematical Cardiac Torso (MCAT) Phantom (35) use simple mathematical formulas to model organs and their motions. Organs would be modeled using simple shapes such as cylinders and ellipses, and motion would be modeled by varying their parameters. Since the shapes are explicitly controlled by the parameters, one can easily determine corresponding regions between different motion states.

In order to generate more realistic organs, one can extract organ boundaries by segmenting volumetric images of patients, such as can be attained with MRI (36). This procedure produces stationary voxelized anthropomorphic phantoms. Developing even one such stationary phantom is usually a very time intensive procedure.

The NURBS-Based Cardiac Torso (NCAT) Phantom strikes an impressive balance between being anthropomorphic and mathematical (34). It is based on voxelized segmentations of MRI, but the organ boundaries are then modeled using Non-Uniform Rational B-Splines (NURBS) (42), giving them an infinite resolution, smooth, parametric model. This effectively makes it an anthropomorphic mathematical Phantom, since the parameters of the NURBS can be modified to place the model into different motion states. The NCAT Phantom incorporates models of heart beating and breathing based on real patient data (37). It does not, however, incorporate other types of deformations, such as bending and twisting.

3.1.3 Typical Assessment Approaches

In most cases, the assessment of the motion correction algorithm is limited to a motion model validation followed by an analysis of the algorithm's performance on clinical data containing motion (8) (9) (38). The motion model validation demonstrates that the algorithm can correct the type of motions for which it was designed. This is usually done by deforming a numerical phantom using the same motion model that the correction algorithm uses and then applying the correction algorithm to the doctored data (17). In the case of rigid body motion, one can also use a physical phantom in this check (9).

One way to provide a ground truth for non-periodic motion (the heart will beat and the patient will breathe) in a patient study is to perform a second acquisition with the same patient during which they wouldn't move (39). The two volumes would not necessarily be registered, but at least there would be a real basis for comparison of accuracy.

Another way to provide the ground truth is to use a dynamic anthropomorphic phantom that can simulate the type of realistic motion that the algorithm is attempting to address (40) (41). The NCAT phantom can simulate heart beating and breathing. (37). Since these deformations are actually based on real human motion, then running the correction algorithm with these data actually says something about how they would do on real patients.

Another benefit of using anthropomorphic numerical phantoms which incorporate realistic motion is that the observation data, such as the motion of markers on the patient's chest and abdomen, may be simulated as well and used to assess the quality of the parameter estimation. This differs from the validation described earlier, where the motion model parameters are usually specified directly.

Frequently, however, a ground truth is not available for the performance analysis (8) (9) (38). In those cases, the only comparison that can be made is between a usually unintelligible standard reconstruction and motion corrected output of the algorithm (38). At best, one can claim that the corrected results look like they have fewer detectable artifacts than the uncompensated ones, but since there is no ground truth to compare against, one cannot substantiate any claims about the correction approach’s effects on reconstruction accuracy.

4 Motion Correction Assessment Framework

The goal of this work is to create anthropomorphic numerical phantoms based on real human data that can be used to simulate non-periodic non-rigid body deformations. This framework will need to be able to exercise all parts of the SPECT motion correction - from parameter estimation, to motion corrected reconstruction, to quantitative assessment of reconstruction accuracy compared to a ground truth. Most importantly, since we intend to generate the test data MRI images of real human volunteers performing real deformations, the performance measures attained with these data can actually be used to substantiate claims about clinical performance.

Since the reconstruction procedure produces a numerical voxelized volume, the ground truth data has to at least be a voxel-based anthropomorphic numerical phantom with as fine or finer resolution. In order to simulate motion, these will have to be dynamic phantoms which can be deformed into several configurations. Since numerical phantoms will be used, we will need a model of the acquisition system (as described in the SPECT Reconstruction section) to generate projection data. Finally, we will need a method to generate the observation data needed by the motion correction portion of the reconstruction algorithm.

Figure 10 illustrates the steps involved in developing a motion simulation system from volumetric images for the purpose of testing SPECT motion correction algorithms. It is partitioned into two high level steps: Acquisition of the patient models and true internal motion and Simulation of the patient motion for use in assessing a motion estimation algorithm. The central section is the data that the Acquisition section provides to the Simulation section.

In the Acquisition section, volumetric images of the patient in different configurations are acquired via Magnetic Resonance Imaging along with the corresponding motion observation data (such as the 3D locations of markers on the patient's body). The volumes are then partitioned into different tissue types during segmentation, providing the boundaries of the organs as output. These boundaries are used to develop different phantoms of the patient, such as an organ density map and a model of the uptake of radioactivity in the body. These data combined with a forward projection procedure that models the equipment to be used in real acquisitions (not shown in diagram) provide all the necessary information to simulate a moving patient and be able to run the motion correction algorithm. Comparing the motion corrected reconstruction to the ground truth models used to generate the projection data provide a way to measure the accuracy of the algorithm.

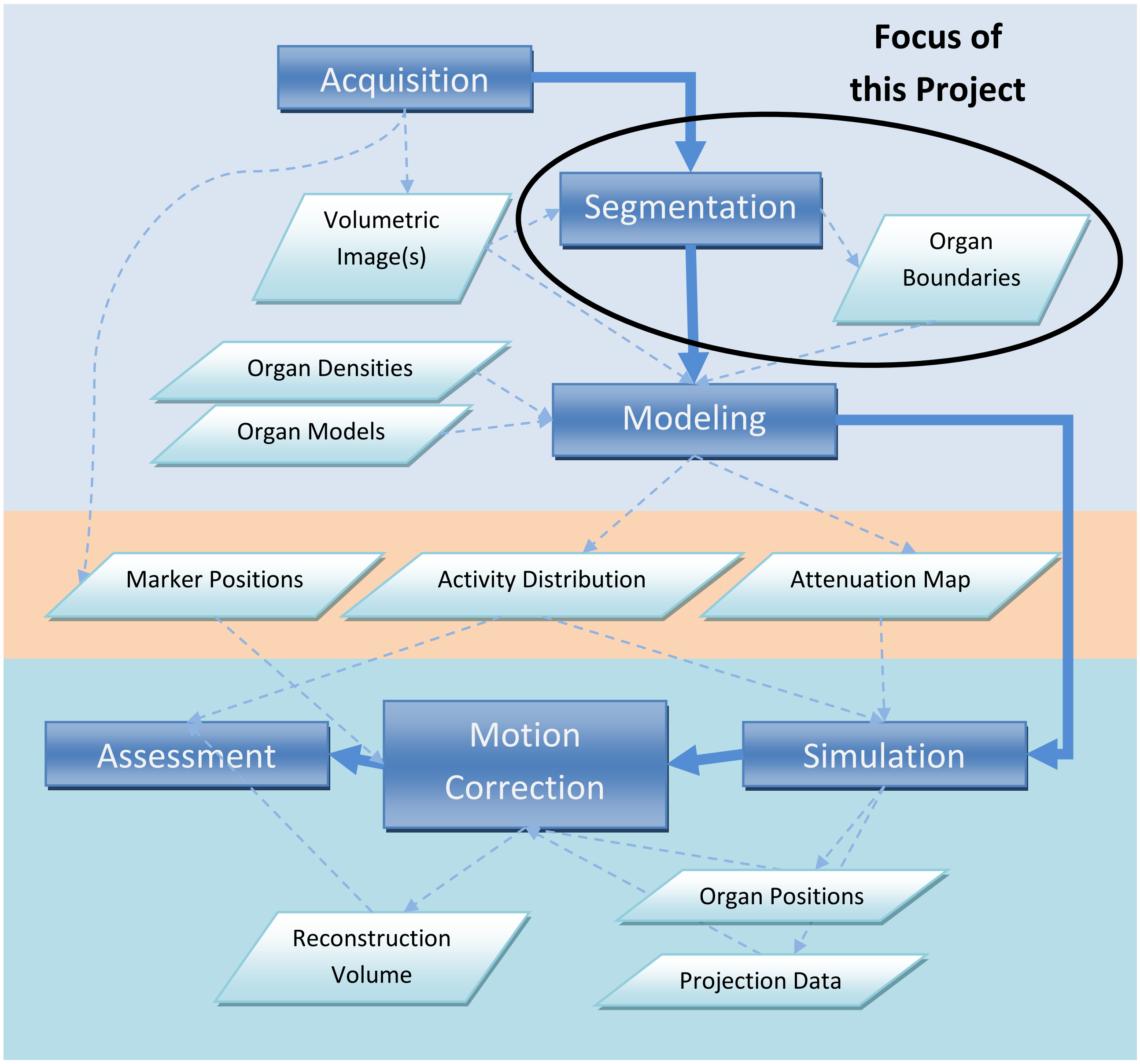


Figure 10 Motion Assessment Framework

4.1 Dynamic Anthropomorphic Numerical Phantoms

One can simulate motion by projecting from different stationary phantoms of the same person at different times during the simulated acquisition. The MCAT and NCAT Phantoms generate this set and any corresponding observation data by varying the parameters of their mathematically represented organ structures (35) (37). When such a mathematical model is not available, one can generate a set of voxelized numerical phantoms for the activity distribution and attenuation map based on segmented organ boundaries from high-resolution MRI images of the same volunteer in different configurations, as we do in this work. In that case, the observation data will have to be acquired at the same time as the volumetric images, since there will not be any explicit numerical deformations with which to generate the data.

4.1.1 Set of Volumes

In order to generate a set of anthropomorphic numerical phantoms, we start by acquiring a set of high resolution MRI scans of a real volunteer in various poses. We then segment the organ boundaries using techniques described later in this report. If we assume that the segmentations are accurate, each of the volumes in this set can be considered to be a deformed version of any of the other volumes. Since the deformations are caused by the volunteer's real motion, the resulting set of volumes incorporates realistic patient motion.

Since the actual deformations are not known, the observation data, such as the position of external markers on the patient's surface, will have to be captured at the same time as the volumetric images from which the phantoms will be generated. These data can then be associated with the phantom corresponding to that specific body configuration.

The voxelized numerical phantoms with the corresponding observation data are enough to generate all the data that the motion correction reconstruction algorithm needs to generate its results. Specifically, the projection model can be used in conjunction with the phantoms to generate the projection data, and the observations can be fed in as is. This is the version of Dynamic Anthropomorphic Numerical Phantoms we begin to implement in this work.

4.1.2 Set of Deformations

In addition to just segmenting the organ boundaries, one can also attempt to determine voxel correspondences between the pairs of different volumes in the set using non-rigid registration (32) (42). This data could then be used to refine the organ boundary segmentation by incorporating data from the other volumes (43) as well as provide a way to generate the motion observation data directly, without having to capture it with the volumetric images.

Each pair of volumetric images, $A(\vec{x})$ and $B(\vec{x})$, of the same volunteer provides a view of the same organs and slightly altered configurations. Non-rigid registration provides a deformation $\overrightarrow{W}_B^A(\vec{x})$ mapping voxels in volume A onto voxels in volume B, while segmentation extracts the organ boundaries $S_A(\vec{x})$ and $S_B(\vec{x})$ from each volume individually. Ideally, $S_B(\vec{x}) = S_A\left(\overrightarrow{W}_B^A(\vec{x})\right)$ (applying the deformation computed during the registration stage to the organ boundaries in one volume should exactly map them onto the organ boundaries in the registered volume). If we assume that the segmentations are exactly accurate, then we can use the difference between these two organ boundaries to refine the registration and make it more accurate. On the other hand, if we assume that the registration is perfect, then we can use the misalignment of the boundaries to refine segmentation. In reality, neither the registration nor the segmentation will be exactly perfect. Optimizing both at once can provide more data than doing either one alone and in theory lead to both a better registration and segmentation (43).

When both accurate segmentations and registrations mapping the different volumes into each other are accessible, then one can, in addition to assessing reconstruction accuracy, also measure the accuracy of the motion estimates generated by the motion correction algorithm.

4.1.3 Generate NCAT Phantoms

A voxelized representation of an organ boundary can be fitted with a NURBS mesh. Since NURBS are parameterized by the locations of their tie points, the deformation functions computed in the registration step can be applied directly to generate deformed models in different configurations.

5 Identifying Organ Boundaries

This section focuses on the acquisition and segmentation of realistic organ data that will be used during the proposed testing procedure, and the results that we obtained using software we developed.

We segment the body based on the properties of the organs of interest. In these images, the blood in the heart is bright, the air in the lungs is black, and muscle is gray. The ribs are the dark region contained between the lungs and the soft tissue. After the image is denoised using anisotropic diffusion (44), sets of connected components are extracted using thresholding. These represent the initial, rough boundaries for the organs of interest. They are subsequently refined using morphology (45) and active contours (46). This leads to the “threshold and refine” approach that we follow to extract the organs on a slice-by-slice basis from the MRI volumes.

