2016

BIVENTRICULAR FINITE ELEMENT MODELING AND QUANTIFICATION OF 3D LANGRAGIAN STRAINS AND TORSION USING DENSE MRI

Zhanqiu Liu
University of Kentucky, lafeir.lew@gmail.com

Digital Object Identifier: http://dx.doi.org/10.13023/ETD.2016.187

BIVENTRICULAR FINITE ELEMENT MODELING AND QUANTIFICATION OF 3D LANGRAGIAN STRAINS AND TORSION USING DENSE MRI

THESIS

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering in the College of Engineering at the University of Kentucky

By
Zhanqiu Liu
Lexington, Kentucky
Director: Dr. Jonathan F. Wenk, Professor of Mechanical Engineering
Lexington, Kentucky

2016

ABSTRACT OF THESIS

BIVENTRICULAR FINITE ELEMENT MODELING AND QUANTIFICATION OF 3D LANGRAGIAN STRAINS AND TORSION USING DENSE MRI

Statistical data suggests that increased use of evidence-based medical therapies has largely contributed to the decrease in American death rate caused by heart disease. And my studies are about two applications of magnetic resonance imaging (MRI) as a non-invasive approach in evidence-based health care research. In my first study, the achievement of a pulmonary valve replacement surgery was assessed on a patient with tetralogy of Fallot (TOF). In order to evaluate the remodeling of right ventricle, two biventricular finite element models were built up for pre-surgical images and post-surgical images. In my second study, 3D Lagrangian strains and torsion in the left ventricle of ten rats were investigated using Displacement ENcoding with Stimulated Echoes (DENSE) cardiac magnetic resonance (CMR) images. Tools written in MATLAB were developed for 2D contouring, 3D modeling, strain and torsion computations, and statistical comparison across subjects.

KEYWORDS: Biventricular Finite Element Modeling, Tetralogy of Fallot, DENSE MRI, 3D Langragian Strains, Rats, Custom Programs with MATLAB

Zhanqiu Liu
04/18/2016

BIVENTRICULAR FINITE ELEMENT MODELING AND QUANTIFICATION OF 3D LANGRAGIAN STRAINS AND TORSION USING DENSE MRI

By

Zhanqiu Liu

Dr. Jonathan F. Wenk
Director of Thesis

Dr. Haluk E. Karaca
Director of Graduate Studies

04/28/2016

ACKNOWLEDGMENTS

Though an individual work, the following thesis would not have been possible without the help and support of many people.

First of all, I would love to express my genuine gratitude to my supervisors, Dr. Jonathan F. Wenk, for tutoring and enlightening me over the years, for his offer of academic freedom, for sparking my thinking, for his patience with my crazy ideas, and for his respect for my decision.

In addition, I am truly grateful to Greg Wehner. It's really generous of you to give me so much of your time, to offer me insights about DENSE, and to timely provide me with instructive comments about my questions.

Thank you to Dr. Julius M. Guccione at the UCSF for allowing me to access Cine CMR images of a patient in TOF. I would also like to thank Dr. Liang Ge at the UCSF for his guidance about MeVisLab.

Next, I wish to thank my thesis committee members: Dr. Keith Rouch and Dr. Moriel Vandsburger. Thank you for taking time to participate in my thesis defense.

Rat preparation and DENSE MRI acquisition were done by Dr. Xiaoyan Zhang and I would particularly like to thank her for producing wonderful images. Thank you to Wang Hua for her on-going support. And thank you to Amir Nikou and Dara Singh for their helps.

Last but not least, a huge hug to my amazing parents. Without their unconditional supports and encouragements, I would not go through all the ups and downs. I cannot thank them enough.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

1.1 Background

According to latest reports of CDC WONDER online database, from 1999-2014, diseases of heart was the No. 1 leading cause of dead for both men and women in the United States [1]. From 1999-2000 to 2013-2014, the death rate caused by heart disease was decreased by 24% [2]. Studies suggest that increased use of evidence-based medical therapies has contributed to 47% of the decrease in US deaths attributable to coronary heart disease from 1980 to 2000 [3]. As a non-invasive approach, magnetic resonance imaging (MRI) has been widely used in many evidence-based health care practices.

1.2 Anatomy of the Heart

As shown in Figure 1.1, the human heart is located between lungs in the middle of chest. It is behind and slightly to the left of sternum. Depicted in Figure 1.2, the human heart is a four-chamber structure consisting of two large chambers as ventricles and two smaller chambers as atria. The septum is the wall separating the left and right ventricles, while the free wall is the rest part of ventricular wall except the septum and the apical wall. There are four important valves inside the heart. The atria are connected to the ventricles by two atrioventricular (AV) valves. Between the left atrium and left ventricle is the mitral valve, while between the right atrium and right ventricle is the tricuspid valve. The arteries are connected to the ventricles via two semilunar valves which include the aortic valve between left ventricle and aorta and the pulmonic valve between right ventricle and pulmonary artery.

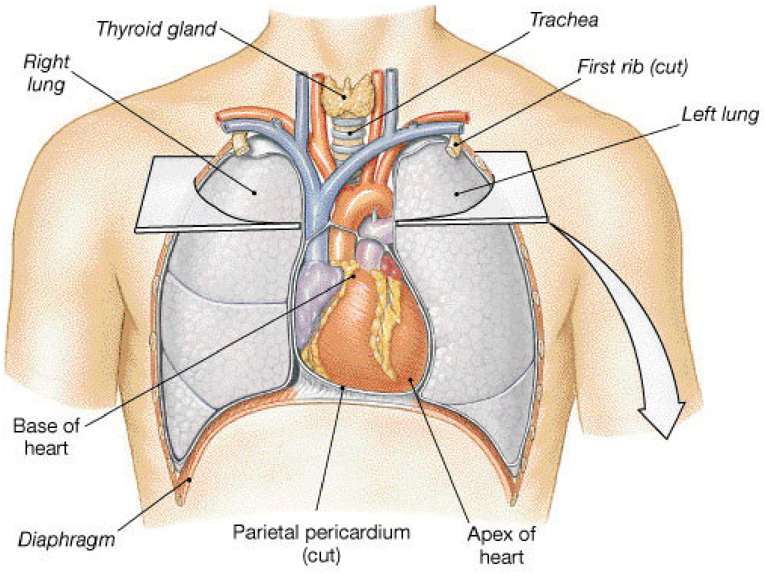


Figure 1.1: Heart location from an anterior view of human body [4]

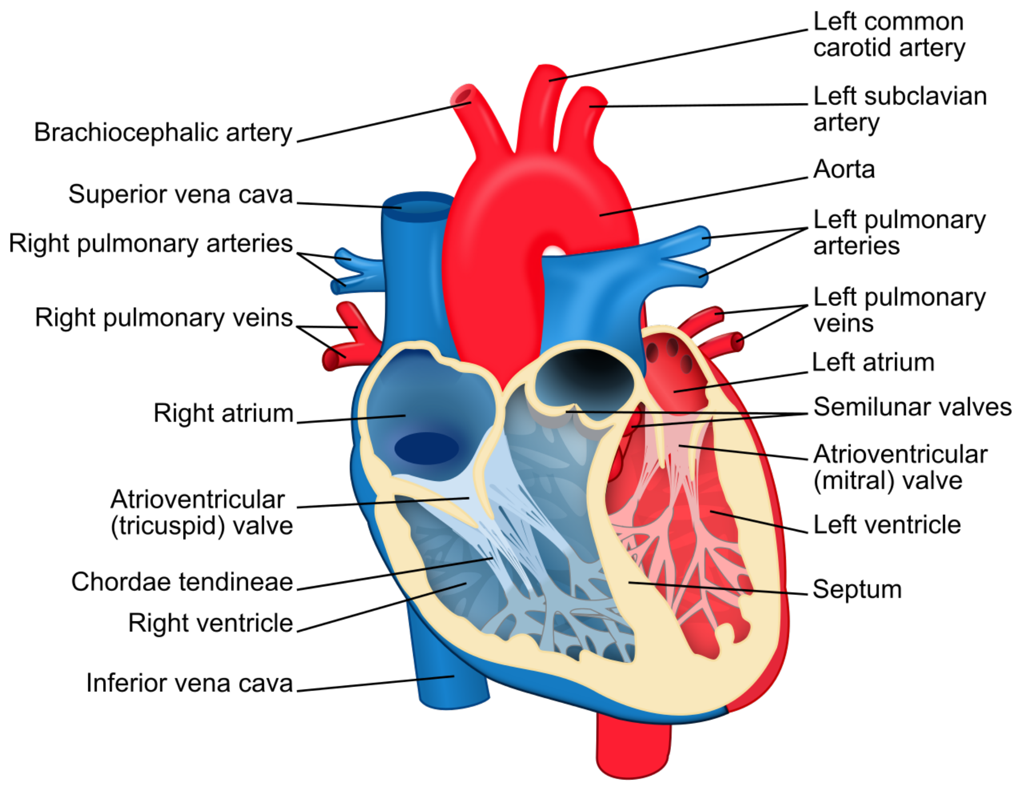


Figure 1.2: Structure Diagram of the human heart from an anterior section [5]

The base of the heart is the widest part and constitutes the upper part of heart, where venae cavae and left pulmonary artery are attached to the left ventricle (LV), and where aorta and left pulmonary veins are attached to the right ventricle (RV). The apex of the heart is the lowest part of the heart. From an anterior view of human body (Figure 1.1), the base of heart is upward and leaning toward the right shoulder, while the apex directs downward, forward, and pointing to the left shoulder.

The ventricular wall consists of three layers (Figure 1.3). The innermost layer is called endocardium, while the outer layer of heart tissue is termed epicardium. The majority of ventricular wall is myocardium locating between endocardium and epicardium. Myocardial fibers lying within the myocardium are responsible for the ventricular contraction. The fiber angle varies from endocardium to epicardium.

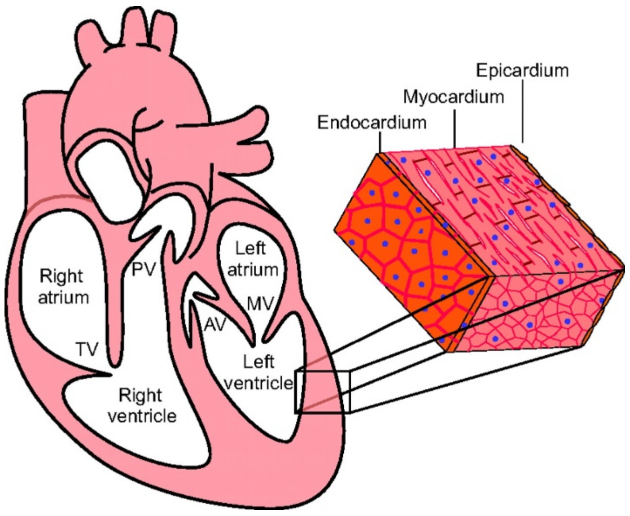


Figure 1.3: Layers of the heart wall [6]

1.3 Physiology of the Heart

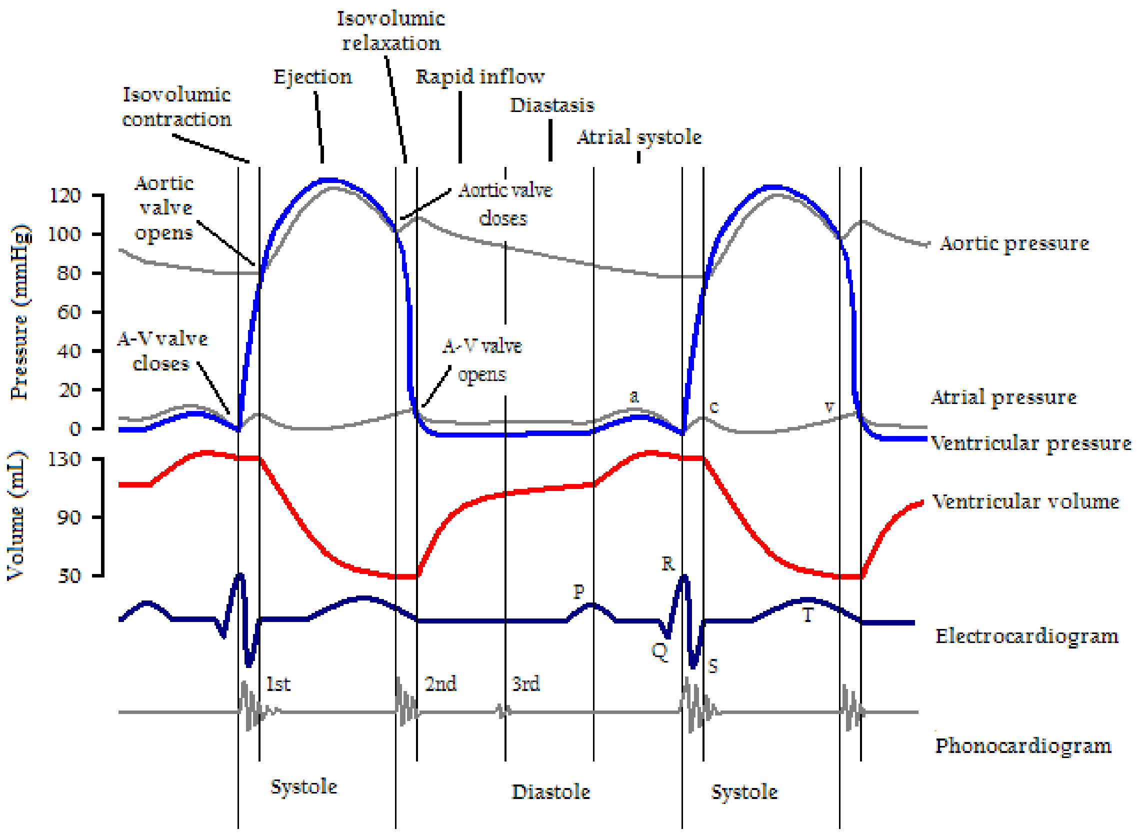


Figure 1.4: Wiggers Diagram of a cardiac cycle [7]