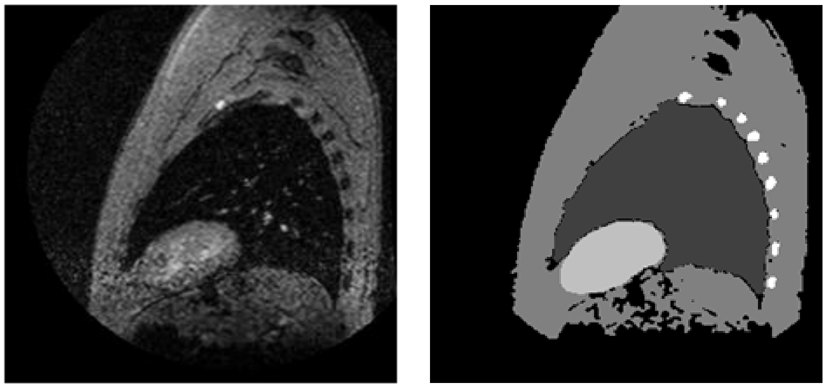


Figure 11 Input Slice (Left) and its segmented organs (Right)

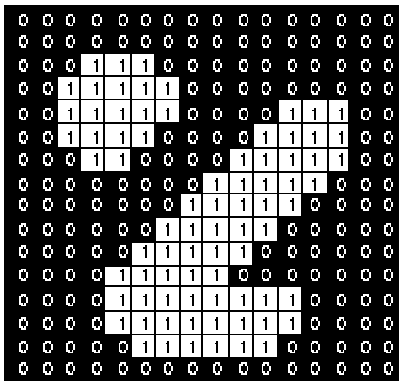
5.1 Segmentation Techniques

Morphological dilation thickens a boundary while erosion cuts away at it from the borders. Active contours refine a boundary by applying three forces over a specified number of iterations. The balloon force either pushes the boundary outward when it is positive, or shrinks it inward when it is negative. The advection slows the boundary’s propagation around image edges, and the curvature force acts to smooth out the boundary.

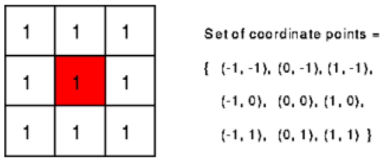
5.1.1 Morphology

Morphological operations are performed by translating a structuring element over the image region and performing a set operation (45). One can represent regions using binary images with pixel values of {0, 1}; A pixel whose value is 1 is considered part of the region, and a 0 pixel represents the background. Morphology uses two binary images – the image on which the morphological operation will be performed, and the structuring element that is used to perform it.

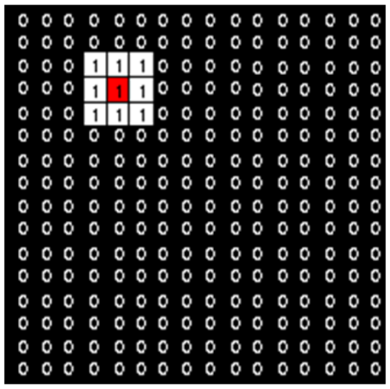
A structuring element is a binary image that has an origin. If we let B be the set of on pixels in a structuring element, then we can define an image B_z that is the same structuring element translated so that its center is at pixel z. This gives us the foundation to define morphological operations.



Binary Image A



Structuring Element B with the origin (red pixel) at center (Above). B_z (Right) is an image the size of A (Left) with structuring element B centered at the pixel z. Here, $z = (3,4)$.



$B_{(3,4)}$

Figure 12 Binary Image (A) and Structuring Element (B) from (47)

An erosion of region A by structuring element B is the set of all z's where B_z is a subset of A. The set of all z's can itself be represented as a binary image. Erosion can be thought of as carving away the borders of region A using B. It can be used to separate regions that are connected by a thin connection.

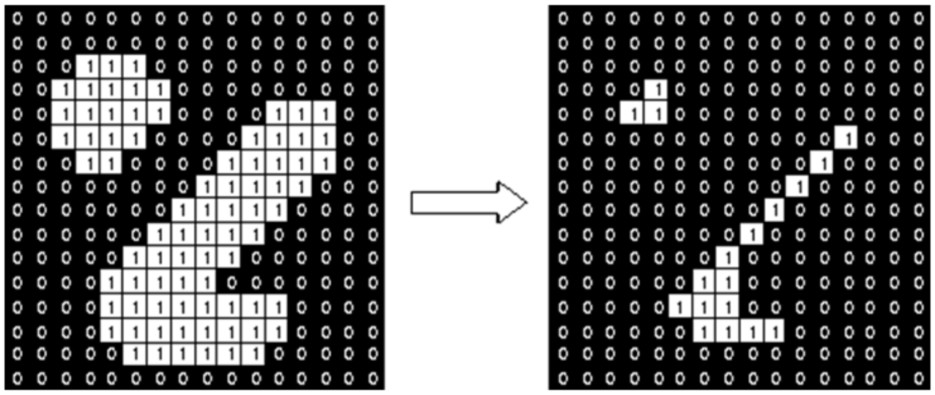


Figure 13 Erosion of (A) by (B) from (47)

A dilation of region A by structuring element B is the set of all z's where B_z intersects A. It can be thought of as enlarging region A using B and can be used to connect regions together.

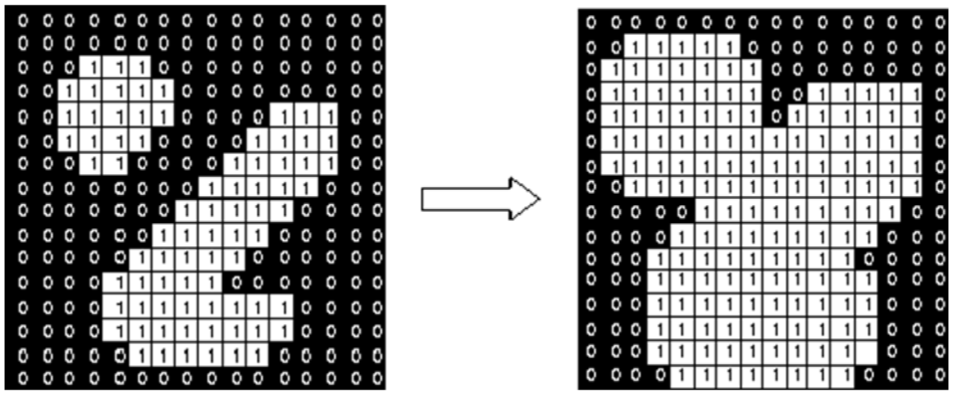


Figure 14 Dilation of (A) by (B) from (47)

In our work, we used square structuring elements with the origin at the center. This produces very rough boundaries. However, this was not an issue as they always would be refined by active contours which produce smooth boundaries.

5.1.2 Active Contours using Level Sets

Active Contours are a general method for iteratively improving a boundary estimate (46) (48). In the Level Sets formulation (46), the boundaries are implicitly defined inside of a scalar field the size of the image being segmented. The boundaries are defined as the set of points where this field is equal to zero with negative regions correspond to region interiors, and positive regions to the exteriors. Even though only the region of the boundary is of interest, the entire field gets updated every iteration.

The initial estimate of the boundary should be close to object boundary, though the forces that act on it during the iterations will move it to new locations, with the goal of providing a better final segmentation. The update step uses forces derived from the image being segmented to guide the boundaries to the object(s) of interest.

Since the boundaries are expressed implicitly as the set of zero crossings in the scalar field, topological changes are also naturally and implicitly handled. Thus, boundaries can merge or split as necessary.

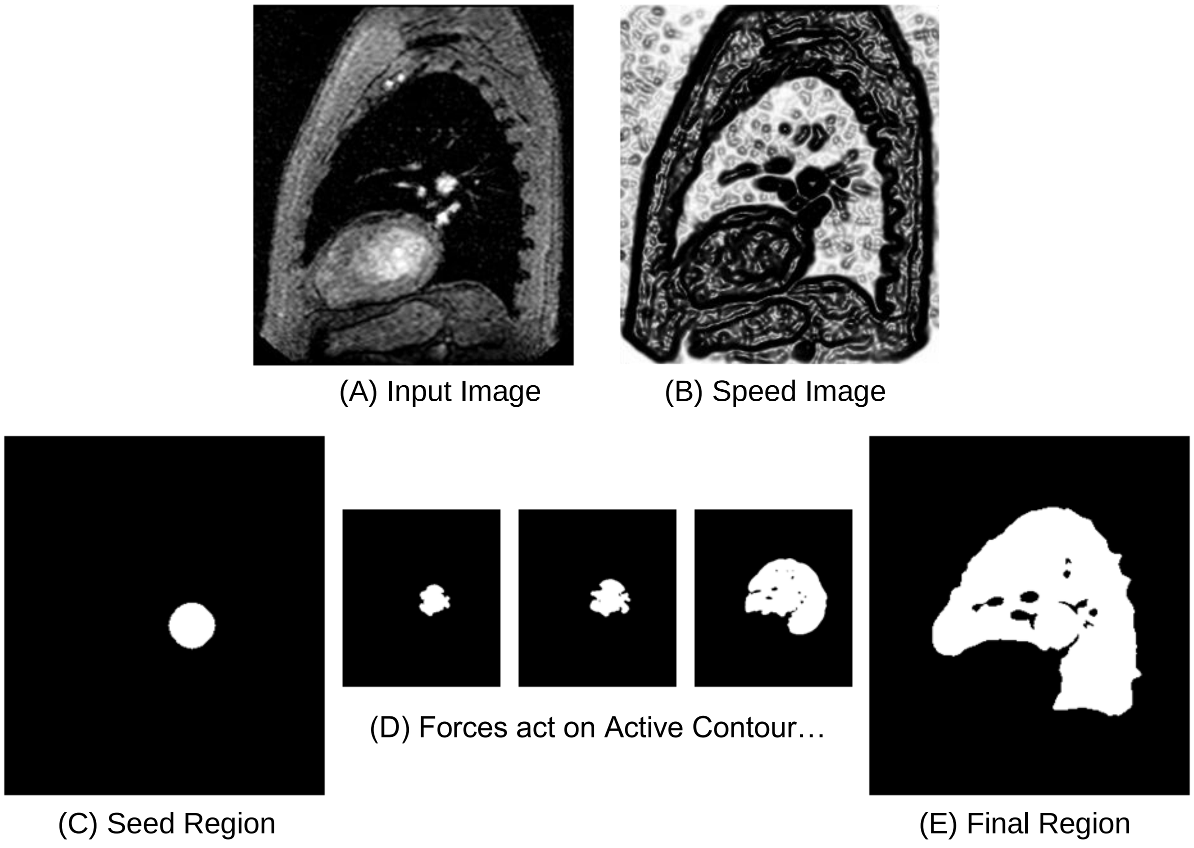


Figure 15 Active Contours

Forces

The advection A force is also called the Speed Image (Figure 15 (B)). Its role is to stop or slow down boundary propagation at the borders of objects (edges) and to move the boundary more quickly in smooth regions (which are not of interest). Edges should have speed values close to 0, stopping the boundary, while the smooth regions that are of no interest should have speed values close to 1. One can generate the speed image by apply sigmoid to the gradient image. Using a sigmoid also allows one to choose control the contrast between regions of varying gradient intensities.

$$sigmoid(I) = \frac{1}{1 + e^{-\left(\frac{I-\beta}{\alpha}\right)}}$$

Equation 5 Sigmoid

Where I is the pixel intensity in the input image and α and β are parameters the user gets to specify. α effects how sharply the image intensities are condensed, while β controls the value around which the function is centered.

The propagation (balloon) force P expands or shrinks the boundary. It is constantly acting, but can be spatially varying. For our purposes, P is always a positive constant when the boundary should inflate, and a negative constant when it should contract.

The curvature Z force controls how rapidly the boundary can vary. It helps keep the boundary together and smooth even in noisy data. It can be spatially varying, but is usually set to a non-negative constant.