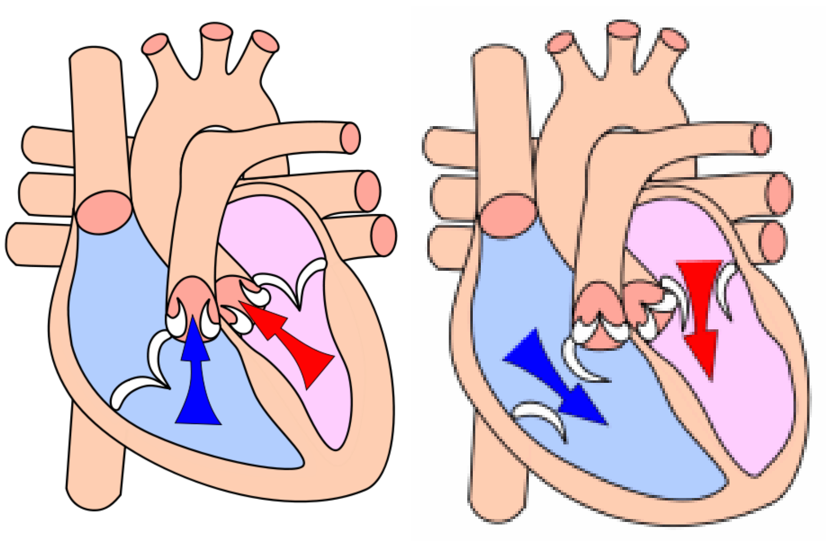
During one heartbeat, contraction and relaxation occur alternately, which is call a cardiac cycle (shown in Figure 1.4). The frequency of the heart beat is the heart rate with beats per minute as most common unit. A cardiac cycle includes two phases, systole and diastole, or five stages (summarized in Table 1.1).

Table 1.1: Phases and stages of cardiac cycle

Cardiac phases	Cardiac stages	AV valves	Semilunar valves
Systole	1. Isovolumetric contraction	Closed	Closed
	2. Ventricular ejection	Closed	Open
Diastole	3. Isovolumetric relaxation	Closed	Closed
	4. Ventricular filling and atrial diastole	Open	Closed
	5. Ventricular filling and atrial systole	Open	Closed

Ventricular systole begins at the QRS complex illustrated on the electrocardiogram (ECG) curve of Figure 1.4. During systole, as shown in Figure 1.5a, blood is driven out of the heart. The left ventricle pumps blood via the aortic valve into systemic part of the body, while the right ventricle pumps blood via the pulmonic valve into the lungs. Systole involves the first two stages. In the first stage, ventricles begin to contract. Since both valves are closed (shown in Table 1.1), no blood can enter or leave the ventricle, and thus the ventricle is contracting isovolumically while pressure is rising dramatically (from time A to time B on Figure 1.6). In the second stage (from time B to time C on Figure 1.6), ejection is rapid at first, slowing down as systole progresses. At the end, ventricles reach minimum volume at end systole (ES).

During diastole, as shown in Figure 1.5b, owing to relaxation, the left ventricle fills with oxygenated blood from the left atrium via the mitral valve, while right ventricle fills with deoxygenated blood from the right atrium via the tricuspid valve. Diastole involves the next three stages. In the third stage, both valves are closed again as ventricles relax and pressure within drops. Thus, volume stays the same but pressure drops rapidly (from time C to time D on Figure 1.6). The fourth stage is in early diastole, both the atria and ventricles are relaxed and passively filled in decreasing rate. Late diastole commences at the fifth stage, atrial systole, when P wave appeared in ECG curve (Figure 1.4). As atria contract, blood is pumped into ventricles again. The additional flow of blood is called atrial kick. Eventually ventricles reach maximum volume at end diastole (ED).



(a) Systole [8] (b) Diastole [9]

Figure 1.5: Schematic of blood circulation with red arrows as oxygenated blood and blue arrows as deoxygenated blood

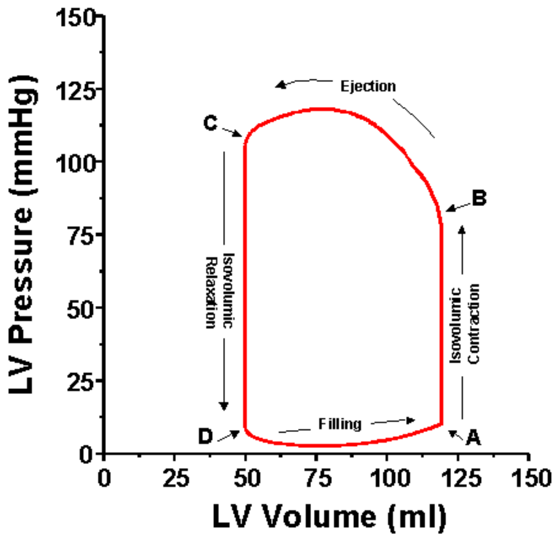
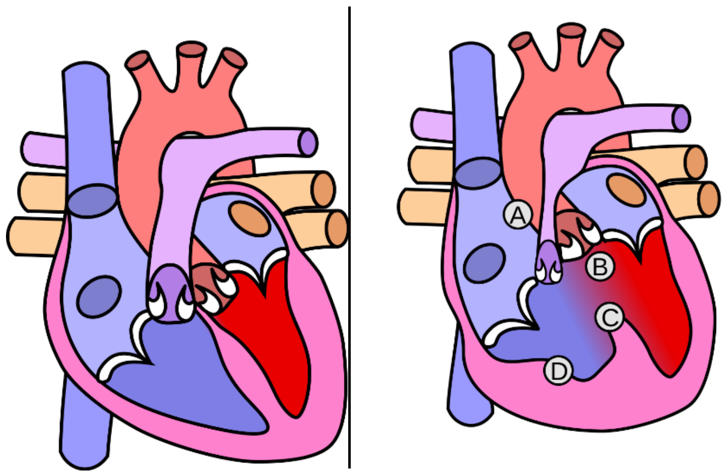


Figure 1.6: Pressure volume loop of a typical human cardiac cycle [10]

2 Assessment of a Pulmonary Valve Replacement in Tetralogy of Fallot

2.1 Background

Recent statistics from the American Heart Association (2011) indicate that Congestive Heart Failure (CHF) affects 5.7 million Americans [11]. CHF is characterized by the hearts inability to pump blood efficiently into the body, caused by either impaired myocardial contraction leading to global ventricular dilation (systolic heart failure), or decreased ventricular relaxation (diastolic heart failure). The tetralogy of Fallot (TOF) is one type of the most common congenital heart defect. It often results in impaired functional capacity and even early death. By definition, TOF involves four abnormalities co-occurring frequently (summarized in Figure 2.1b). In Figure 2.1b, location A shows a narrowing of the right ventricular outflow tract near pulmonary valve. Location B shows an over-riding aortic valve connecting both LV and RV due to a ventricular septal defect as shown in location C. And since RV has more muscle in location D, a characteristic boot-shaped bump appears. All of these lead to abnormal RV outflow causing RV dilation. One of typical symptoms of TOF is cyanosis, like the blue coloration of the finger-tips shown in Figure 2.2.



(a) Normal heart [12] (b) Tetralogy of Fallot [13]

Figure 2.1: Comparison between a healthy heart and one with tetralogy of Fallot

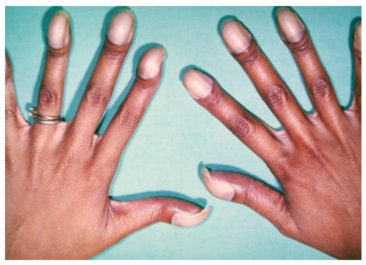


Figure 2.2: Cyanotic nail beds in an adult [14]

Patients with TOF usually take corrective surgery within the first year of life. But they still experience long-term problems including arrhythmia, pulmonary regurgitation, and re-operation [15]. And after the repair of TOF, pulmonary valve replacement (PVR) can reduce pulmonary regurgitation and decreases RV dilation for a patient.

Our goal of this study is to evaluate if the pulmonary valve replacement surgery helps the patient to recover from abnormal remodeling of RV.

2.2 Methods

2.2.1 Cine MRI Acquisition

Cine MR angiography images of the heart are basically short movies with the ability to display average heart motion throughout the cardiac cycle. Cine MRI is taken the same way as a traditional MRI. A cardiac cycle begins with the R wave of the ECG, ends with the subsequent R wave, thus ECG tracing is used to trigger cine MRI. During the acquisition, a cardiac cycle is split into multiple frames depending on the heart rate. The heart is repeatedly imaged at a single slice location throughout the cardiac cycle so that each image is the information obtained during the same frame across multiple cardiac cycles (the same colored boxes in Figure 2.3).

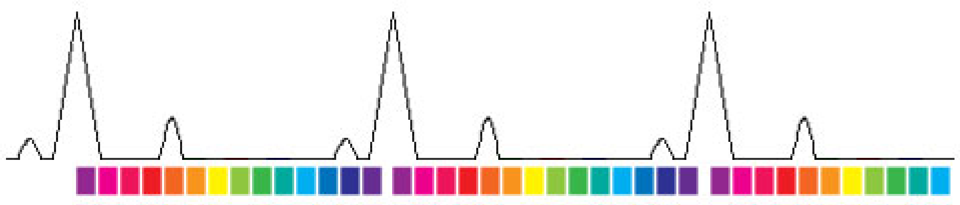


Figure 2.3: An ECG tracing with multiple frames in different colors

Depicted in Figure 2.4, in order to generate tomographic images, three cardiac planes are introduced during MRI [16]. These three corresponding views are perpendicular to each other. Two chambers, left ventricle and atrium, are seen in the vertical long-axis (VLA) view, while all four chambers are shown in the horizontal long-axis (HLA) view.

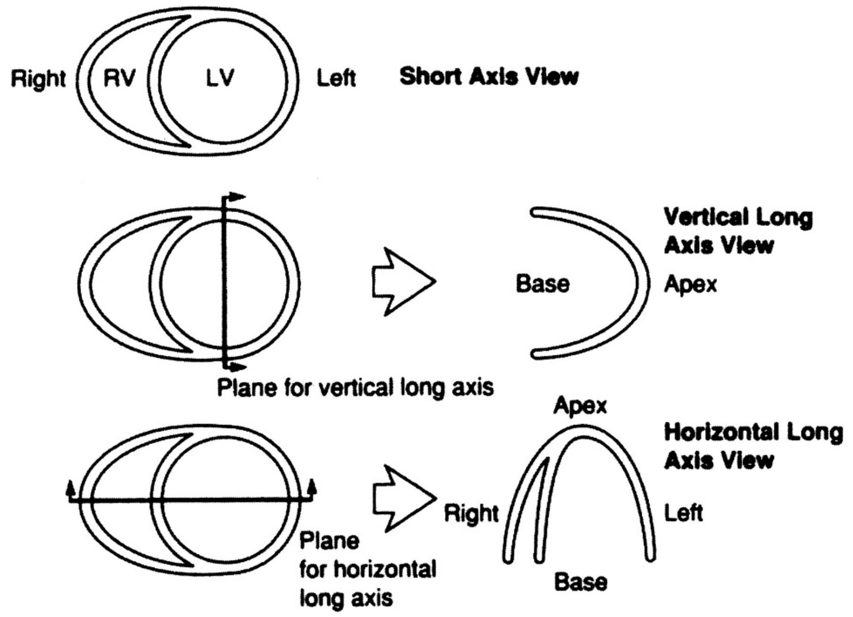

Right
RV
LV
Left
Short Axis View
Vertical Long Axis View
Base
Apex
Plane for vertical long axis
Apex
Horizontal Long Axis View
Right
Left
Plane for horizontal long axis
Base

Figure 2.4: Definitions of short axis (SA), vertical long-axis (VLA), and horizontal long-axis (HLA) views [17]

In order to fully assess the entire heart, separate cine image sets are acquired at various locations, like base, mid-ventricle, and apex for short-axis images, or various views, like 2-, 3-, and 4-chamber views for long-axis images. At last, the series of images are gathered together to produce a movie (cine). A typical acquisition of images for a given slice location are in 10 to 20 seconds depending on the sequence, well within the single breath hold capabilities of most patients. Since the base pulse sequence is a bright-blood technique, dark pixels in the images represent tissue.