Update Equation

The forces are applied in the direction of the gradient. This amounts to propagating the contour in the normal direction. Since forces applied in the direction of the contour on the other parameterized curve but do not change the shape, all such forces can be discarded in this implicit, nonparametric, framework (46). The formula below describes how the scalar field Φ at time i is updated to produce the field for iteration $i + 1$.

$$\Phi_{i+1} = \Phi_i - F|\nabla\Phi_i|$$

Equation 6 Level Set Active Contours Update Step

In this equation, Φ is the scalar field containing the implicit boundary. F is the combined force applied to the boundary – it is a linear combination of the advection, propagation, and curvature forces.

$$F = \alpha\frac{\vec{A}(\vec{x}) \cdot \nabla\Phi}{|\nabla\Phi|} + \beta P(\vec{x}) - \gamma Z(\vec{x})\kappa(\vec{x})$$

Equation 7 Active Contour Forces, General Form

Where κ is the curvature of the level set field. Since the propagation and curvature forces are constant, this equation simplifies to:

$$F = \alpha \frac{\vec{A}(\vec{x}) \cdot \nabla\Phi}{|\nabla\Phi|} - z\kappa(\vec{x}, t) + p$$

Equation 8 Active Contours Forces with Curvature and Propagation Forces Constant

Where a lower case z and p have been used to signify that these forces are kept constant.

5.2 Segmentation Approach

In this section, we describe the specific procedures used to extract each of the organs.

5.2.1 Body

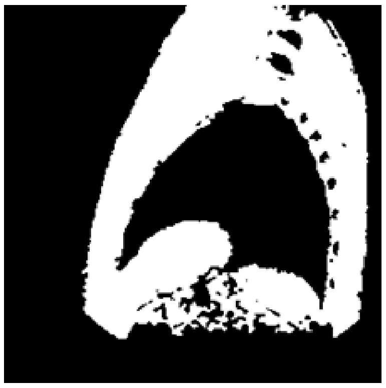


Figure 16 Body Region

We extract the body region using thresholding and connected components. The threshold is chosen such that it includes as much as possible of the gray soft tissue in the image, but high enough that the noise in the background does not significantly distort the body contour. The largest connected component is extracted and used to represent the body. In later steps, the heart and ribs will be extracted from this region as well.

5.2.2 Lung

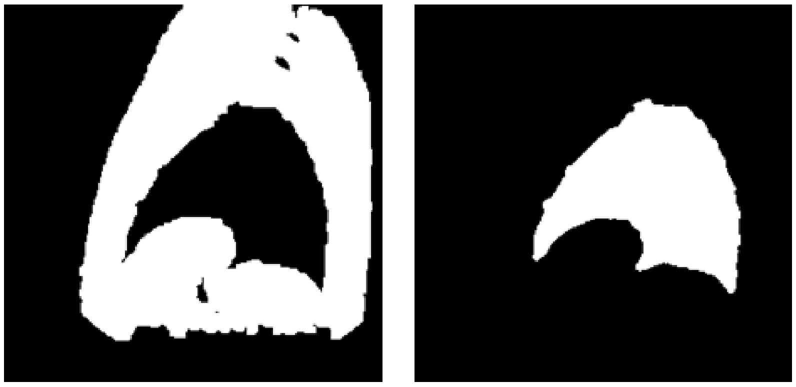


Figure 17 Body Dilation (Left) and Lung Final Region (Right)

The lung is the black region inside the body. Frequently, this region is connected to the dark region outside of the body due to the dark air in the stomach, the MRI's restricted field of view, and noise present in the image. We close these gaps in the body by dilating it with a structuring element whose radius is selected to be just large enough to close these gaps. This also has the effect of eroding the dark region inside the body corresponding to the lungs. We blow that region back up using active contours with a positive propagation force. The advection and curvature forces are chosen such that the boundary doesn't leak too much into the ribs, but still gets into the sharp corners above the diaphragm and near the heart.

5.2.3 Ribs and Spine

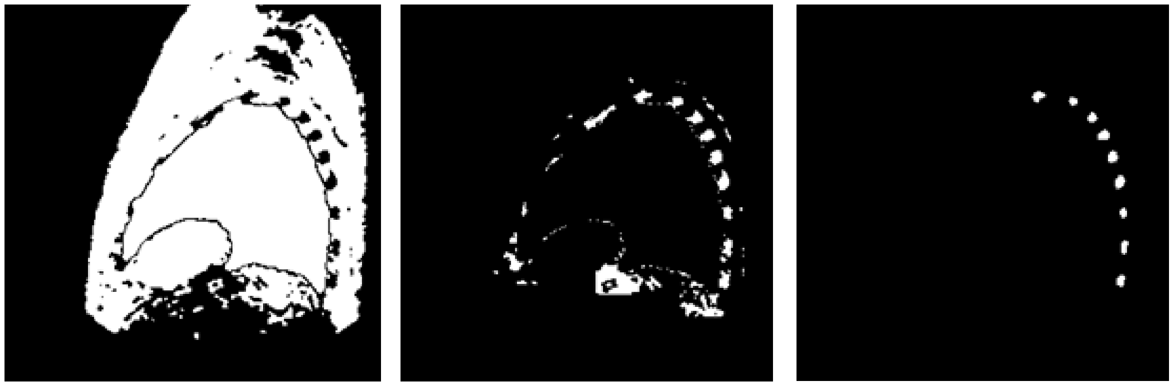


Figure 18 Dilated Body plus Lung (Left), Initial Ribs (Center), Final Ribs (Right)

The ribs are the dark regions in the body just outside of the lungs. We obtain an initial region for the ribs by extracting the dark region between the lungs and the body within a user specified number of pixels of the lung's border. A different version of the body is used in this case, thresholded at a higher value in order to expose more of the dark ribs.

This initial region is refined using active contours to smooth out the boundaries. The final ribs are extracted from all the remaining connected components by specifying a smallest and largest size for the ribs and then manually removing spurious features that fall within that range.

Parts of the spine are extracted using the same approach, relying solely on the proximity to the lungs and the darkness of the vertebrae in the MRI images.

5.2.4 Heart

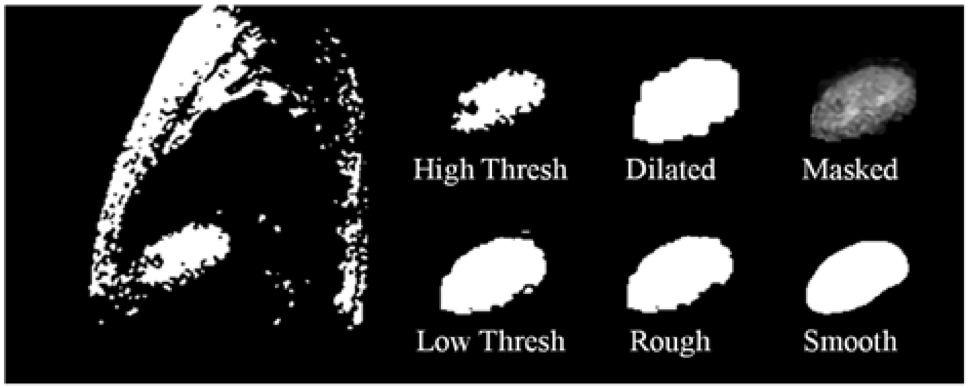


Figure 19 Heart Segmentation

The blood inside the heart is brighter than the rest of the soft tissue around it. This allows us to use a high threshold value to separate the blood from the rest of the body. We then dilate that region and mask the original input image to extract the tissue around the blood. This allows for a lower threshold to be used, which captures more of the soft tissue of the heart. The largest component is extracted, but the boundary is still rough, so we smooth it out using active contours.

5.3 Results

We tested our segmentation approach on data acquired in an earlier study aimed at assessing the effects of patient motion on the location of the heart (49). Volunteers held their breath and two cardiac triggered MRI acquisitions were performed one after another. This assured that the heart was at the same location in the cardiac cycle in each slice and the lung volume was the same in both poses. The MRI volumes spanned the part of the thorax containing the heart. Each volume was made up of 20 sagittal slices, each 4mm thick with a 2mm gap between slices. Each slice image is 256x256 pixels, with 1.367mm^2 pixels. The segmentation results are provided at in section 5.3.1. Each segmentation took approximately 3 hours to perform on a Dell XPS M1330 Laptop, most of the time coming from manual manipulation of parameters.

This approach does fairly well on slices that only contain lung, heart, body, and ribs (1-9 in the volume A and 1-8 in B). In slices where the larger vasculature appears, it becomes difficult to separate the heart from the aorta and the vena cava, which should be segmented as separate regions from the heart (11-13 in the first volume and 10-13 in the second).

Extracting the spine using this method is less reliable. It is extracted using the same approach as for the ribs, looking for the "dark region inside the body close to the lungs." Thus, if there are no lung or trachea regions near the spine in a specific slice, the current approach will not find the spine regions (13-17 in volume, finding no spine at all, and 13-17 in B, with some partial segmentation). In slices 18-20, the air in the trachea gets classified as lung and acts as the starting region for the spine segmentation.

5.3.1 Two Segmented MRI Volumes

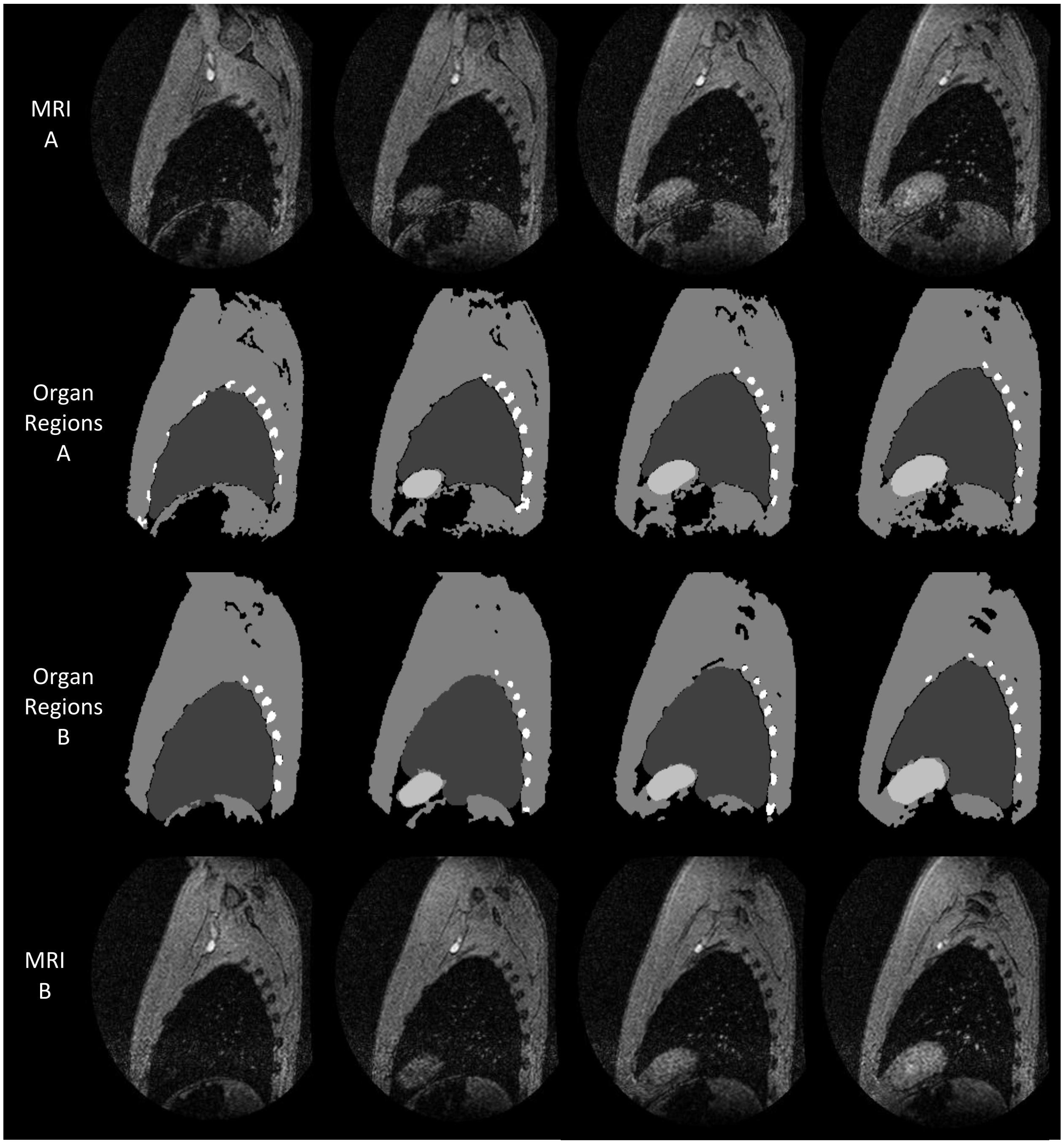


Figure 20 Slices 1-4

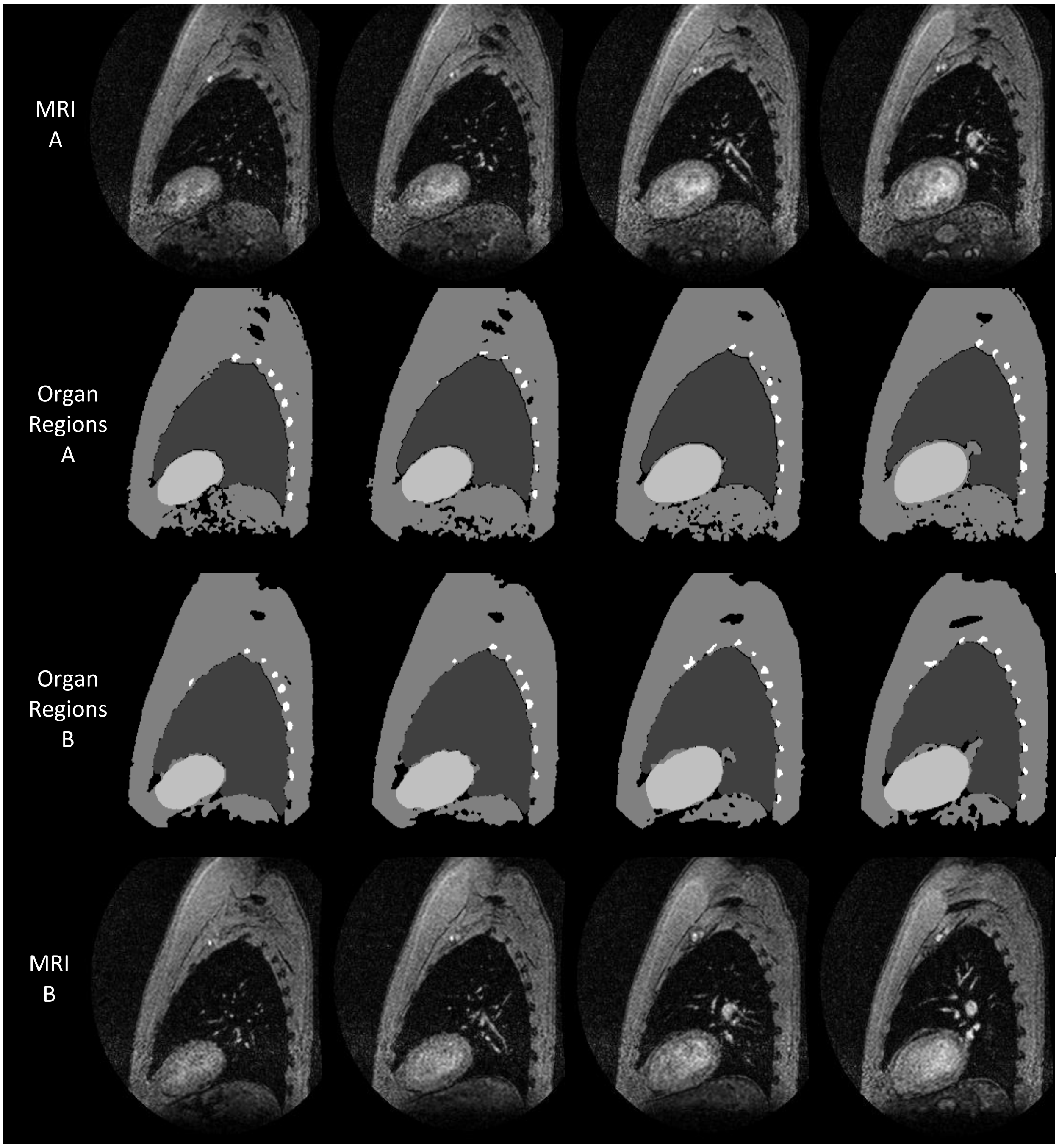


Figure 21 Slices 5–8

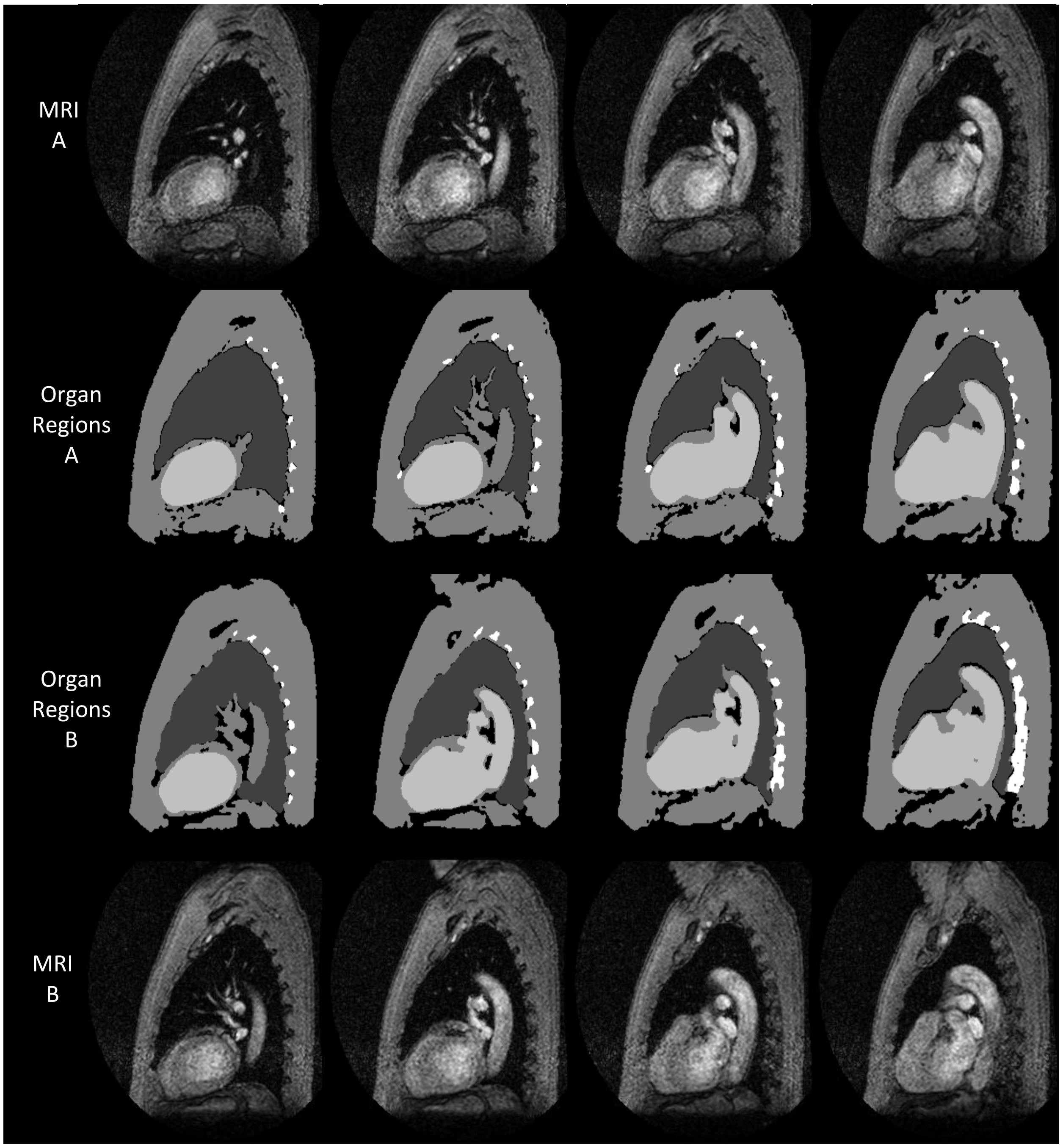


Figure 22 Slices 9-12

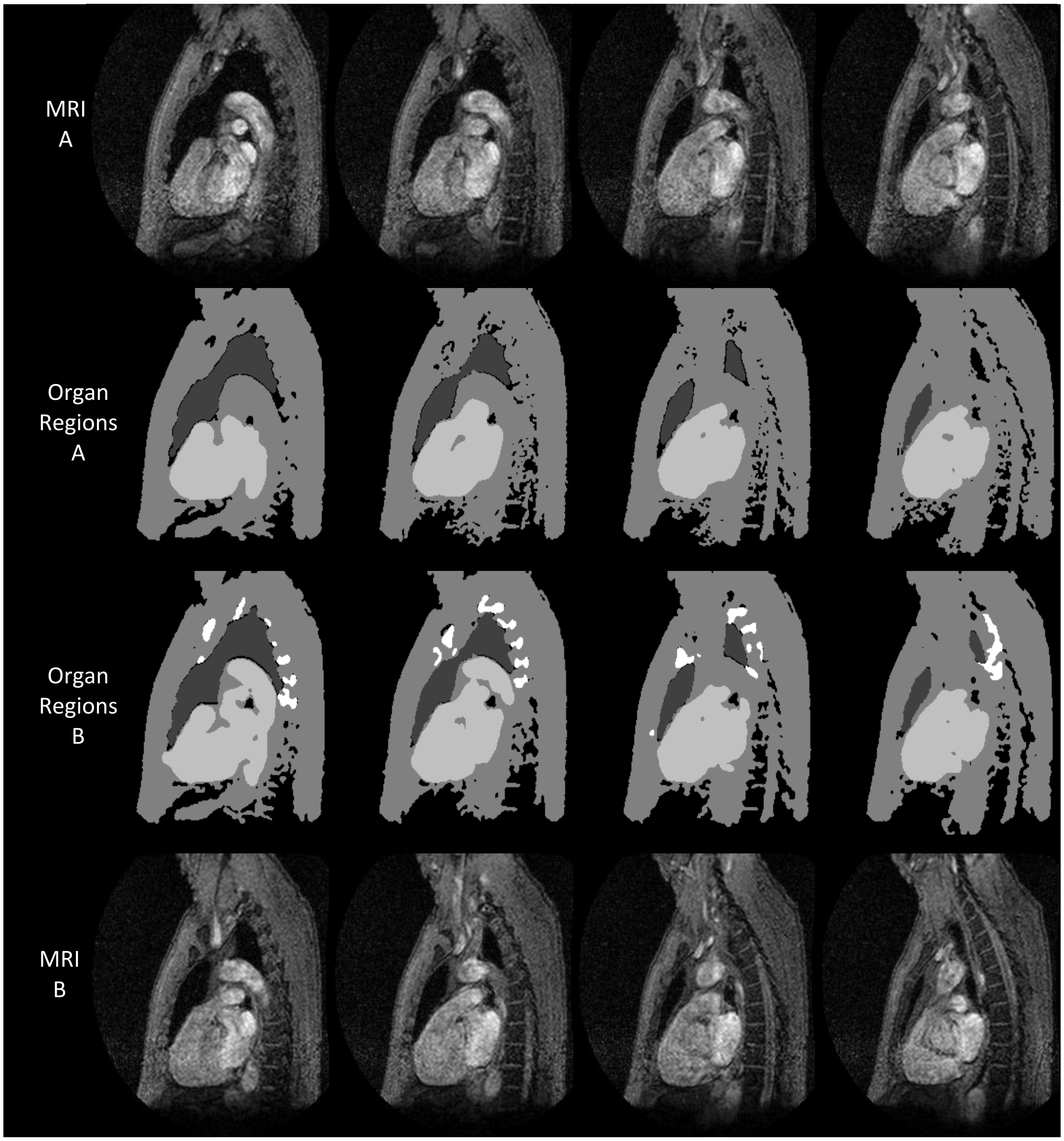


Figure 23 Slices 13-16

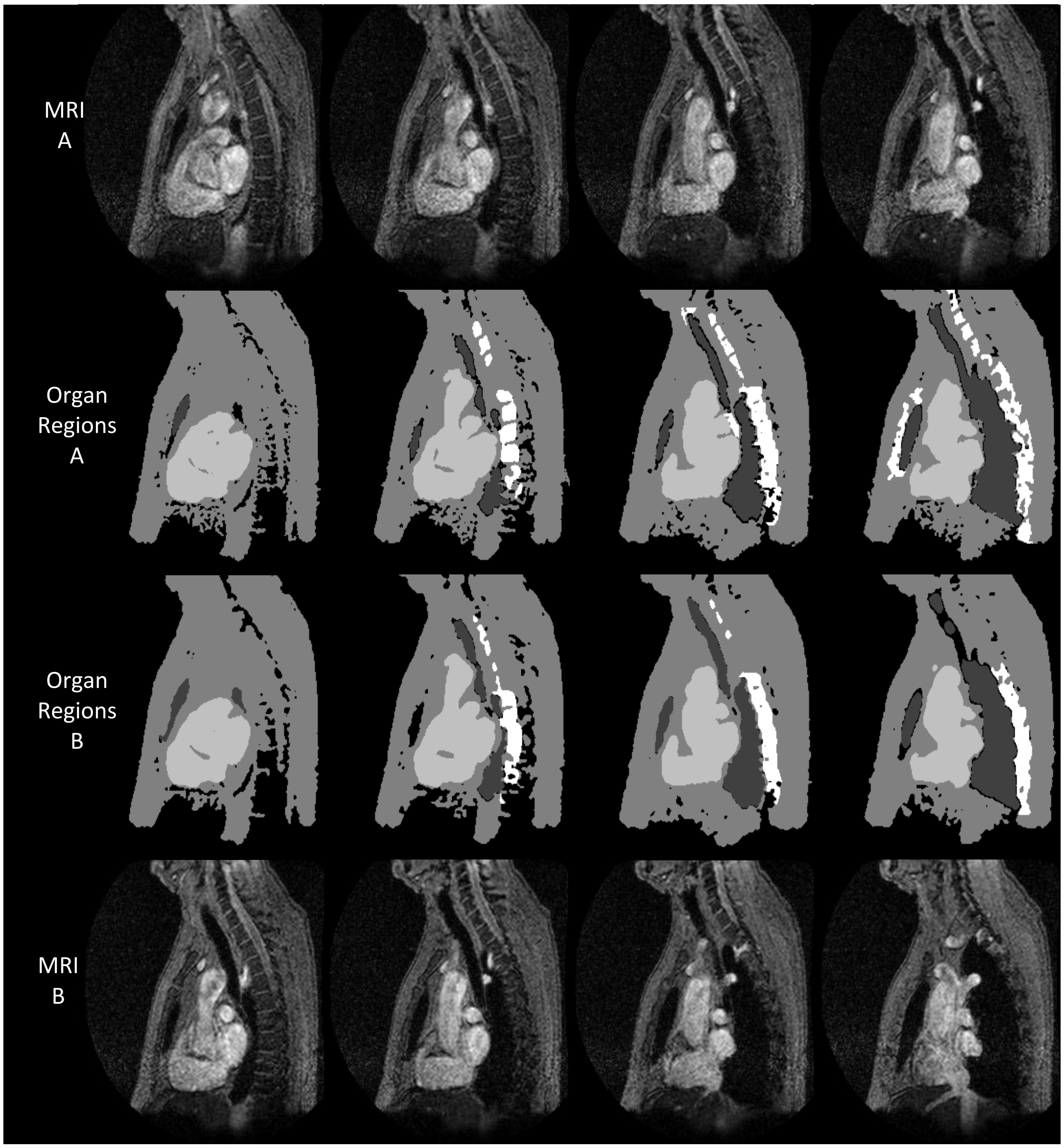
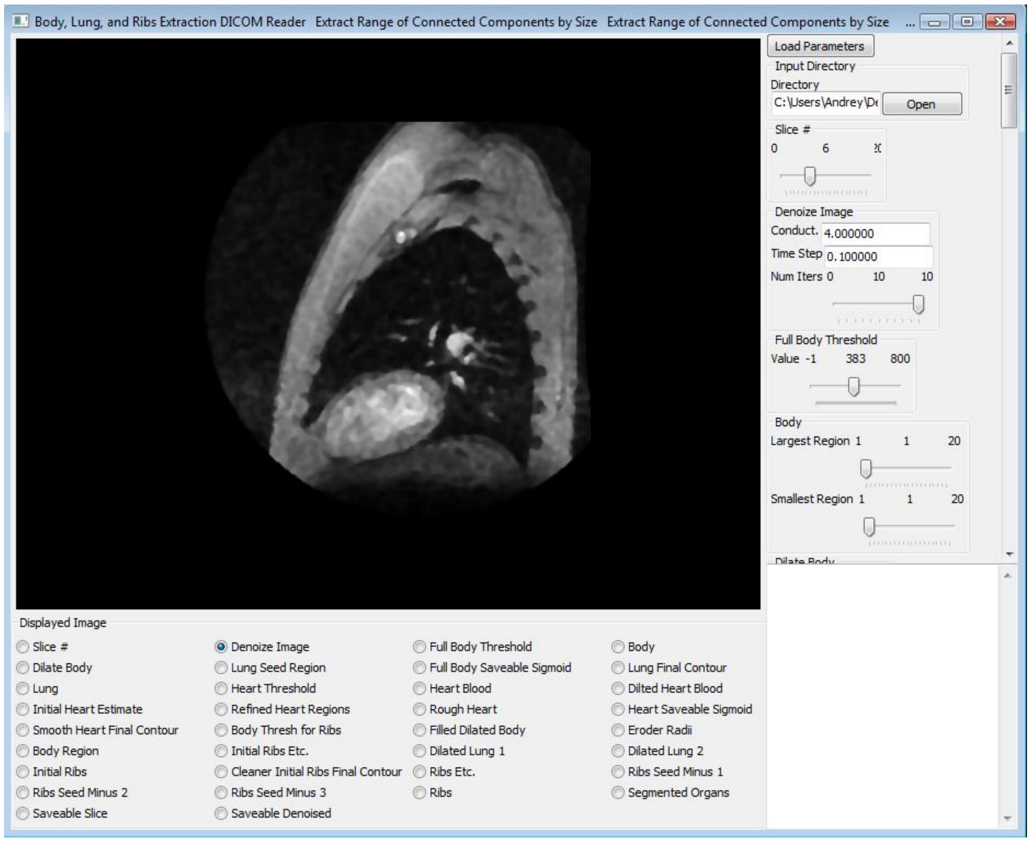


Figure 24 Slices 17-20

6 Implementation

In order to extract the organ boundaries, we needed software that would allow us to ingest volumetric images produced by the MRI scans and perform the segmentation steps described above. We decided to implement the software ourselves using available open-source software libraries.



Directory C:\Users\Andrey\De
Slice # 0 6
Conduct. 4.000000
Time Step 0.100000
Num Iters 0 10 10
Value -1 383 800
Largest Region 1 1 20
Smallest Region 1 1 20
Displayed Image: [✓ Denoize Image]

Figure 25 Interactive Image Processing GUI Application

Figure 25 shows the final application developed for performing the image processing required in this project. The main portion of the program window is taken up by the visualization component and the rest of the program window consist of the interactive GUI widgets for parameter specification. Widgets are grouped according to which filter they provide the interface to. In the rest of this section, we describe the engineering efforts that went into developing the software framework that was used to create application.

6.1 Requirements

An interactive image processing and segmentation program needs at least three components:

- an image processing library that implements the filters we intend to use,
- an interactive image display that would could visualize the image processing effects, and
- a graphical user interface by which to specify filter parameter values.

The image processing library needs to support multiple step pipelines (e.g. denoise an image, threshold it, perform some morphological operations, then run active contours). It needs to provide a way to save and load the values of the parameters specified to the filters in order to be able to reuse the hand tuned values in later segmentations. Finally, in addition to being able to visualize intermediate outputs, the application should also be able to save them to disk for offline use in presentations and reports.

6.1.1 Choice of Libraries and Programming Language

The choice of programming language to use for this project was driven by the existence of publically available libraries that met the above requirements. File input/output and graphical user interface libraries are available in many general purpose programming languages and did not provide any strong constraints for programming language selection. For visualization, we found an excellent open-source library - The Visualization Toolkit (VTK) (53) - implemented in C++ with bindings in Python and Java. The same company (54) that produced VTK also provides an open-source image processing library - Insight into Segmentation and Registration Toolkit (ITK) (55). It implements many current segmentation and registration techniques and is currently only available in C++. This decided the programming language to use for this project. Finally, we chose to use wxWidgets (56) for the graphical user interface component because there was already an interface developed for displaying a VTK window in wxWidgets applications called wxVTK (57).

6.2 Overview of Filter Pipelines (ITK and VTK)

The fundamental building blocks of ITK and VTK applications are filters. A transform filter takes in an image, does some processing, and produces an image. A source filter produces an image but takes no inputs, and a sink filter takes in an image but produces an output. Source filters are used to implement things such as loading an image from disk, sink filters are used for saving files and displaying output to the screen, and transform filters perform the image processing operations.

ITK is purely focused on image processing, so it does not have any display filters. VTK is all about visualization and provides many different ways of visualizing and processing data for visualization. There exists a Filter that converts ITK image types to a format that VTK can use, allowing us to build image processing pipelines using ITK and display them using VTK visualizers.

In addition to taking images as inputs, filters usually also need to have different parameter values specified in order to perform the appropriate operation desired by the user. ITK and VTK provide setter and getter methods for these properties, but leave it up to the programmer to specify them at runtime. For interactive applications, these parameter values come from GUI widgets, so that moving a slider that provides a parameter to a transform filter would modify the visualization output in real time.

6.3 First Pass

The first application that we developed using these libraries was a tool for active contours segmentation of 2D images. It exercised all the requirements specified above and provided a straightforward coding framework for writing interactive image processing applications.