A cine MRI scan was performed on a female patient with repaired TOF at the University of California, San Francisco (UCSF). During the scan, 16 short-axis slices and 24 long-axis slices of the 4-chamber view were acquired with a 7-mm slice thickness. Additionally, 16 frames of images in a cardiac cycle were collected with 0.05 s as repetition time, leading to a heart rate of 75 beats per minute.

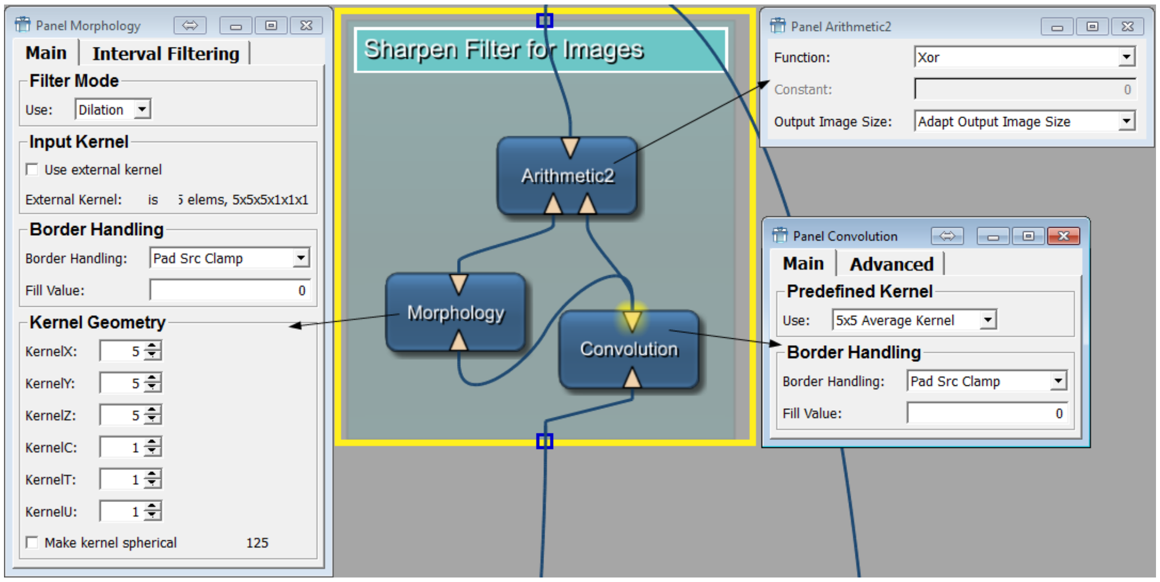
Two years after pulmonary valve replacement surgery, the patient had another cine MRI scan at UCSF. This scan was set up with 14 short-axis slices with a 8-mm slice thickness, single long-axis slice of the 4-chamber view, and single long-axis slice of the 2-chamber view. In this time, 24 frames were imaged throughout a cardiac cycle with 0.031 s as repetition time, which results in a heart rate of 81 beats per minute.

2.2.2 Contouring Technique

In order to post-process cine images, MeVisLab (MeVis Medical Solutions AG and Fraunhofer MEVIS) was used to replicate the in-vivo geometry of biventricular contours.

MeVisLab is a research and rapid prototyping platform for medical image processing and visualization. It provides us a powerful modular framework of visual programming. It integrates many widely-used third-party libraries, such as graphics standard OpenGL, the application framework Qt, and the visualization and interaction toolkit Open Inventor. Moreover, more dynamic functions can be added to a self-developed application via scripting utilizing popular scripting languages, such as JavaScript and Python.

Applications and algorithms were developed visually in a manner of modular design. Depicted in Figure 2.5, each unit (module) has its own input, output, and function implemented by built-in method or an algorithm. For example, by manipulating the processing pipeline connecting modules, three built-in modules built a subnetwork of sharpen filter for images (Figure 2.5).



Panel Morphology

Main | Interval Filtering

Filter Mode

Use: Dilation

Input Kernel

☐ Use external kernel

External Kernel: is 5 elems, 5x5x5x1x1x1

Border Handling

Border Handling: Pad Src Clamp

Fill Value: 0

Kernel Geometry

KernelX: 5

KernelY: 5

KernelZ: 5

KernelC: 1

KernelT: 1

KernelU: 1

☐ Make kernel spherical 125

Panel Arithmetic2

Function: Xor

Constant: 0

Output Image Size: Adapt Output Image Size

Panel Convolution

Main | Advanced

Predefined Kernel

Use: 5x5 Average Kernel

Border Handling

Border Handling: Pad Src Clamp

Fill Value: 0

Figure 2.5: Sharpen Filter for Images

Since cine MRI follows Digital Imaging and Communications in Medicine (DICOM) standard for handling, storing, printing, and transmitting information, a built-in DICOM reader module with multiple visualization modules can implement a function of reading images (shown in Figure 2.6).

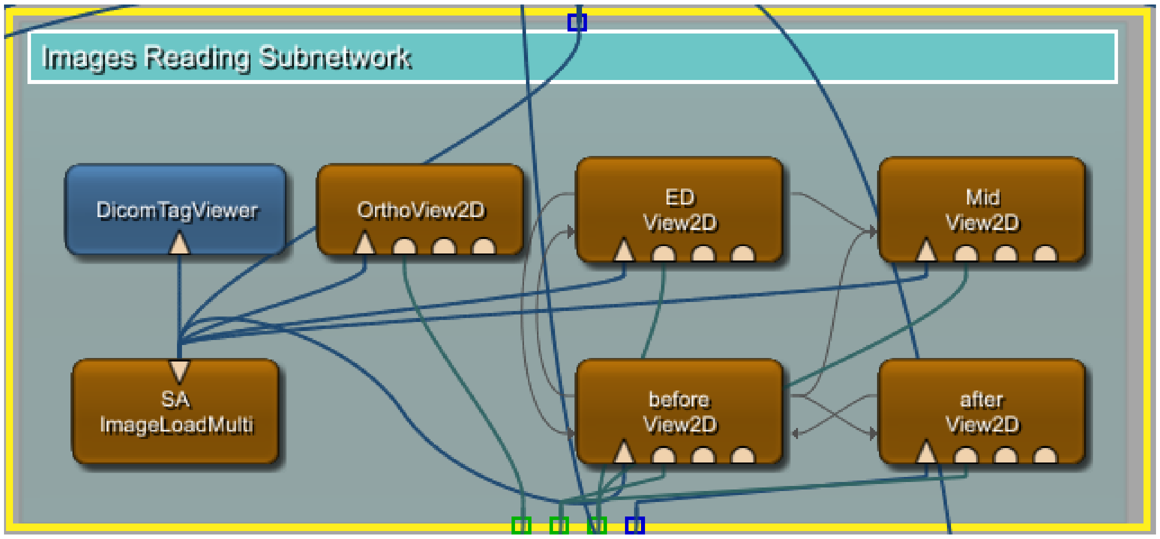


Figure 2.6: Images reading subnetwork

In order to extract the endocardial and epicardial biventricular surfaces, contours were drawn freehand on end-systolic and end-diastolic images (Figure 2.7) of each slice. Since MeVisLab provides many built-in modules for analyzing, maintaining, grouping, and converting contour segmentation objects (CSO), hand-drawn contours were stored with CSO which are in pixel coordinates. A subnetwork for drawing, managing, and saving CSO is shown in Figure 2.8.

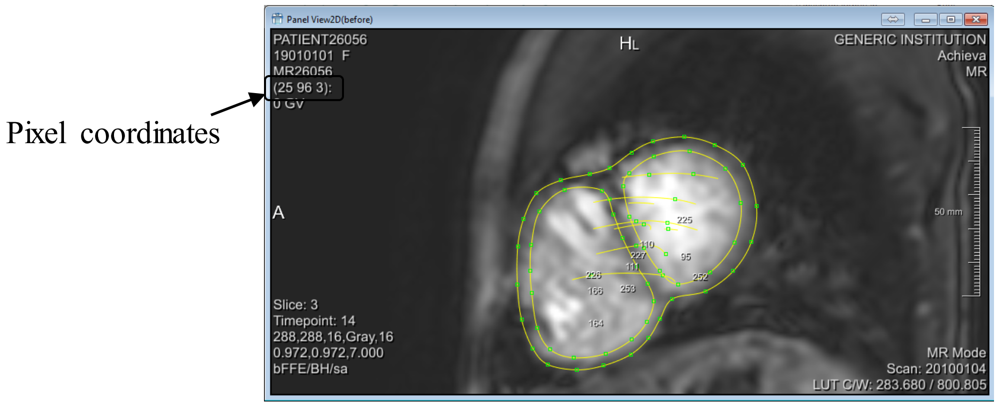


Figure 2.7: Contours in pixel coordinates drawn on a pre-surgical end-diastolic image

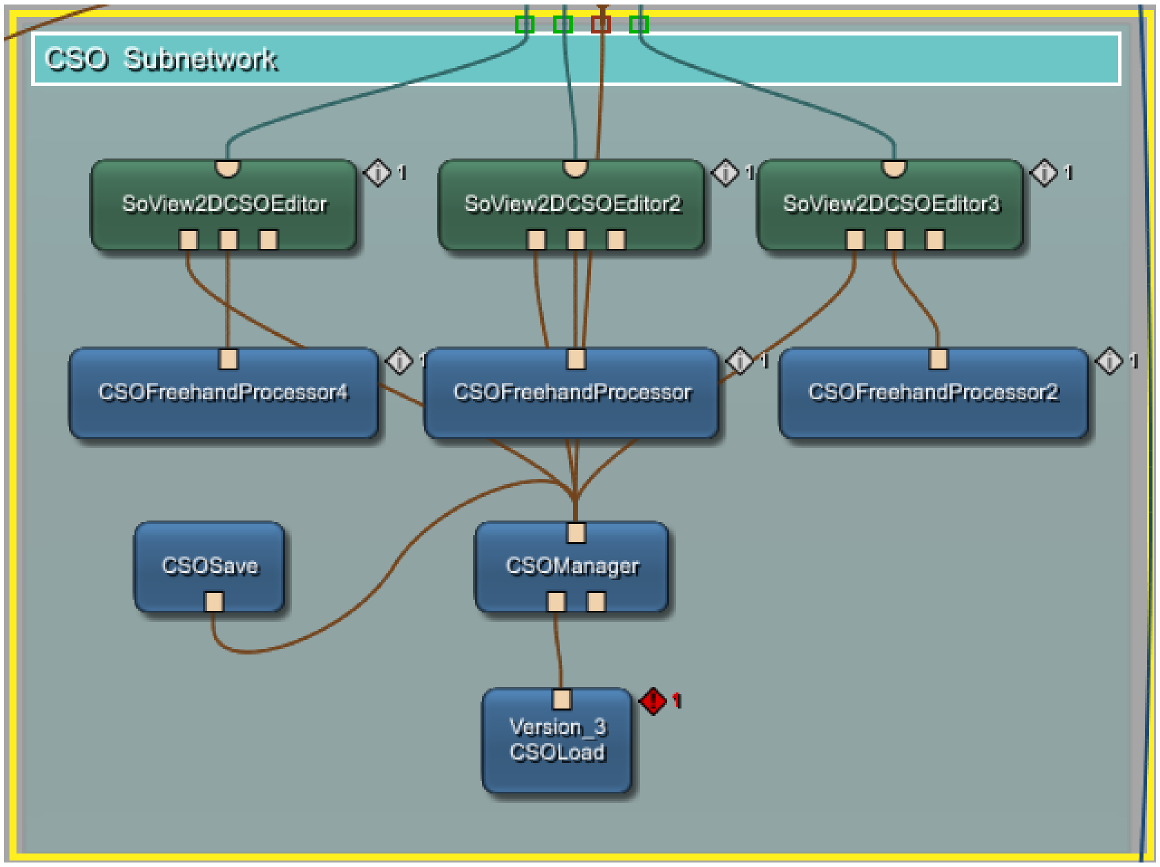


Figure 2.8: CSO manipulation subnetwork

Finally, an application for contouring was done via connecting images reading subnetwork, sharpen filter subnetwork, and CSO manipulation subnetwork (shown in Figure 2.9).

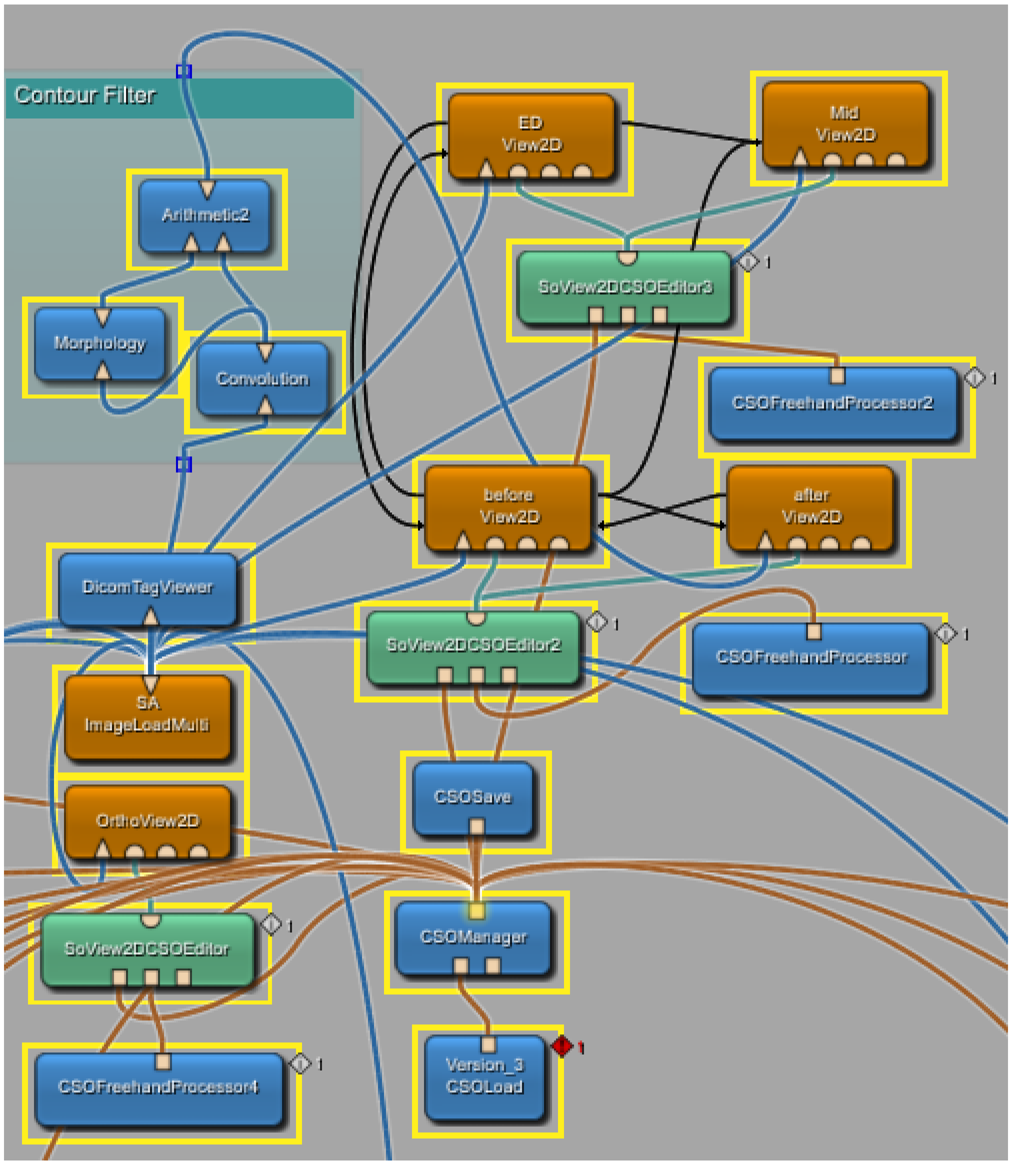
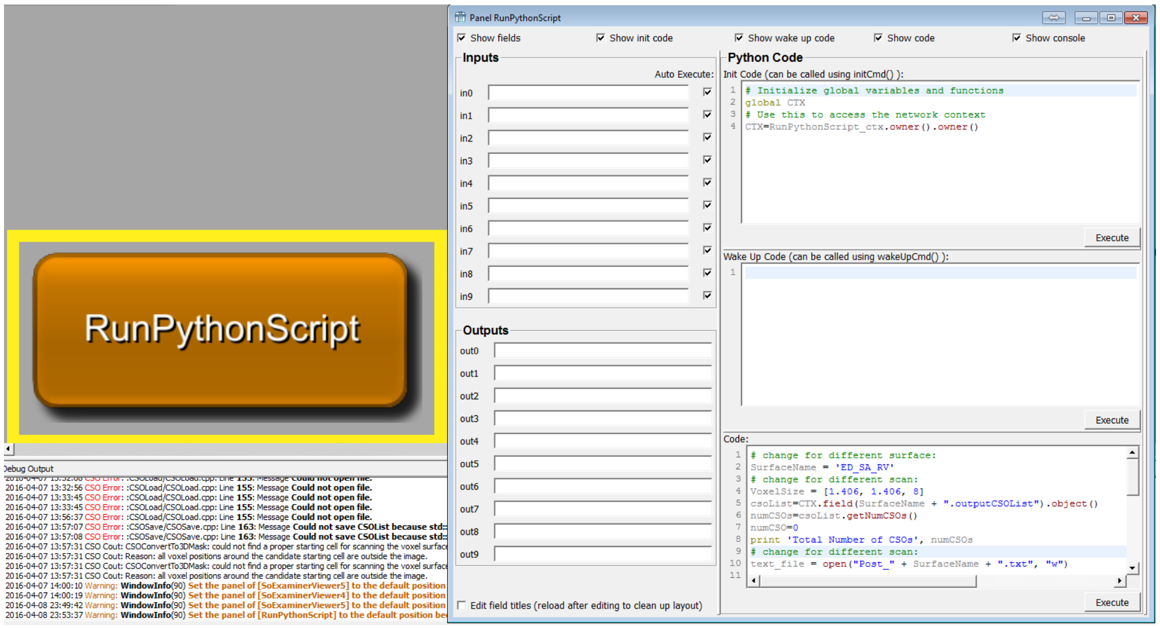


Figure 2.9: Contouring application created with Mevislab

CSO were stored in pixel coordinates. In order to reconstruct the actual size of a heart, the information of volumetric pixels (voxels) size, slice location, and slice orientation stored in the tags of DICOM was used to convert CSO data into Cartesian coordinates in the global coordinate system created by the scanner. A self-developed macro module (Appendix B) was coded with Python to get the actual Cartesian coordinates of biventricular contours (Figure 2.10).



Init Code (can be called using initCmd()):

```
# Initialize global variables and functions
global CTX
# Use this to access the network context
CTX=RunPythonScript_ctx.owner().owner()
```

Wake Up Code (can be called using wakeUpCmd()):

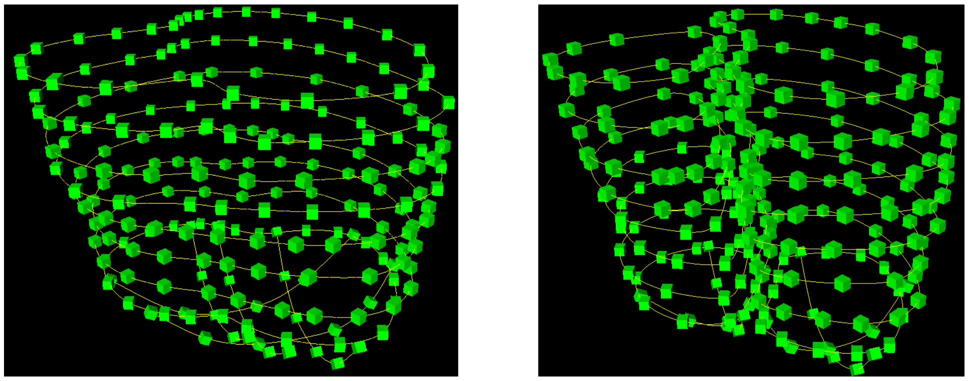
Code:

```
# change for different surface:
SurfaceName = 'ED_SA_RV'
# change for different scan:
VoxelSize = [1.406, 1.406, 8]
csoList=CTX.field(SurfaceName + ".outputCSOList").object()
numCSOs=csoList.getNumCSOs()
numCSO=0
print 'Total Number of CSOs', numCSOs
# change for different scan:
text_file = open("Post_" + SurfaceName + ".txt", "w")
```

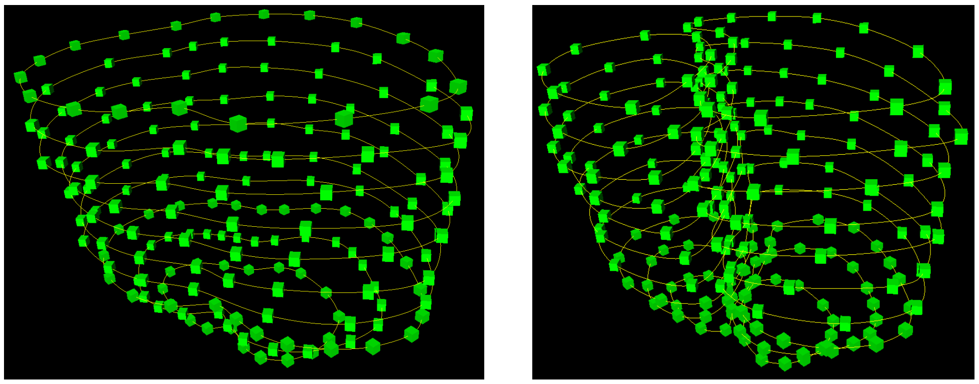
```
2016-04-07 13:32:56 CSO Error: :CSOLoad/CSOLoad.cpp: Line 155: Message Could not open file.
2016-04-07 13:33:45 CSO Error: :CSOLoad/CSOLoad.cpp: Line 155: Message Could not open file.
2016-04-07 13:33:45 CSO Error: :CSOLoad/CSOLoad.cpp: Line 155: Message Could not open file.
2016-04-07 13:56:37 CSO Error: :CSOLoad/CSOLoad.cpp: Line 155: Message Could not open file.
2016-04-07 13:57:07 CSO Error: :CSOSave/CSOSave.cpp: Line 163: Message Could not save CSOList because std::
2016-04-07 13:57:08 CSO Error: :CSOSave/CSOSave.cpp: Line 163: Message Could not save CSOList because std::
2016-04-07 13:57:31 CSO Cout: CSOConvertTo3DMask: could not find a proper starting cell for scanning the voxel surface
2016-04-07 13:57:31 CSO Cout: Reason: all voxel positions around the candidate starting cell are outside the image.
2016-04-07 13:57:31 CSO Cout: CSOConvertTo3DMask: could not find a proper starting cell for scanning the voxel surface
2016-04-07 13:57:31 CSO Cout: Reason: all voxel positions around the candidate starting cell are outside the image.
2016-04-07 14:00:10 Warning: WindowInfo(90) Set the panel of [SoExaminerViewer5] to the default position
2016-04-07 14:00:19 Warning: WindowInfo(90) Set the panel of [SoExaminerViewer4] to the default position
2016-04-08 23:49:42 Warning: WindowInfo(90) Set the panel of [SoExaminerViewer5] to the default position
2016-04-08 23:53:37 Warning: WindowInfo(90) Set the panel of [RunPythonScript] to the default position be
```

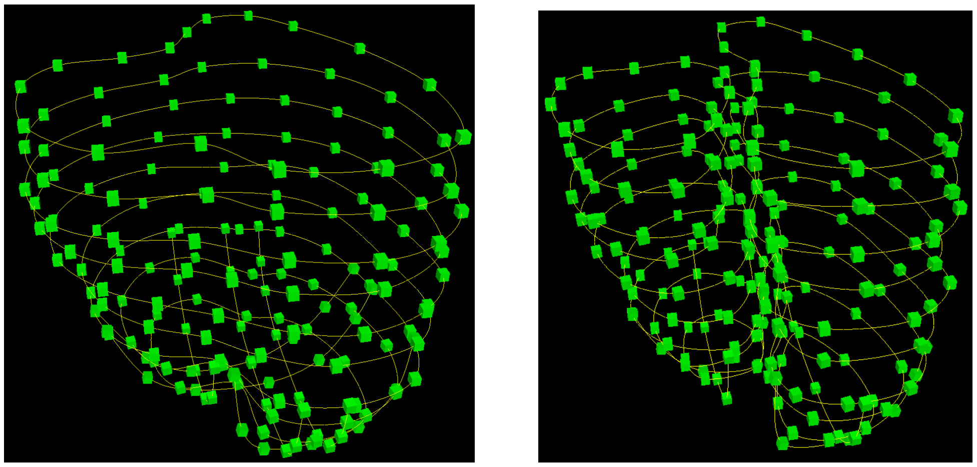
Figure 2.10: Self-developed Python coding module for coordinates transformation for CSO

Overall, biventricular endocardium and epicardium were contoured both at end-systole and end-diastole of pre-surgical and post-surgical images, followed by the transformation of coordinates. All contours are depicted in Figure 2.11.