The framework consisted of three distinct subsystems - the image processing subsystem, the display, and the parameter specification GUI. Each of these subsystems was encapsulated in its own class, using getter and setter methods to pass parameter values to the image processing filters and to retrieve filter outputs for display. Since wxWidgets is an event driven GUI library, each text box and slider that was used to specify parameter values needed its own callback function which would in turn call setter method on the image processing subsystem.

This approach generates a lot of code, most of which is associated with filter parameter specification. But that is the most fundamental and most frequently used component of an interactive image processing application. There needed to be a better way for application developers to write these sorts of programs that would be less distracting and time-consuming, allowing one to focus on the high-level tasks of putting filters together rather than on the low-level tasks of wiring the callbacks for the input widgets.

6.4 High Level Interactive GUI Language (HLING)

Rather than discard a working software development framework, we developed the High Level Interactive GUI Language (HLING) that allows one to express the high level image processing ideas succinctly. It then gets compiled down into the corresponding C++ implementation described above. Most significantly, it removes the details of event driven GUI programming, providing a more straightforward way to write image processing filters with interactive filter parameter inputs.

The language itself was written using PLT Scheme (58) using the Programming Language: Application and Interpretation module. It was not intended to alleviate the programmer from knowing C++, but is rather a tool intended to simplify development for this specific type of C++ coding framework.

6.4.1 Language Structure

Programs written in HLING are divided into three high-level sections: filter type definitions, filter object definitions, and state transition specification. In the filter type definition section, the programmer first defines and names the pixel types (integer, short, float, double, etc.) that will be used to represent data in the images. Image types are defined using these pixel types and the number of dimensions of the image (two or three in this case). Filter types are split into three categories: source, transform, and sink and are parameterized on the input and output image types.

The filter object section is where the interactive portion of the application is defined. A filter object is an instance of the specified filter type defined above. It provides a short description for the filter, which will be displayed in the GUI, as well as the specifications for all of the interactive parameters.

The parameters are integer or floating point numbers entered through a text box or a slider. The language compiler takes care of creating the callback functions and creating the widgets. If there is a setter function whose value is directly specified by the value of the widget, the application developer doesn't have to write any C++ code (e.g. an upper threshold on a threshold filter). If the value needs to be processed before being passed on to the setter, the application developer can also write C++ code that calls the filter's setters after performing some computations (e.g. calling a function to create a morphological structuring element before passing it to the dilation filter).

The second part of the filter object section describes how filter objects connect to one another. It uses a straight forward functional syntax to specify which filters are inputs to what other filters in the graph. It also provides syntax for multiple outputs.

The state transition section specifies which filters are active at what points during the program's execution. It is intended to prevent the system from trying to perform computation when no image data are loaded. State transitions are triggered by the user opening the necessary input files required by the specific filters.

6.4.2 Language Translations

HLING is implemented in three layers, requiring two translations. The first two layers - `user-spec` and `application` – each have a syntax tree defined in Scheme (definitions included in the section Appendix – HLING). The final layer is C++ code, which gets generated directly from the application layer. The `user-spec` provides the highest level of the abstraction by providing shorthands for specifying different parts of the interactive application. The `application` level expands all of these shorthands into a common syntax structure that can be used directly to generate C++ code.

Translating a user-spec to an application

The most notable shorthands provided at the `user-spec` level were those for input/output types of the filters and shorthand for specifying whether to permit displaying and saving of a filter's image output.

For virtually all of the image processing filters that we used, the input image data was specified by calling SetInput() on the filter, and the output image was gotten by calling the GetOutput() method. Thus, in order to express that "filterA acts on filterB's output", one would have to write

```
filterB->SetInput(filterA->GetOutput())
```

functional notation already provides the semantics, but is much more compact and familiar.

```
filterB(filterA)
```

thus, we decided to provide the shorthand of not having to explicitly specify the names of the output and input functions, except for when the default assumptions of SetInput and GetOutput were inaccurate (such as when specifying two different images as the input of the active contours filter - one being the feature image and the other the initial level set).

We also wanted to be able to save and display the outputs of different filters. This required creating ITK sink filters for writing the image to disk and converting it to the format that VTK could handle for display. But these filters would all be very similar for all the different filters whose outputs we wanted to display or save, the variation being in the different image types that they would produce. There is no reason that somebody writing an image processing application should have to worry about this level of detail. Thus, the user was able to specify whether they wanted to be able to display the image, save the image, both, or neither. The application layer took care of performing the type inference and generating the structured representation of the image writer and display filters.

As a benefit of having the `user-spec` syntax structure explicitly at our disposal, merging applications was really easy. All that had to be done was just to copy all of the types, filters objects, state transitions, etc. into the same corresponding list to generate a new user-spec that now contained two different subgraphs. All that the programmer had to do was add another connection from the output of one of the filters in one graph to the input of the filters in the other graph. The two pipelines were linked without having to re-write the code of either one. This was especially useful for filters and subgraphs that were commonly reused, including thresholding, active contours, and connected component extraction.

Translating an application to C++

After the `user-spec` is expanded to an `application`, the `application` is compiled to C++ code. This is where the declarative syntax of the `application` is converted to the imperative syntax of using callbacks in C++. Different parts of the `application` syntax tree get extracted in order generate different parts of the source code. For example, in order to generate all the callback functions for the input widgets, the syntax tree was mapped and filtered to extract the filter names, input names, and other necessary data. These data were then used to generate callback functions with appropriate type signatures that would call the proper setter functions on the filters. Similar code had to be present in the instantiation of the GUI widgets in the GUI subsystem's constructor, and thus used a similar subset of the syntax tree to generate the C++ code.

A specifically intricate example of converting the declarative syntax to the callback-based implementation is the case when a filter's setter method needed two or more input values. In this case, the values of two widgets have to be accessed and passed whenever either one of them was modified. A straightforward way to implement this is to create one function that takes no parameters and reads the values of all the widgets involved in this parameter update and call the filter's setter method. Each widget's callback would just call this thunk whenever its value updated. This is exactly what happens when the C++ code is generated. On the other hand, when only one value needs to get passed to the filter, the most straightforward thing to do is to just forward the value passed to the callback function without reading an specific widget's value. Since this single parameter specification is more common, coming up with the architecture for the two or more input case took some extra care.

6.4.3 Feature Creep

As we began to write applications using HLING, a number of additional features that were not anticipated at the onset of the language implementation were added as their need became apparent. A large part of these additions were related to dynamically updating slider minimum and maximum values. Another was the need to specify non-image types as parameters to filters as well as import helper C++ files. This section describes the details of these additions.

Minimum and Maximum from Widgets

The first addition that became necessary was to provide a way by which to assure that a parameter whose value was semantically larger than another one, such as the maximum and minimum values for a threshold, could never be specified erroneously. All that the program had to do was to update the minimum value of the maximum threshold whenever the minimum threshold is moved, and vice versa. Since the user can only move one slider at a time, no race conditions were introduced.

Minimum and Maximum from Filters

Creating a slider for choosing a slice in a dynamically loaded MRI file, or selecting a specific connected component from an image required a value from the image processing layer (number of MRI slices or number of connected components in these cases) to be propagated to the GUI layer. In the MRI slices case, the operation is started by the user opening a file from the GUI, providing a specific event after which to sample the desired filter parameter. The number of connected components in the thresholded image, however, could change due to events happening in other filters upstream from the connected component extractor in the graph.

The correct way to think about the data flow in this situation is that slider's minimum or maximum value comes from the filter. This is completely analogous to the filter’s parameter value coming from a GUI widget. Thus, they received the same syntax: when specifying a slider, the programmer may specify a constant value for the minimum or maximum, or they may specify that it comes directly from a getter method on the associated filter. They may also specify their own C++ function that takes zero formal parameters and returns the same type as the slider takes (integer or float). These getter functions get called every time **any** widget’s callback gets called – since updating another filter may have effects on the desired filter’s output values.

Non-Image-Type Filter Type Parameters

Another feature that had to be added was being able to specify non-image types as parameters to the filter types. Specifically, the morphological operators required that the type of the structuring element (the Neighborhood type in ITK, which is not an Image type) be specified as well as the input and output image types. We created a set of helper C++ function for generating rectangular neighborhoods to be used as structuring elements for erosion and dilation and needed to be able to include the header file in which we defined them. This did not fit nicely into the framework, which assumed that the only included header files were those for image and filter types. We resolved this by providing a “helper-type” section that could be used specifically to specify additional non-filter non-image include files.

7 Interactive Image Processing Application Requirements

In this section we discuss the lessons learned from the software development process of this project by laying out a set of requirements for interactive image processing applications written in C++ using ITK, VTK, and a GUI library.

We implemented HLING in order to provide a more intuitive way to write the GUI component of the applications. It wasn't intended to completely abstract the C++ implementation of the applications. As such, someone developing an interactive image processing application with HLING needed to both understand its syntax as well as be proficient in C++. Since application developers should already be familiar with C++, making this GUI writing layer in C++ would alleviate application developers from having to learn, use, and further develop HLING.

7.1 High-Level Requirements

There are several guiding principles that should be followed when developing the interactive GUI framework. Most of these points find their roots in declarative programming paradigms and the programming languages that implement them (58) (61) (62) (63), though one need not be familiar with these programming languages to understand these guiding principles. The aim is to simplify how programs are written and debugged while exposing all of the available tools and power of ITK to the programmer.

1.) *Programs should be written as declaratively as possible*

    It is natural to think of image processing applications as "filters chained together into pipelines with parameter values coming in through a GUI." ITK already provides a declarative syntax for specifying the image processing pipelines. It was the wxWidgets GUI implementation that used imperative C++, which significantly increased the amount of code necessary to express the "values coming in through a GUI" component.

    The approach taken in this work to resolve this problem was to write a high-level, declarative language that the programmer could use to specify the GUI which would then get compiled down to the spaghetti code of a callback-based program. Alternatives exist that use straight C++, though, and should be preferred. The QT GUI library (59), for example, uses slots to provide a declarative syntax for specifying widget value propagation through the program and should be considered for the next iteration.

2.) *The programmer should be able to use all available components of ITK*

    In writing the high-level programming language used in this iteration of the software development we had to add new structures to the language in order to allow access to the existing functionality in ITK. This overhead could be avoided by using C++ directly.

    There are components that will require non-image inputs and produce non-image outputs, the neighborhood structure, for example; image statistics are also a useful feature that can be used in the future. Since these parts of ITK are already written in C++, no additional program structure needs to be added to use it.

3.) *The programmer usually shouldn't have to think about pixel and image types when writing pipelines*

Since ITK relies heavily on C++ templates, it is necessary to specify all of the image and pixel types for every filter when composing ITK filter graphs at compile-time. Many times, though, default types can be safely assumed and used for different types of filters. Similarly, if two filters are linked, the input type of the receiving filter will have to match the output type of the producing filter. Using a combination of reasonable default types combined with a static type inference mechanism can make the graph building process a lot less verbose. At the same time, the default type values can be superseded to fit the program developer's needs.

4.) *Filter pipelines and GUIs should be easily reusable*

Putting together different pipelines and their associated GUIs should be as straight forward as it was in HLING. This was not possible to do using the C++ coding framework developed in this work, as each widget of every filter needed its own global callback function. HLING enabled this functionality by concretely representing the program structure at a higher level, where they could be easily combined. It then generated the necessary C++ code that implemented them. Future implementations may be able to forego the need to have a callback function for every single widget in the application, which would enable C++ programs to be combined at a high level as well.

7.2 Application Layout

The application layout in the current implementation was very intuitive (Figure 25). We suggest that it be reused, with some minor additions, for the next iteration. The user interface could also provide a way to collapse the widget group associated with a specific filter. Since the filters are organized into pipelines, it may also be useful to be able to collapse the set of filters making up a pipeline. This would be especially useful if the application were made up of a number of independent pipelines whose outputs all fed into a final filter or pipeline that combined the results.

7.3 Hypothetical Source Code

The hypothetical source code below should generate a full interactive image processing application that implements the basic segmentation method used in this work (threshold then refine using active contours). The programmer need only specify the types of the filters to be used in the application using the filter wrappers, which encapsulate the interactive GUI and ITK filter graph building tasks. Thus, the most attention should be placed on coming up with a good way to write these filter wrapper classes.

```
// Filter Wrapper classes.  They implement the GUI components
// for individual ITK filter types.
#include <ImageReader.hpp>
#include <Denoiser.hpp>
#include <Thresholder.hpp>
#include <Gradient.hpp>
#include <Sigmoid.hpp>
#include <ActiveContours.hpp>
```

```
#include <Application.hpp>

using namespace igw; // Interactive GUI Wrappers

// ...
// A developer would use this sort of syntax to specify
// an interactive image processing application.

Denoiser denoised(ImageReader());
Thresholder intialRegion(denoised);
Sigmoid speedImage(Gradient(denoised));
ActiveContours contour(initialRegion, speedImage);
// The interactive application containing the display and
// input widgets for all of the filters that contour depends on.
Application thresholdRefineApp(contour);

// ...
// Or, naming fewer of the filter wrapper objects
Denoiser denoised(ImageReader());
ActiveContours contour(Thresholder(denoised), Sigmoid(Gradient(denoised)));
Application thresholdRefineApp(contour);

// ...
```

Figure 26 Hypothetical Source Code

7.4 Filter Wrapper Requirements

The only thing the `Application` needs to know about filter wrappers is how to get their renderer and the widget group used to specify parameter values to the filter. The choice of which renderer use, which parameters to expose using which widgets, and how the parameter input components are laid out should be left entirely up to the author of the filter wrapper.

Since we are using VTK for visualizing and rendering the data, there are many different visualizers to choose from. In fact, an elaborate graph of VTK filters can be created (53) to visualize the results of one ITK filter.

The filter wrapper should also expose the ITK filter directly. ITK has a powerful data flow infrastructure with an intuitive interface for linking filters into pipelines. The wrappers are only intended to automate parameter specification and the other tasks outlined in this section, not to provide an opaque interface to the underlying image processing pipeline. The functional style constructor illustrated in the hypothetical syntax would just be convenient shorthand for specifying the filter connections. Most notably, linking multiple outputs from a single filter is most effectively done using the output getter methods that ITK already provides.

It will be important to support the use of widget inputs to control widget values, such as is needed to ensure that the minimum threshold that a user can specify can never be larger than the maximum threshold for the same image.

A more intricate requirement will be the ability to use filter values to control widget values. This becomes necessary when making a slider for selecting a slice out of an MRI volume or a specific connected component from a thresholded image. Neither of these values are known when the application is written, and the number of connected components can change based on the threshold value specified through another filter in the pipeline.

The filter wrappers should handle saving and loading parameter values specified through the GUI. This is a very simple requirement to implement, but it is largely what makes the entire interactive parameter specification process worthwhile. As a very simple example, one can use parameter values saved in an earlier run of the program as a starting point for analyzing new data.

Finally, one can determine whether or not the filter should be enabled by checking whether all of the source filters further upstream from it have loaded data. There should be no need to explicitly specify the state transitions, as had to done in the current coding framework.

8 Discussion and Future Work

We have demonstrated the first steps in automating the phantom generation process from MRI volumes of real volunteers. Having a fast and accurate segmentation process is going to be essential for generating the libraries of segmentations that will be necessary for generating the dynamic anthropomorphic phantoms needed to perform motion simulation and testing. The myriad of image processing filters that ITK provides combined with the next iteration of the Interactive GUI framework for ITK filters will make developing such elaborate segmentation applications tractable.

Future directions include increasing the accuracy and reducing the time necessary to segment the volumes by using more sophisticated segmentation and image processing techniques. We can use a database of previously preformed segmentations of the organs and look for them in the most probable places (50). We can explicitly look for specific structures within the volumes, since, for example, we know there should be twelve ribs in the body that attach to the vertebrae in known locations. This would really help for the thin front ribs, which are missed in most of the demonstrated segmentations. It would also complete the spine segmentation we partially accomplish in this work.