(a) Pre-surgical end-systolic epicardial (left) and endocardial (right) contours


(b) Pre-surgical end-diastolic epicardial (left) and endocardial (right) contours


(c) Post-surgical end-systolic epicardial (left) and endocardial (right) contours

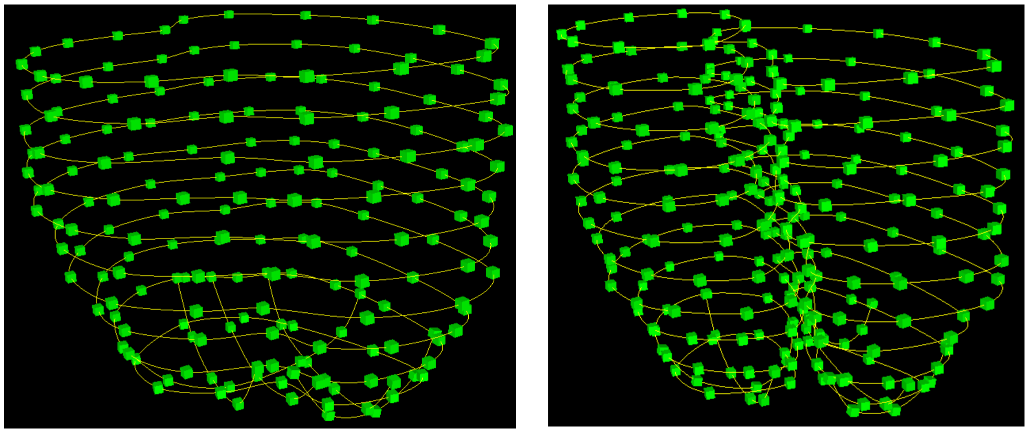

(d) Post-surgical end-diastolic epicardial (left) and endocardial (right) contours

Figure 2.11: Biventricular endocardial and epicardial contours at ES and ED in the global coordinate system

2.2.3 3D Modeling and Visualization

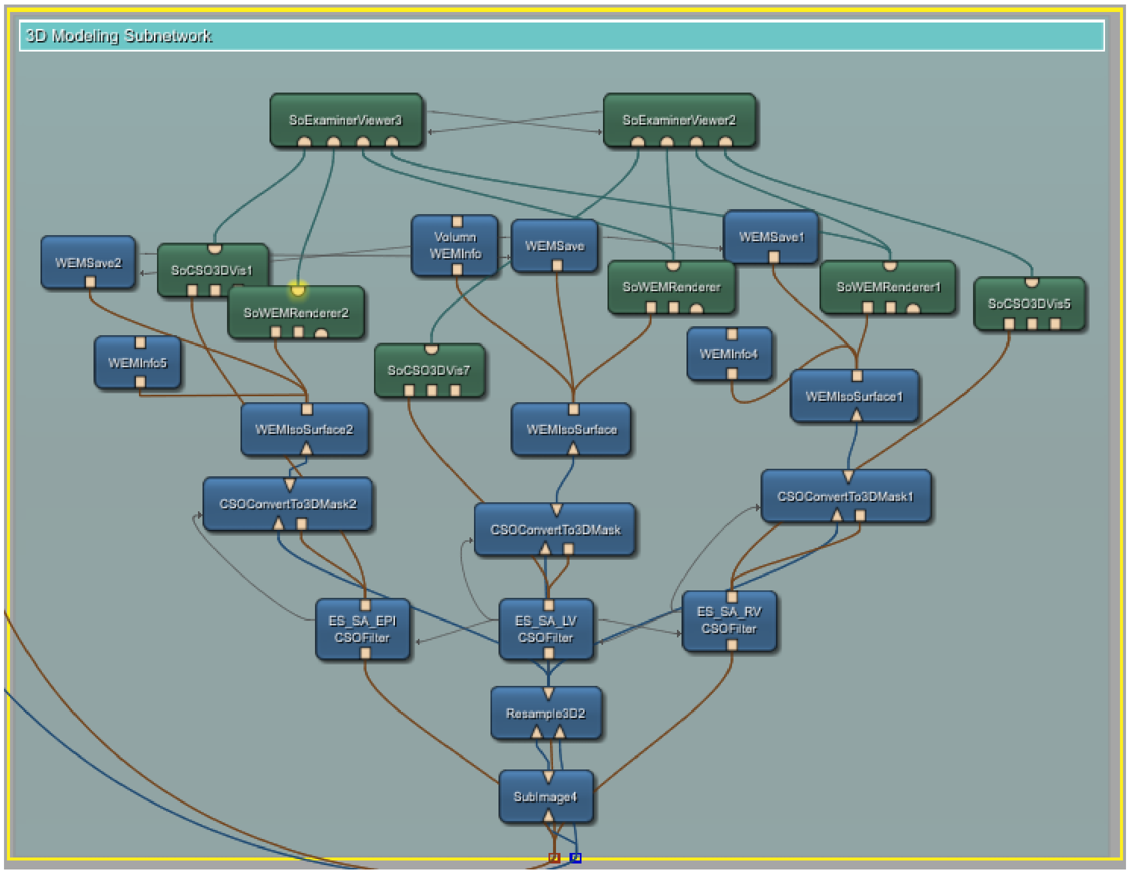
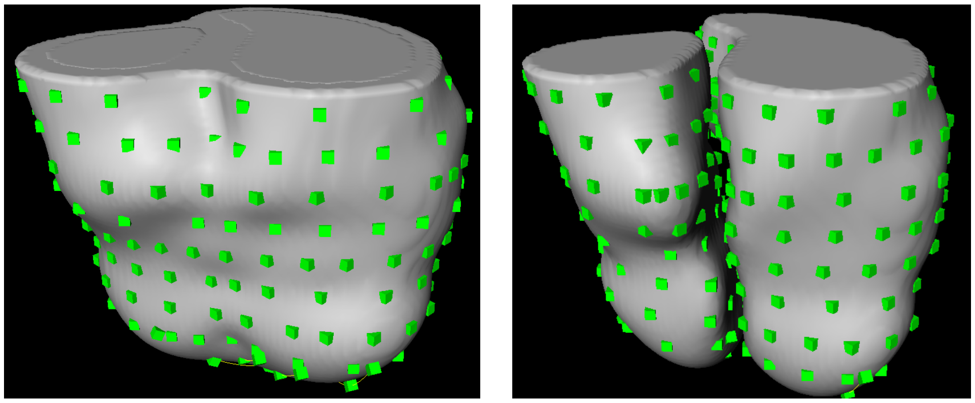
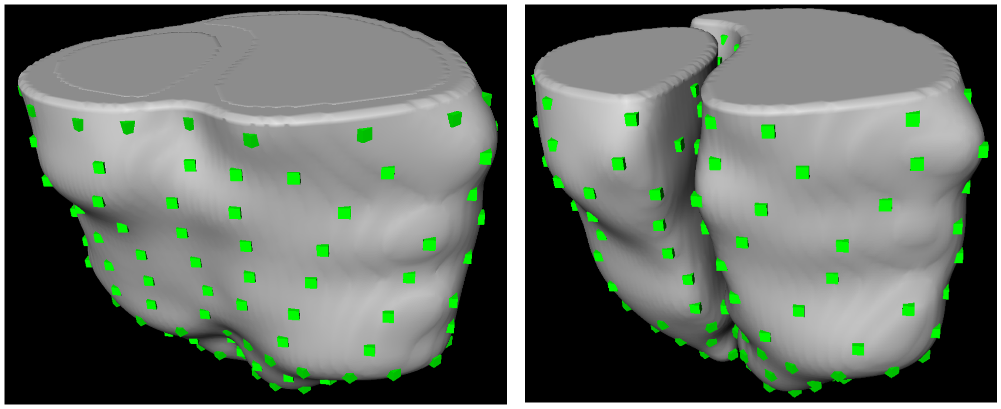
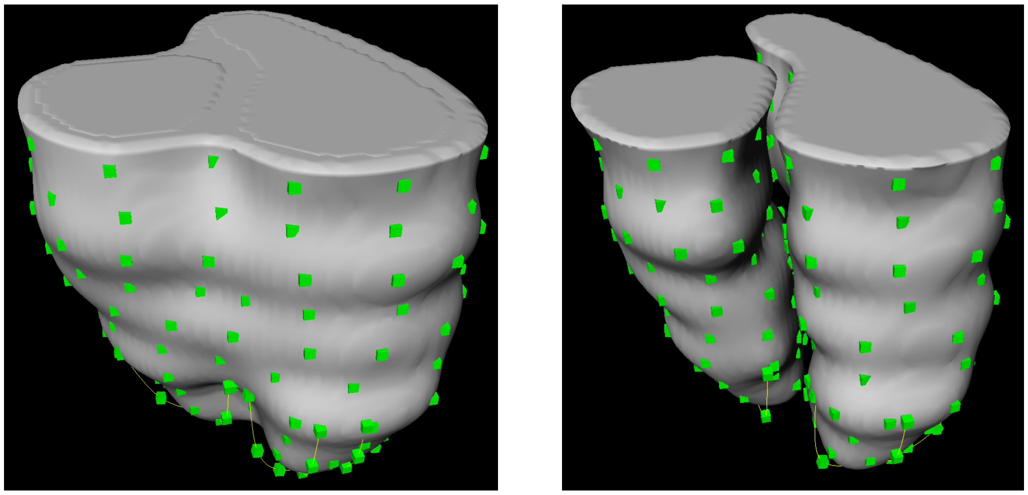


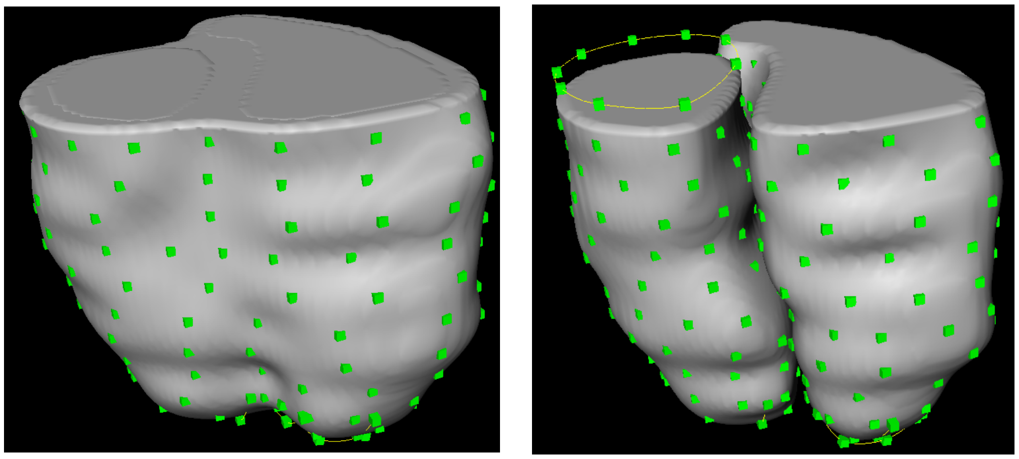
Figure 2.12: 3D modeling application

Furthermore, the 3D epicardial and endocardial surfaces were reconstructed in 3 dimensions by being fitted into the biventricular contours in the global coordinate system. Likewise, MeVisLab provides many built-in modules for manipulating Winged Edge Mesh (WEM), thus all surfaces were stored with WEM. An application for 3D modeling is shown in Figure 2.12. First, the built-in module CSOConvertTo3DMask fitted a continuous model to CSO contours from multiple slices, followed by refining and smoothing. Afterward, the built-in module WEMIsoSurface converted the model into a WEM. Last, WEM was rendered via the built-in module SoWEMRenderer. 3D surfaces of epicardium and endocardium are depicted in Figure 2.13.


(a) Pre-surgical end-systolic epicardium (left) and endocardium (right)


(b) Pre-surgical end-diastolic epicardium (left) and endocardium (right)


(c) Post-surgical end-systolic epicardium (left) and endocardium (right)

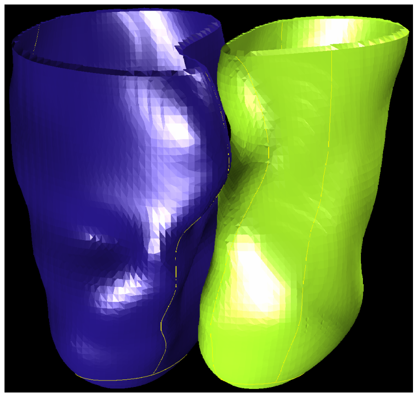


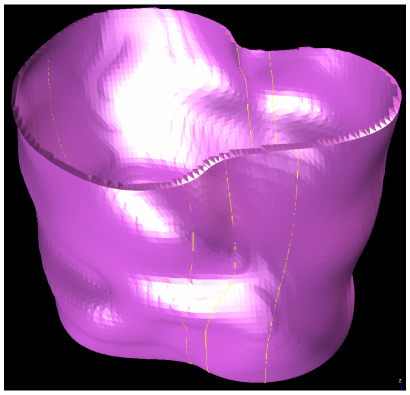
(d) Post-surgical end-diastolic epicardium (left) and endocardium (right)

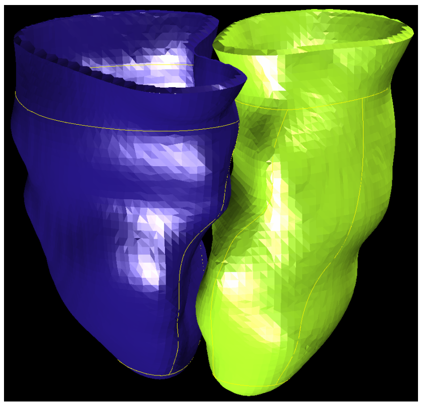
Figure 2.13: Biventricular epicardium and endocardium surfaces with their corresponding contours at ES and ED

2.2.4 Finite Element Meshes

End-systolic epicardial and endocardial surfaces were utilized to construct pre-surgical and post-surgical biventricular meshes with Truegrid (XYZ Scientific Applications, Inc., Livermore, CA). The spaces between the epicardial and endocardial surfaces were filled with 8-noded brick elements. Since RV inserting locations were twisting along the longitudinal direction, surface geometries here were pointy. In order to replicate the complicated in-vivo geometries here, 3D splines were manually drawn on the surfaces (Figure 2.14), followed by the alignment of element edges along the splines. Subsequent volumetric meshes were refined by adding more elements in the circumferential direction. Considering the fact that the septum is thicker than the lateral wall, four elements in the septum and three elements in the lateral wall were deployed transmurally. In addition, in order to make elements evenly spaced, some nodes were manually moved, followed by the execution of smoothing commands. A layer of shell elements was used to line the endocardial surface of both the LV and RV, in order to form an enclosed volume for ventricular cavity volume measurements (Figure 2.15). The result of meshes is shown in Figure 2.16, for a total of 23,752 elements for the pre-surgical case and 24,778 elements for the post-surgical case.


(a) Pre-surgical endocardial surfaces


(b) Pre-surgical epicardial surface


(c) Post-surgical endocardial surfaces

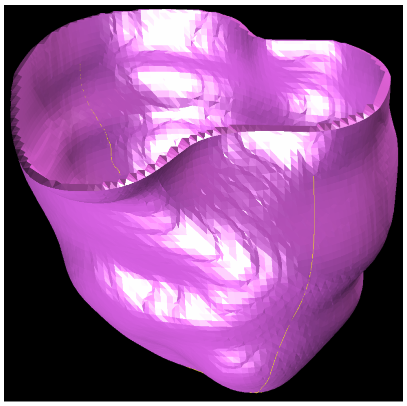
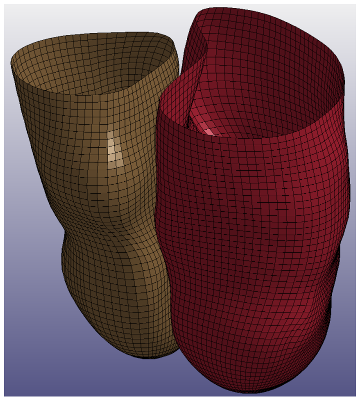

(d) Post-surgical epicardial surface

Figure 2.14: Freehand contours replicating complex geometries


(a) Pre-surgical endocardial mesh

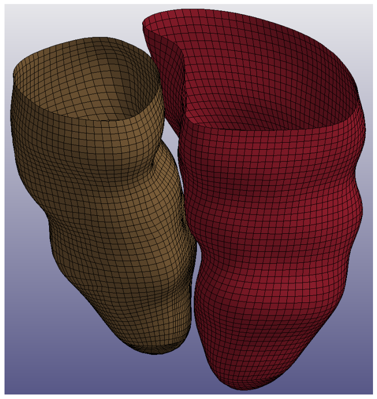

(b) Post-surgical endocardial mesh

Figure 2.15: Endocardial Meshes for ventricular cavity volume evaluation

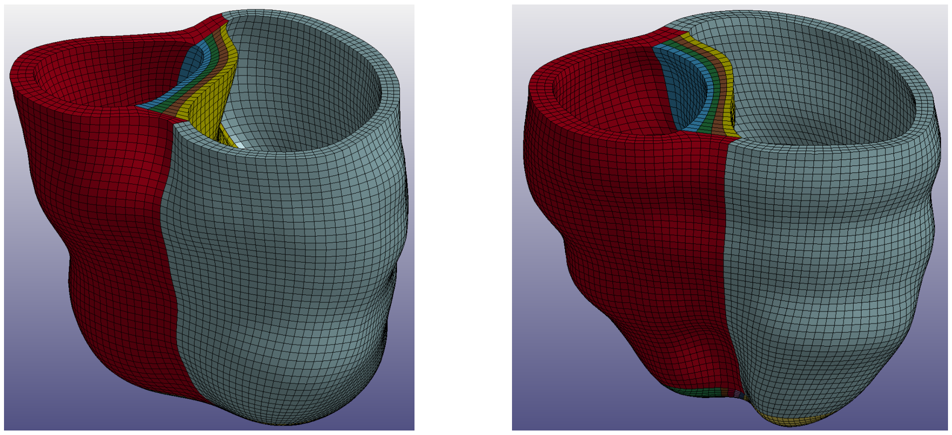
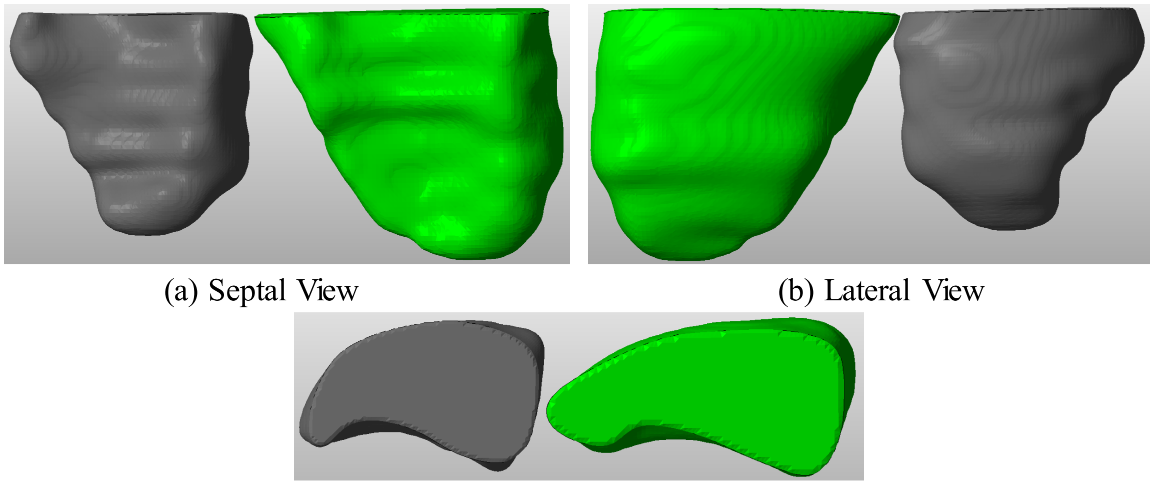

(a) Pre-surgical biventricular mesh (b) Post-surgical biventricular mesh

Figure 2.16: Biventricular meshes created with Truegrid

2.3 Results and Discussion


(a) Septal View (b) Lateral View

(c) Top View

Figure 2.17: Comparison between pre-surgical (grey) and post-surgical (green) RV cavities at end diastole

First, the patient still had a severe problem of dilated RV after the surgery. The post-surgical right ventricular end-diastolic volume (RVEDV) of blood was 186 ml, while the normal range is between 100 and 160 mL [18]. What's worse, the post-surgical RVEDV (186 ml) was much larger than pre-surgical RVEDV (134 ml) (Figure 2.17a and Figure 2.17b). Figure 2.17c also shows that the cross-sectional area of basal plane is larger after the surgery, compared with that before the surgery.

Another index used for evaluating the functionality of RV is right ventricular ejection fraction (RVEF). RVEF is the ratio between right ventricular stroke volume (RVSV) and RVEDV, where RVSV is the volume of blood pumped out by RV per beat. As a result, the post-surgical RVEF is 34.33%, compared with 40%-60% in normal subjects [18]. Thus, the RV still functioned abnormally after the surgery.

2.4 Conclusion

Overall, a contouring technique was developed with MeVisLab. A methodology of 3D modeling and visualization of 3D contours was proposed. Biventricular surfaces in repaired TOF were modeled before and after the pulmonary valve replacement surgery. Right ventricular end-diastolic volume and ejection fraction were computed.

Unfortunately, the patient still suffered severe RV dilation. When two post-surgical indices were compared with those of normal subjects, the dysfunction of RV in the patient was still indicated. Therefore, the pulmonary valve replacement failed to reduce pulmonary regurgitation or decreases RV dilation for the patient.

3 Quantification of 3D Lagrangian Strain and Torsion in Rat Left Ventricles with DENSE MRI

3.1 Background

Displacement ENcoding with Stimulated Echoes (DENSE) is an MRI tissue tracking technique which allows the analysis of myocardial deformation.

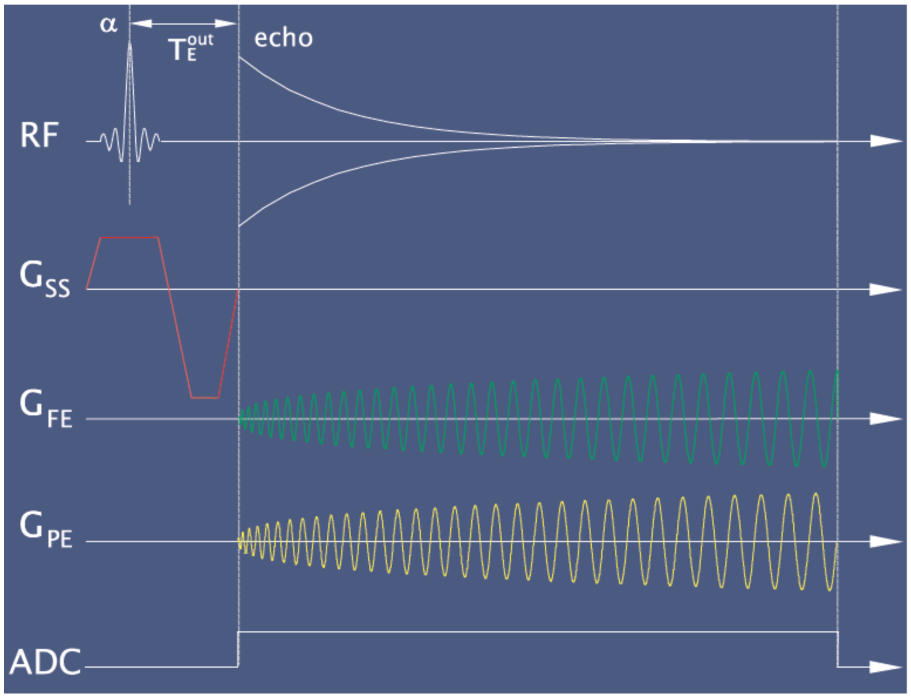
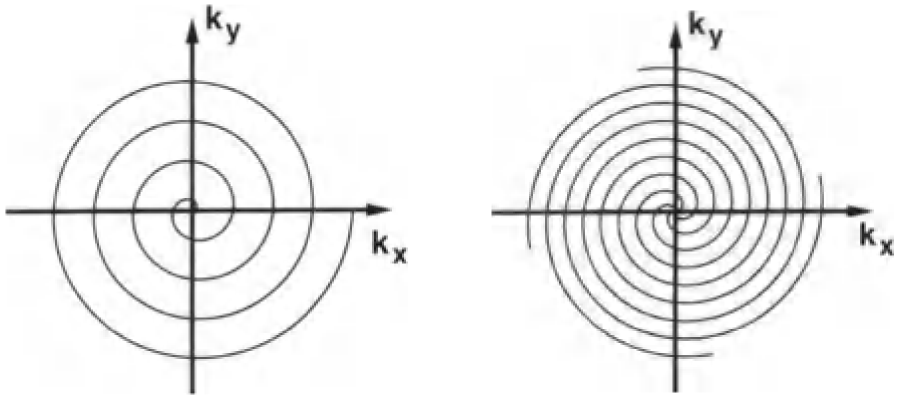


Figure 3.1: Timing diagram for the 3D cine DENSE spiral pulse sequence [19]

A diagram of DENSE acquisition protocol of spiral pulse sequence is depicted in Figure 3.1. Once triggered by the R wave of the ECG, an initial R_f excitation is emitted, followed by two k-cycle sinusoidal readout gradients G_{FE} and G_{PE} in two orthogonal directions of a plane. G_{FE} and G_{PE} oscillate with a slowly increasing amplitude until they reach the amplitude limit. Assuming a single effective readout gradient G_{RO} is the vector sum of G_{FE} and G_{PE}, G_{RO} rotates as G_{FE} and G_{PE} change their directions and spirals out as G_{FE} and G_{PE} increase their amplitudes. As a result, G_{RO} proceeds along a k-space trajectory on the plane. As shown in Figure 3.2a, a single k-space trajectory is completed in a single readout window. It's a Archimedean spiral which can be constructed by running a angle θ as a function of the radius of a specific trajectory point. Figure 3.2b shows four interleaved k-space trajectories which are conducted during four readout windows. Compared with a single-shot spiral scan, an interleaved spiral scan can bring more signal for the same spatial resolution leading to less blurring; but it also take longer time to complete. As G_{RO} spirals out, phase dispersion is introduced across tissue in a direction perpendicular to G_{RO}. Hence, phase dispersion also proceeds along k-space trajectories on the plane. Consequently, echo of radiation due to a magnetic moment change is recorded from the center to the periphery along those trajectories until a circular field of view (FOV) is fully covered. Here, G_{RO} is assumed to have a magnitude G_R G/cm and a duration time t_{enc} (period of a readout window).