The "threshold then refine" method discussed in this work doesn't work well for segmenting the spine; it is made up of multiple intensities in these images, making it difficult to generate an initial contour. Fortunately, there are other structures and features that can be used to segment the spine: the structure of the spine can be summed up as a curved column of vertebrae separated by spinal discs. In the MRI, this looks like a curve of dark squares separated by light lines. One can use a technique that can exploit these features directly to improve the spine segmentation (51). One can also use the locations of the ribs from previous slices as a starting point. Additionally, 3D segmentation can try to extract the ribs and vertebrae as one three-dimensional region (52).

In future work, we plan to develop ways to extract other structures present in thorax, such as the scapula, sternum, and collar bone. There also seems to be enough contrast between the blood and the muscles in the heart in these images to extract the heart walls. Some of these structures can be extracted using the threshold and refine approach used in this work, while others will require a more sophisticated approach.

9 Appendix – HLING

HLING uses two concrete syntax trees used to represent applications. The `user-spec` defines the syntax we used to write interactive GUI programs in this work. The `user-spec` gets translated into an `application` which expands all of the shorthands used in the `user-spec` into a form that can be used directly to generate a set of C++ files implementing the application. The C++ code is compiled into the final executable we used to generate our segmentation results.

The language grammars are structures with fields whose types are specified using Scheme contracts. The basic structure of a program is defined by the app-TYPE, filterT (filter type), and filterO (filter object) types. The filterO's use the callback type provide a declarative syntax (despite their name) for specifying what input widgets' values feed into which formal parameters of the filter's setter methods.

The filters which we referred to as "transform filters" are actually called just "filters" in ITK. This naming convention makes the term "filter" ambiguous, and so we renamed it in the body of the report. In this grammar, however, we follow ITK's convention, overloading the word to mean both the general unit of the ITK framework (source, transform, and sink) or only the transform filter. The intended meaning should be evident from context.

9.1 HLING Concrete Syntax Structures

```
;; variable and function names have to comply with C++ format
(define var-exp  #rx"^[a-zA-z][a-zA-Z0-9]*$")

;; a lot of things have names which one needs to get at easily
(define (named item-type?)
  (cons/c var-exp item-type?))

(define-type app-TYPE

  ;; the abstract syntax for what the user types
  (user-spec (title string?)
             (pixel-types (listof (named pixelT?)))
             (image-types (listof (named imageT?)))
             (source-types (listof (named filterT?)))
             (filter-types (listof (named filterT?)))
             (sink-types (listof (named filterT?)))
             ;; the type name, the include, and the typedef
             (helper-types (listof (named (cons/c string? string?))))
             (filter-objects (listof (named filterO?)))
             (connections (listof connection?))
             ;; list of the states
             (states (listof var-exp))
             ;; a transition is either
             ;; ((filter-name callback-name param-name) . added-state-name)
             ;;   Adds added-state-name to the state when the filter's callback
             ;;   parameter is set successfully. Only the names of (open-file)
             ;;   callbacks are valid for callback-name.
             ;; (filter-name . added-state-name)
             ;;   expands to the above form by finding the first (open-file)
             ;;   callback in the filter
             (state-transitions
               (listof
                (or/c
                  (cons/c
                    (cons/c var-exp (cons/c var-exp (cons/c var-exp empty)))
                     var-exp)
                  (cons/c var-exp var-exp))))
            ;; the states for which the specified filters are activated
            ;; (((state-name ...) (filter-name ...)) ...)
            (state-filters (listof (cons/c (listof var-exp) (listof var-exp)))))

  ;; The abstract syntax from which the application is compiled.
  ;; Produced by (pre)processing the user-spec.
  ;; filterT* and filterO* are the expanded versions of
  ;; their counterparts in the user-spec.
  (application (title string?)
               (pixel-types (listof (named pixelT?)))
               (image-types (listof (named imageT?)))
               (source-types (listof (named filterT*?)))
               (filter-types (listof (named filterT*?)))
               (sink-types (listof (named filterT*?)))
               (helper-types (listof (named (cons/c string? string?))))
               ;; automatically generated types for displaying in VTK
               ;; and saving ITK filters' outputs
               (display-types (listof (named filterT*?)))
               (writer-types (listof (named filterT*?)))
               (filter-objects (listof (named filterO*?)))
               ;; automatically generated display and writer objects
               (display-objects (listof (named filterO*?)))
               (writer-objects (listof (named filterO*?)))
               (connections (listof connection*?))
               ;; automatically generated filter connections for
               ;; the displays and writers
               (display-cons (listof connection*?))
               (writer-cons (listof connection*?))
               (states (listof var-exp))
               (state-transitions
                 (listof
                   (cons/c
                     (cons/c var-exp (cons/c var-exp (cons/c var-exp empty)))
                     var-exp)))
               (state-filters (listof (cons/c (listof var-exp) (listof var-exp))))))


(define-type program-TYPES
  (pixelT (base-type string?))
  (imageT
   ;; the ITK image class
   (template var-exp)
   ;; the pixel-type can only be one of the defined pixel types
   (pixel-type var-exp)
   (dimensions exact-positive-integer?))
  (filterT
   ;; the ITK filter type
   (template var-exp)
   ;; The filter's parameter types, and the functions they are associated with.
   ;; The types specified as strings are not checked to be valid image types.
   ;; The types specified using the (listof var-exp) construct are of the form
   ;; '(type-name func1-name func2-name ...)
   ;; type-name is checked to be a valid image type.  The func-names are then
   ;; bound to having that (input or output) type.
   ;; Most filters will just use the 'SetInput and 'GetOutput
   ;; The first two types may be specified using a single type, with no associated
   ;; functions.  For sources, the first type specified is assumed to be for
   ;; GetOutput.  For sinks, the first type specified is assumed to be for SetInput.
   ;; For filters, the first type is for SetInput and the second is for GetOutput.
   (type-args (listof (or/c var-exp (cons/c var-exp (listof var-exp)))))
   ;; specification of which of the above typed functions are inputs or outputs
   (inputs (listof var-exp))
   (outputs (listof var-exp)))
  ;; sources, filters, and sinks are treated differently,
  ;; but specified in the same way
  (filterO
   ;; the one of the defined filter types that this object realizes
   (type var-exp)
   ;; short, descriptive label
   (label string?)
   ;; code will be run once to initialize a filter object.
   (init string?)
   ;; the functions that define interaction with this class
   (parameters (listof (named callback?)))
   ;; adds the default functionality
   (auto-sinks
     (listof
       (or/c 'none 'all
             (listof
               (or/c (or/c 'display (cons/c 'display string?))
                     (or/c 'writer (cons/c 'writer string?))))))))
  (connection
   ;; a connection specifies all the inputs for a specific filter
   ;; in the order they are listed in the filter type
   (filter-name var-exp)
   ;; a list of output functions from other filters corresponding
   ;; to the inputs listed in the filter type definition
   ;; The (cons symbol symbol) form represents (filter-name function-name)
   ;; for the input
   ;; The var-exp form represents the input filter's name and expands to
   ;; (filter-name 'GetOutput)
   (inputs (listof (or/c var-exp (cons/c var-exp var-exp)))))
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;; types that enforce constraints valid after preprocessing
  (filterT*
   (template var-exp)
   ;; all type args are in the (image-type func1 ...) format
   (type-args (listof (cons/c var-exp (listof var-exp))))
   (inputs (listof var-exp))
   (outputs (listof var-exp)))
  (filterO*
   (type var-exp)
   (label string?)
   (init string?)
   (parameters (listof (named callback?)))
   (display-labels (listof (or/c string? 'none)))
   (writer-labels (listof (or/c string? 'none))))
  (connection*
   (filter-name var-exp)
   ;; all inputs are in the (filter-name func-name) form
   (inputs (listof (cons/c var-exp var-exp))))
  ;;
  (callback (header (listof (named (cons/c string? input?))))
            (body cb-body?)))

(define-type cb-body
  ;; the method-name is the same as a setter method for the specific filter
  ;; that takes the specified inputs
  (setter)
  ;; the method-name does not (necessarily) correspond to a specific setter
  ;; within the class.  Instead, the behavior is specified in the body string
  ;; as C++ code.
  (stub (body string?)))

(define-type input
  ;; a labeled text-box that takes in numbers of the specified type
  (text (type input-type?))
  ;; a labeled slider that enumerates the numbers of the specified type in the given
range
  (slider (type input-type?)
          ;; min and max can either have a static value the entire time, or
          ;; they can get their value from a getter.  In that case, a default
          ;; value is specified until the field is enabled, after which point
          ;; the getter function's output is used as the value.
          ;; (cons default-value getter)
          (min (or/c number? (cons/c number? (named getter-stub?))))
          (max (or/c number? (cons/c number? (named getter-stub?))))
          (val number?)
          ;; disregarded for integer type
          (num-values exact-positive-integer?))
  ;; a boolean input value
  (check-box)
  ;; for readers only.
  (open-file)
  (open-directory)
  ;; for writers only
  (save-file))

(define-type getter-TYPE
  (getter)
  (getter-stub (body string?)))

;; the types of values an input field generates or takes in
(define input-type?
  (or/c 'int 'float))
```

10 Bibliography

1. *Prognostic utility of the exercise thallium-201 test in ambulatory patients with chest pain: comparison with cardiac catheterization.* **S Kaul, DR Lilly, JA Gascho, DD Watson, RS Gibson, CA Oliner, JM Ryan and GA Beller.** s.l. : American Heart Association , 1988, Circulation, Vol. Vol 77, pp. 745-758.

2. *Inflammation, Atherosclerosis, and Coronary Artery Disease.* **Hansson, Göran K.** 16, s.l. : The New England Journal of Medicine, April 21, 2005, Vol. 352.

3. **Michael A. King, Stephen J. Glick, P. Hendrik Pretorius, R. Glenn Wells, Howard C. Gifford, Manoj V. Narayanan, and Troy Farncombe.** Attenuation, Scatter, and Spatial Resolution Compensation in SPECT. [book auth.] Aarsvold J. N. Wernick M. N. *Emission Tomography: The Fundamentals of SPECT and PET. 1st ed.* San Diego, CA : Elsevier, 2004, pp. 473–498.

4. **Cullom, S. James.** Principles of Cardiac SPECT. [ed.] Ernest V. Garcia, Daniel Sholom Berman E. Gordon DePuey. *Cardiac SPECT Imaging.* 2001.

5. *Exercise Myocardial Perfusion SPECT in Patients Without Known Coronary Artery Disease.* **Rory Hachamovitch, MD, et al.** Orlando, Fla : American Heart Association, June 1994, Circulation, pp. 905-914.

6. *Analytic and Iterative Reconstruction Algorithms in SPECT.* **Bruyant, Philippe P.** 10, s.l. : Journal of Nuclear Medicine, 1993, Vol. 43, pp. 1343-1358.

7. *Maximum Liklihood Reconstruction for Emission Tomography.* **Vardi, L. A. Shepp and Y.** 2, October 1982, IEEE Transactions on Medical Imaging, Vol. 1.

8. *Correction of Heart Motion Due to Respiration in Clinical Myocardial Perfusion SPECT Scans Using Respiratory Gating.* **Gil Kovalski, Ora Israel, Zohar Keidar, Alex Frenkel, Jonathan Sachs and Haim Azhari.** 4, 2007, Journal of Nuclear Medicine, Vol. 48, pp. 630-636.

9. *Use of three-dimensional Gaussian interpolation in the projector/backprojector pair of iterative reconstruction for compensation of known rigid-body motion in SPECT.* **B. Feng, H. Gifford, R. Beach, G. Boening, M. Gennert, and M. King,.** July 2006, IEEE Transactions on Medical Imaging, Vol. 25, pp. 838-844.

10. *A Quantitative Assessment of Patient Motion and Its Effect on Myocardial Perfusion SPECT Images.* **Elias H. Botvinick, YuYing Zhu, William J. O'Connell and Michael W. Dae.** 2, 1993, The Journal of Nuclear Medicine, Vol. 34, pp. 303-310.

11. *Single- vs. dual-head SPECT for detection of myocardial ischemia and viability in a large study population.* **J. Bucerius, A. Joe, I. Lindstaedt, A. Manka-Waluch, K. Reichmann, S. Ezziddin, H. Palmedo, H. Biersack.** 4, July 2007, Clinical Imaging, Vol. 31, pp. 228-233.

12. *Dynamic Acquisition with a Three-Headed SPECT System: Application to Technetium 99m-SQ30217 Myocardial Imaging.* **Kenichi Nakajima, Junichi Taki, Hisashi Bunko, Masamichi Matsudaira, Akira Muramori, Ichiro Matsunari, Kinichi Hisada and Takashi Ichihara.** No. 6, 1991, The Journal of Nuclear Medicine, Vol. 32, pp. 1273-1277 .