(a) Single k-space spiral (b) Four interleaved spiral

Figure 3.2: Diagram of k-space trajectories covered over a field-of-view [20]

Before a second R_f excitation is applied, a tissue displacement of Δx occurs during a time period of TR. After the second R_f excitation, a second effective readout gradient G_{RO} causes another phase dispersion. For stationary spins, the phase dispersion is complete. For protonic spins which have moved a distance of Δx, a net phase φ_1 of the echo signal is accumulated, which is illustrated by Equation (3.1):

$$\varphi_1 = \gamma_H G_R t_{enc} \Delta x \tag{3.1}$$

where γ_H is the gyromagnetic ratio in Hz/Tesla.

Imaging with slice selection is performed by a series of n-frame sampling per R_f excitation with TR as repetition time. A scan is acquired at a cardiac sycle.

In order to obtain displacements in the FOV, the first scan serves as the phase reference data set. Next, a complementary data set is acquired through repeating the sequence once more with altered amplitude G_R' of effective readout gradient. The difference in effective readout gradients is set for a specific value of displacement encoding. Since the amplitude of effective readout gradient changes, accumulated phase φ_2 is different, which is illustrated by Equation (3.2). Illustrated by Equation (3.3), the phase shift $\Delta\varphi$ between two sequences is used to compute Δx. Although the difference of effective readout gradient is small, TR can be set to be large enough when Δx occurs so that a noticeable phase difference $\Delta\varphi$ can be detected. Accordingly, DENSE has a high spatial resolution [21]. When the R_f excitation frequency is fixed between two scans, the acquisition location in the complementary scan slightly shifts from that in the phase reference scan due to the tiny difference in effective readout gradients.

$$\varphi_2 = \gamma_H G_R^{'} t_{enc} \Delta x \tag{3.2}$$

$$\Delta\varphi = \gamma_H (G_R - G_R^{'}) t_{enc} \Delta x \tag{3.3}$$

A balanced four-point encoding strategy is adopted to acquire displacements in three directions, which can offer homogeneous phase noise in three directions and increase phase SNR as suggested by Zhong and colleagues [22]. As shown in Figure 3.3, in order to get the 3D displacements in the centroid of a regular tetrahedron, a phase reference scan and a complementary scan are performed on the four vertices of the tetrahedron, respectively, contributing to four sets of phase differences. Later, weighting vectors are used to derive x, y, and z displacements from phase differences. At last, tissue displacement values are encoded directly into the intensity of each pixel of phase images along each direction. Consequently, a total of eight cine data sets are acquired for encoding phase images along three directions, presenting a map of 3D displacement of tissue on a specific slice.

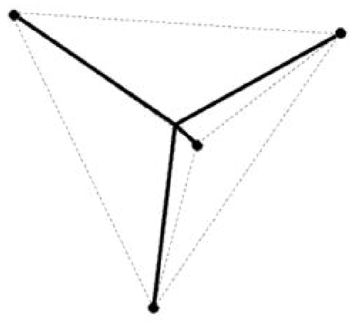


Figure 3.3: Four weighting vectors for 3D balanced multi-point encoding method [22]

Magnitude images are averaged over the phase reference scans. Depicted in Figure 3.1, due to the nonsynchronous motion of the blood in the ventricle, black blood images are produced. This type of image contrast is desirable in our cardiac MRI studies. First, black blood can exhibit a contrast between myocardium and ventricular cavity, help us to separating them. Besides, without bright blood signal, motion artifacts can be reduced along the phase direction (shown in Figure 3.4).

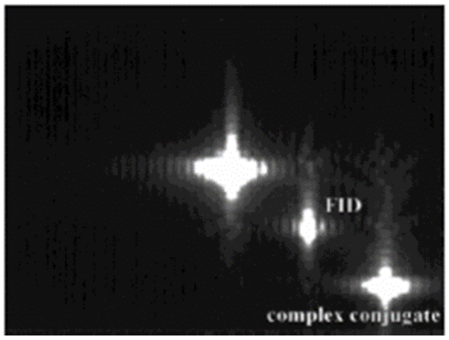


Figure 3.4: Striping Artifacts caused by bright blood signal [23]

3.2 Introduction

Several previous studies have investigated Lagrangian strain in the mouse LV. Fornwalt *et al.* [24] investigated circumferential, longitudinal, and radial strains with DENSE. Variations in systolic and diastolic strain rates across five normal mice were reported transmurally. They also reported basal, mid-ventricular, and apical torsion and ventricular synchrony with CURE and RURE indices. Zhong and Yu [25] investigated 2D strain and torsion using DENSE. They evenly divided each of apical and basal slices into four circumferential segments. Peak systolic radial and circumferential strains of each segment were compared at baseline and under dobutamine (Dob) stimulation. Torsion was reported over the cardiac cycle at baseline and under Dob stimulation. Li and Yu [26] also studied radial strain, circumferential strain, and torsion at baseline and under Dob stimulation. But they adopted a harmonic phase (HARP) based method of MRI tagging.

In addition, some researchers were interested in rat LV. Stelzer *et al.* [27] investigated 2D strain and torsion with DENSE. End-systolic maximal principal strain E_1 and minimal principal strain E_2 at the apex and base were compared at baseline and under Dob stimulation. Torsion in hypothyroid rats was compared with control rats over the cardiac cycle. Li and Yu [28] studied two principal Lagrangian strains (E_1 and E_2) in apex, mid-ventricle, and base at diastole. They compared 2D myocardial strain measured by HARP with those obtained by conventional homogeneous strain analysis based on manual tracking of tag lines.

Additionally, Hess *et al.* [29] focused on human LV and constructed 3D strain maps of the LV of six volunteers using cine DENSE and cine strain encoding (SENC) MRI. They divided the heart with the 17-segment model of AHA. Averaged principal mid-ventricular strain of each segment was compared in three directions over the cardiac cycle.

3.3 Aims

Surprisingly, it appears that no other research groups have segmented a slice of the LV circumferentially and transmurally. Inspired by the work of Fornwalt *et al.* [24], the goal of the study is to apply a more detailed segmentation to the rat mid-ventricle and quantify six components of Lagrangian strain in polar orientation in each segment. Torsion of the rat LV is also computed. This work also aims to provide tools for rapid Lagrangian strain analysis. We hope strain data of segments can provide other researchers with useful reference.

3.4 Methods

A schematic of the methodology of quantification of 3D Lagrangian Strain and Torsion in LV is summarized in Figure 3.5.

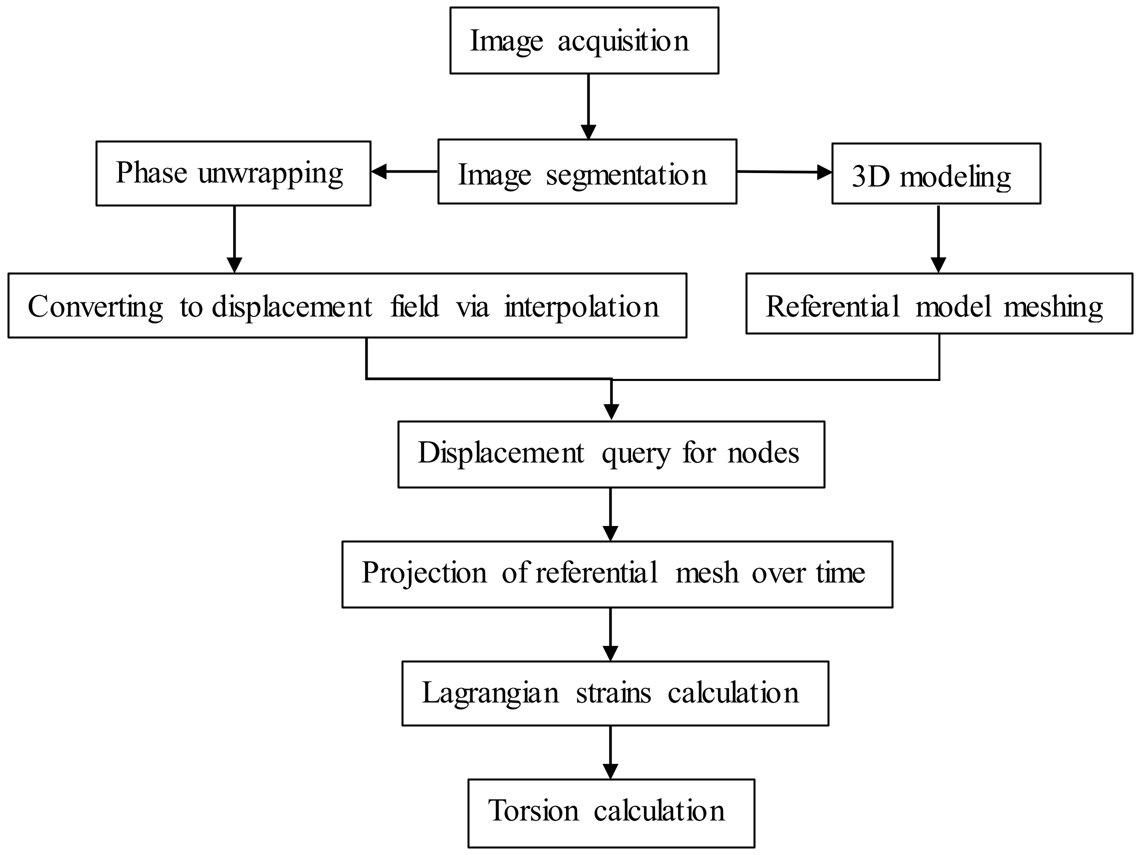


Image acquisition
Phase unwrapping
Image segmentation
3D modeling
Converting to displacement field via interpolation
Referential model meshing
Displacement query for nodes
Projection of referential mesh over time
Lagrangian strains calculation
Torsion calculation

Figure 3.5: Schematic of quantification of 3D Lagrangian strain and torsion in LV

3.4.1 Rat Models

Ten healthy rats were contained inside ventilated cages in a temperature-controlled room. They were provided with acrylic huts and nesting material. Their ages ranged from 5 to 7 months when they were scanned. All procedures implemented on the rats were conducted by Dr. Xiaoyan Zhang. The procedures complied with Public Health Service policies for humane care and use of animals. The procedures also conformed to the protocols approved by the institutional animal care and use committee at the University of Kentucky.

3.4.2 Animal preparation

Animal preparation was conducted by Dr. Xiaoyan Zhang. Before being imaged, the rats were anesthetized with isoflurane. A vaporizer was used to precisely delivering 2.5% isoflurane in oxygen at a rate of 1.5 L/min. Once rats didn't have reflexes, three legs were shaved and ECG electrodes required for cardiac gating were attached. In order to reduce motion artifacts, a diaphragm of respiratory sensor was placed under the abdomen for respiratory gating. The animals were wrapped by a heated pad for the maintenance of body temperature between 36±1 degrees Celsius. A rectal thermometer was inserted to monitor body temperature. Consequently, heart rate, respiratory rate, and body temperature were continuously monitored during scanning with a fiber optic acquisition system (SA Instruments, Inc, Stony Brook, NY).

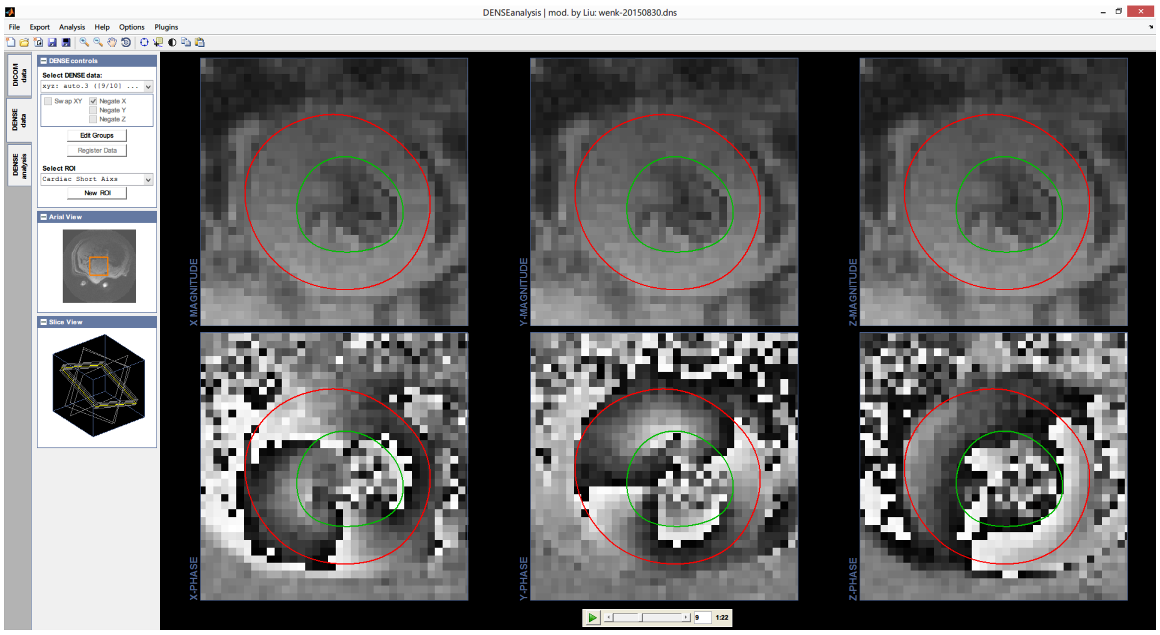
3.4.3 DENSE MRI Acquisition

DENSE MRI acquisition was also conducted by Dr. Xiaoyan Zhang. DENSE MRI was performed on a 7-Tesla BrukerClinScan system (Bruker, Ettlingen, Germany). Immediately after a detection of the peak of the ECG's R-wave during a stable respiratory period, DENSE sequence was emitted. A magnitude image was constructed and three phase images were independently encoded for displacements in x, y and z directions, respectively (Figure 3.1). A total of 17–22 frames per cardiac cycle were collected with a repetition time of 7.4 ms and an echo time of 1 ms. Pixel spacing was 0.357 mm and slice thickness was 1.3 mm, leading to a voxel size of $0.357 \times 0.357 \times 1.3$ mm. Since an image is a matrix of 140×140 pixels, field of view was about 50×50 mm, which fully covers two whole ventricles from both SA views and LA views. After multiple test scans, the displacement encoding frequency was set for a constant value of 0.3 cycles/mm, in which images have a minimal signal-to-noise ratio (SNR).