13. *Fully 4D motion-compensated reconstruction of cardiac SPECT images.* **Erwan Gravier, Yongyi Yang, Michael A King and Mingwu Jin.** 18, 2006, Physics in Medicine and Biology, Vol. 51.

14. *4D Affine Registration Models for Respiratory-Gated PET.* **GJ Klein, BW Reutter, and RH Huesman.** 2000. IEEE Nuclear Science Symposium. Vol. 15, pp. 41-45.

15. **Gu, Songxiang.** *Body Deformation Correction for SPECT Imaging.* Computer Science, Worcester Polytechnic Institute. 2009. Ph. D. Dissertatoin.

16. *Body Deformation Correction for SPECT Imaging.* **Songxiang Gu, Joseph E. McNamara, Joyeeta Mitra, Howard C. Gifford, Karen Johnson, Michael A. Gennert and Michael A. King.** 2007. Medical Imaging Conference.

17. HeartSite.com. *Treadmill Stress Test .* [Online] http://www.heartsite.com/html/regular_stress.html.

18. *Pharmacologic stress testing for coronary disease diagnosis: A meta-analysis.* **Kim C, Kwok YS, Heagerty P, Redberg R.** 6, December 2001, American Heart Journal, Vol. 142, pp. 934-944.

19. *Dipyridamole-Thallium Imaging: The Lazy Man's Stress Test.* **Leppo, Jeffrey A.** 3, 1989, The Journal of Nuclear Medicine, Vol. 30, pp. 281-287.

20. *From Vulnerable Plaque to Vulnerable Patient.* **Morteza Naghavi, MD, et al.** 2003, Circulation, pp. 1664-1672.

21. **Robert Bonow, M.D.** ABC News. *What Is A Nuclear Stress Test?* [Online] February 6, 2008. http://abcnews.go.com/Health/HeartDiseaseScreening/story?id=4222350.

22. *Normal stress-only versus standard stress/rest myocardial perfusion imaging: similar patient mortality with reduced radiation exposure.* **Chang SM, Nabi F, Xu J, Raza U, Mahmarian JJ.** 3, January 19, 2010, Journal of the American College of Cardiology, Vol. 55, pp. 231-233.

23. SPECT Tutorial. *Acquisition Protocols.* [Online] http://www.physics.ubc.ca/~mirg/home/tutorial/acquisition.html.

24. *Comparison of planar imaging and single-photon emission computed tomography for the detection and localization of coronary artery disease.* **J. P. Hacot, M. Bojovic, J. Delonca1, B. Meier and A. Righetti.** 2, June 1993, The International Journal of Cardiac Imaging, Vol. 9.

25. **Slaney, A. C. Kak and Malcolm.** *Principles of Computerized Tomographic Imaging.* s.l. : IEEE Press, 1988. Available in electronic form at http://www.slaney.org/pct/.

26. *Quantitative SPECT Imaging: A Review and Recommendations by the Focus Committee of the Society of Nuclear Medicine Computer and Instrumentation Council.* **M.S. Rosenthal, J. Cullom, W. Hawkins, S.C. Moore, B.M.W. Tsui and M. Yester.** 8, 1995, The Journal of Nuclear Medicine, Vol. 36, pp. 1489-1513.

27. Image Reconstruction from Projections. [book auth.] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing.* 3. 2008, pp. 362-375.

28. *Quantitative Analysis of the Tomographic Thallium-201 Myocardial Bullseye Display: Critical Role of Correcting for Patient Motion.* **Robert Eisner, André Churchwell, Till Noever, Dave Nowak, Karen Cloninger, Daniel Dunn, Wilma Carlson, Joel Oates, Janie Jones, Douglas Morris, Henry Liberman and Randolph Patterson.** 1, 1988, The Journal of Nuclear Medicine, Vol. 29, pp. 91-97 .

29. *A Hybrid 3D Reconstruction Registration Algorithm for Correction of Head Motion in Emission Tomography.* **B. F. Hutton, A. Z. Kyme, Y. H. Lau, D. W. Skerrett, and R. R. Fulton.** 1, February 2002, IEEE Transactions on Nuclear Science, Vol. 49.

30. *Pattern Independent Deformation Estimation Illustrated by MRI.* **Songxiang Gu, Joseph E. McNamara, Joyeeta Mitra, Howard C. Gifford, Andrey V. Sklyar, Karen Johnson, Michael A. Gennert, Michael A. King.** 2008. Medical Imaging Conference.

31. *Comparison of Four Motion Correction Techniques in SPECT Imaging of the Heart: A Cardiac Phantom Study.* **Michael K. O'Connor, Kalpana M. Kanal, Mark W. Gebhard and Philip J. Rossman.** 12, 1998, The Journal of Nuclear Medicine, Vol. 39, pp. 2027-2034 .

32. *Motion Capture of Chest and Abdominal Markers Using a Flexible Multi-Camera Motion-Tracking System for Correcting Motion-Induced Artifacts in Cardiac SPECT.* **Joseph. E. McNamara, Bing Feng, Karen Johnson, Songxiang Gu, Michael A. Gennert, Michael A. King.** 2007. Medical Imaging Conference.

33. *Patient Motion Correction in Computed Tomography by Reconstruction on a Moving Grid.* **Rostyslav Boutchko, Karthikayan Balakrishnan, Bryan W. Reutter, Grant T. Gullberg.** 2007. IEEE Nuclear Science Symposium.

34. *Automatic nonrigid registration of whole body CT mice images.* **Xia Li, Thomas E. Yankeelov, Todd E. Peterson, John C. Gore, Benoit M. Dawant.** 4, March 2008, Medical Physics, Vol. 35, pp. 1507–1520.

35. *Theoretical and Numerical Study of MLEM and OSEM Reconstruction Algorithms for Motion Correction in Emission Tomography.* **King, Joyoni Dey and Michael A.** 5, October 2009, IEEE Transactions of Nuclear Science, Vol. 56, pp. 2739-2749.

36. *Multimodality Image Registration by Maximization of Mutual Information.* **Frederik Maes, Andre Collignon, Dirk Vandermeulen, Guy Marchal, and Paul Suetens.** 2, April 1997, IEEE Transactions on Medical Imaging, Vol. 16.

37. *SPECT Low-Field MRI System for Small-Animal Imaging.* **Christian Goetz, Elodie Breton, Philippe Choquet, Vincent Israel-Jost and André Constantinesco.** 1, 2007, Journal of Nuclear Medicine, Vol. 49, pp. 88-93.

38. *Initial Investigation of preclinical integrated SPECT and MR imaging.* **Hamamura MJ, Ha S, Roeck WW, Wagenaar DJ, Meier D, Patt BE, Nalcioglu O.** 1, Feb 2010, Technology in cancer research & treatment, Vol. 9, pp. 21-28.

39. **Segars, W.P.** *Development and application of the new dynamic NURBS-based cardiac-torso (NCAT) phantom.* Biomedical Engineering, University of North Carolina: Chapel Hill. 2001.

40. *Modeling respiratory mechanics in the MCAT and spline-based MCAT phantoms.* **Segars, W.P., D.S. Lalush, and B.M.W. Tsui.** 1, 2001, IEEE Transactions on Nuclear Science, Vol. 48, pp. 89-97.

41. *Voxel-Based Computational Models of Real Human Anatomy: A Review.* **Coan, M.** 2003, Journal of Radiation and Environmental Biophysics, Vol. 42, pp. 229-235.

42. **Altmann, Markus.** About Nonuniform Rational B-Splines - NURBS. [Online] http://web.cs.wpi.edu/~matt/courses/cs563/talks/nurbs.html.

43. *Study of the efficacy of respiratory gating in myocardial SPECT using the new 4-D NCAT phantom.* **Segars, W.P. and B.M.W. Tsui.** 3, 2002, IEEE Transactions on Nuclear Science, Vol. 49, pp. 675-679.

44. *Quantitative Assessment of Motion Artifacts and Validation of a New Motion-Correction Program for Myocardial Perfusion SPECT.* **Naoya Matsumoto, Daniel S. Berman, Paul B. Kavanagh, James Gerlach, Sean W. Hayes, Howard C. Lewin, John D. Friedman and Guido Germano.** 5, 2001, Journal of Nuclear Medicine, Vol. 42, pp. 687-694.

45. *A flexible multi-camera visual-tracking system for detecting and correcting motion-induced artifacts in cardiac SPECT slices.* **McNamara JE, Pretorius PH, Johnson K, Mitra J, Dey J, Gennert MA, King MA.** 2009, Medical Physics, Vol. 36, pp. 1913-1923.

46. *Modeling the Respiratory Motion of Solitary Pulmonary Nodules for Investigating SPECT Tumor Imaging.* **Smyczynski MS, King MA, Narayanan MV, Pretorius PH, Gifford HC, Farncombe TH, Hoffman EA, Segars WP, and Tsui.** 2001. Nuclear Science Symposium and Medical Imaging Conference.

47. *A study of the effect of cardiac gating in myocardial SPECT using the 4D NCAT phantom.* **Lee TS, Segars WP, and Tsui.** 2003. Nuclear Science Symposium and Medical Imaging Conference.

48. *A Variational Framework for Joint Segmentation and Registration.* **Zöllei, L., Yezzi, A., and Kapur, T.** 2001. IEEE Workshop on Mathematical Methods in Biomedical Image Analysis.

49. *Variable-Conductance, Level-Set Curvature for Image Denoising.* **Xue, R. Whitaker and X.** 2001, International Conference on Image Processing, Vol. 3, pp. 142-145.

50. *Image analysis using mathematical morphology.* **Haralick, R. M., Sternberg, S. R., Zhuang, X.** 4, 1987 , IEEE Pattern Analysis and Machine Intelligence, Vol. 9, pp. 532-550.

51. *Geodesic Active Contours.* **V. Caselles, R. Kimmel and G. Sapiro.** 1997, International Journal on Computer Vision, Vol. 22, pp. 61-97 .

52. Morphology. [Online] http://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm.

53. *Deformable meshes with automated topology changes for coarse-to-fine three-dimensional surface extraction.* **Montanvert, J. Lachaud and A.** September 1998, Medical Image Analysis, Vol. 3, pp. 187-207.

54. *MRI Based Assessment of the Extent to Which Stereo-Tracking of Markers on the Chest can Predict Motion of the Heart.* **Michael A. King, Joyoni Dey, Joseph E. McNamara, Joyeeta Mitra, Karen Johnson, Andrey Lehovich, Songxiang Gu, J. C. Ford.** 2008. Medical Imaging Conference.

55. The Visualization Toolkit. [Online] http://www.vtk.org/.

56. Kitware. [Online] http://www.kitware.com/.

57. Segmentation and Registration Toolkit. [Online] http://www.itk.org/.

58. wxWidgets. [Online] http://www.wxwindows.org/.

59. Yet Another Port of wxVTKRenderWindowInteractor. [Online] http://wxvtk.sourceforge.net/.

60. PLT Scheme. [Online] http://www.plt-scheme.org/.

61. *FranTk - A declarative GUI language for Haskell.* **Sage, Meurig.** 2000, ACM International Conference on Functional Programming, pp. 106 - 117.

62. **Krishnamurthi, Gregory Cooper and Shriram.** *FrTime: Functional Reactive Programming in PLT Scheme.* Computer Science, Brown University. 2004. Technical Report.

63. HaskellWiki. [Online] http://www.haskell.org/.

64. Qt - A cross-platform application and UI framework. [Online] http://qt.nokia.com/products.

65. *Statistical Shape Influence in Geodesic Active Contours.* **Michael E. Leventon, W. Eric , W. Eric L. Grimson , Olivier Faugeras.** 2000, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 1.

66. *Adaptive template moderated spatially varying statistical classification.* **Simon K. Warfield, Michael Kaus, Ferenc A. Jolesz and Ron Kikinis.** 1998, Medical Image Computing and Computer-Assisted Interventation, Vol. 1496, pp. 431-438.

67. *3D active contours.* **V. Caselles, R. Kimmel, G. Sapiro3 and C. Sbert.** 1996, International Conference on Analysis and Optimization of Systems Images, Vol. 219, pp. 43-49.

68. *Issues regarding radiation dosage of cardiac nuclear and radiography procedures.* **Cullom, Randall C. Thompson and S. James.** New York : s.n., October 13, 2008, Journal of Nuclear Cardiology, pp. 19-23.