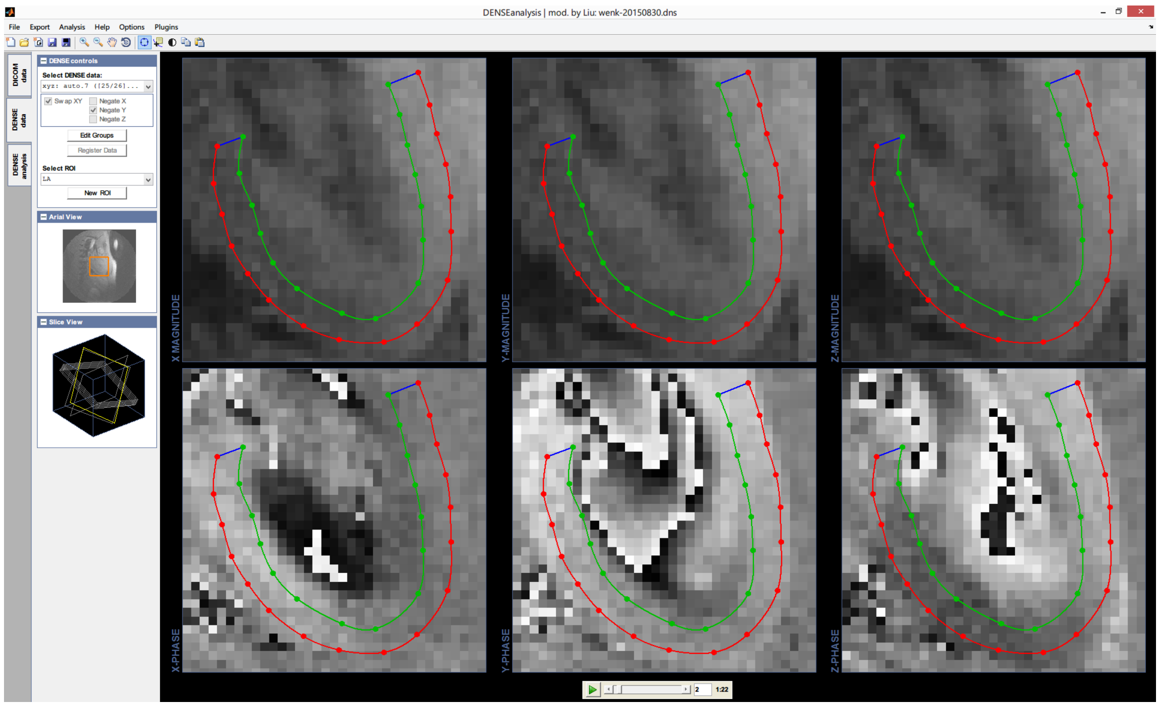
Depend on the ventricular size, 5-6 short-axis slices and 2 long-axis slices were set up for each rat. The long-axis slices were aligned on the 4-chamber view and the 2-chamber view. The short-axis slices were set up perpendicular to the long-axis slices. Specifically, the basal slice was placed in the position where the septum can just be entirely visualized at end systole, while the apical slice was positioned in the location where the cavity can just be visualized at end systole. Consequently, approximately 80% of end-systolic ventricular length was covered by short-axis images.

3.4.4 Image Segmentation

A software DENSEanalysis developed in MATLAB by Dr. Andrew D. Gilliam was used to draw contours of endocardium and epicardium of left ventricle on both short-axis and long-axis images (Figure 3.6). Blood pool and surrounding tissue were recognized from magnitude images and noisy phase data. Endocardial and epicardial boundaries were traced to identify the myocardium in all frames and slices.



(a) Short-axis contours



(a) Long-axis contours

Figure 3.6: Left ventricular contours at end systole on mid-ventricular short-axis images taken on Aug. 30, 2015

First of all, myocardial contours need to be modified by distinguishing myocardium from papillary muscle. Because papillary muscles (2 protrusions in Figure 3.7) were aligned from the mid-level up to the base (Figure 3.8), a segment of endocardial contours may be wrongly placed on them from short-axis views. If papillary muscles are inside myocardial contours, they could bring a tremendous change in wall thickness from slice to slice (Figure 3.8a). Besides, since papillary muscles are pulling atrioventricular valves during diastole, they would be hard to be tracked. Thus, papillary muscles need to be excluded from myocardial contours.

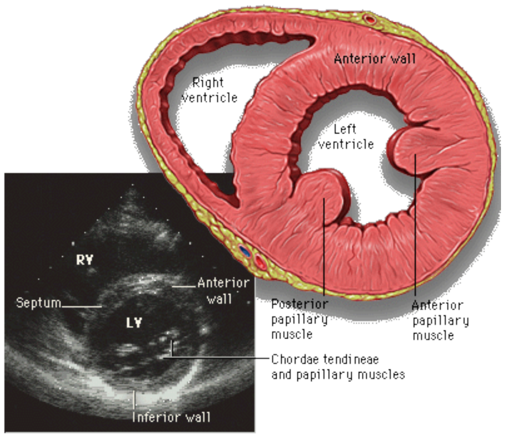
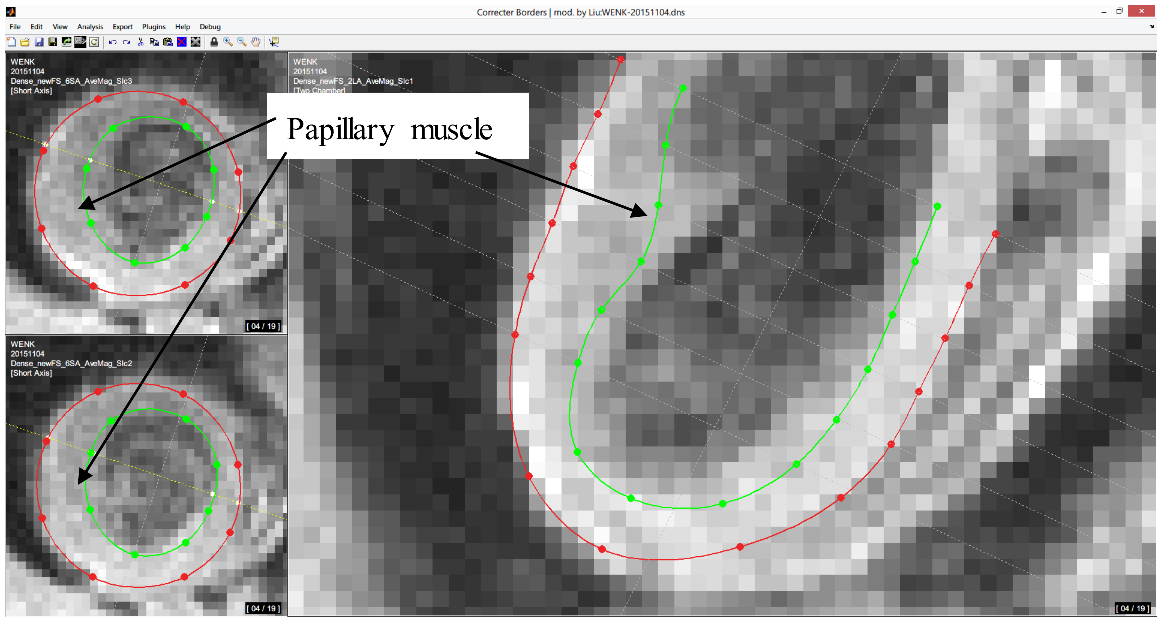
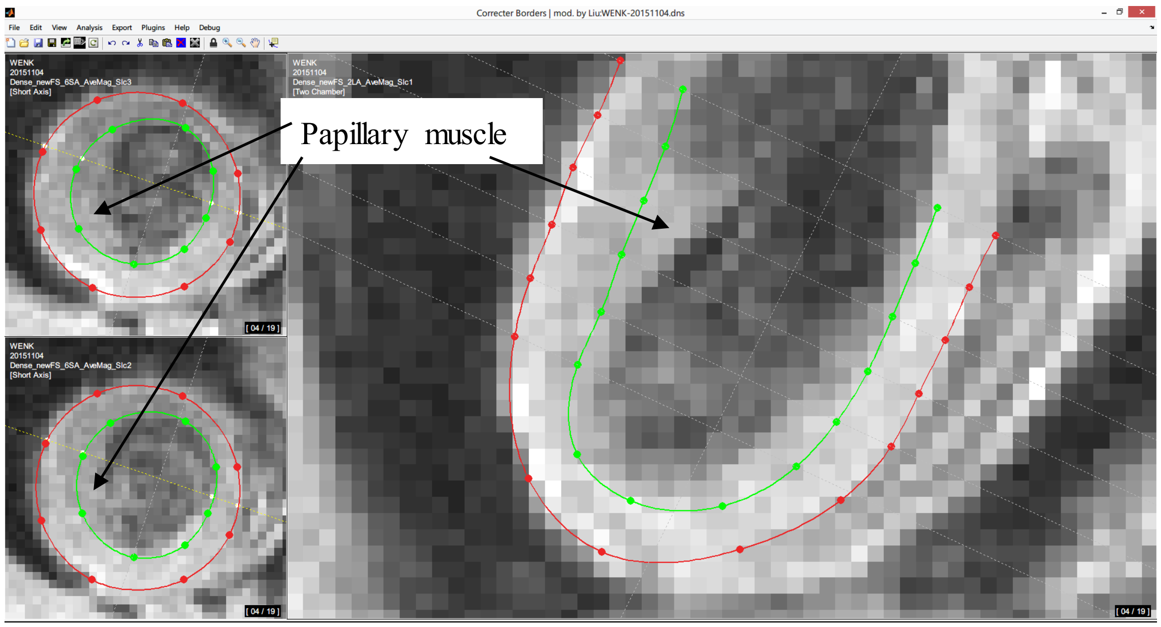

Anterior wall
Right ventricle
Left ventricle
RV
Septum
Anterior wall
LV
Posterior papillary muscle
Anterior papillary muscle
Chordae tendineae and papillary muscles
Inferior wall

Figure 3.7: Papillary muscles in human heart [30]

Fortunately, papillary muscles in rat heart were large enough to be visualized at end systole from long-axis view. Thus, short-axis contours need to be displayed on long-axis images to verify they don't cover papillary muscles. A software CorrecterBorders developed in MATLAB by Dr. Jonathan Suever can display 3D freehand contours simultaneously on both short-axis and long-axis CMR images of DICOM standard. In order to draw contours with DENSEanalysis and CorrecterBorders, two custom modules (Appendix C and Appendix D) were written in MATLAB (Mathworks, Inc., Natick, MA). After associating short-axis images with long-axis images based on locations and orientations of all slices, 2D contours created with DENSEanalysis were converted to 3D contours read by CorrecterBorders. As shown in Figure 3.8a, papillary muscle was hard to tell from myocardioum in a short-axis view but well-marked from a long-axis view. After modifications of short-axis contours (Figure 3.7b) were conducted with CorrecterBorders, the contours were imported back to DENSEanalysis for further adjustments.



(a) Before adjustment



(b) After adjustment

Figure 3.8: Exclusion of papillary muscle from myocardial contours

Furthermore, myocardial contours also need to be adjusted to confront the following principles of ventricular shapes changing in a particular cardiac stage. During isovolumetric contraction or isovolumetric relaxation, myocardial contours are supposed to have same size. Myocardial contours are also supposed to shrink with increasing wall thickness during systole, but enlarge with decreasing wall thickness during diastole.

3.4.5 Phase Unwrapping

As illustrated in Chapter 3.1, Lagrangian displacements indicating where voxels came from with respect to their referential location are encoded in the contrast of phase images. The contrast value of 0-4098 are corresponding to a phase range from -π to π. Depicted in Figure 3.9, since 0.3 cycles/mm ($3\pi/5\ rad$/mm) is selected displacement encoding frequency, displacements of -10/3, 0, 10/3 mm have phase values of -2π, 0, 2π but stored with a same contrast value. Consequently, tissue with the above displacements appear the same from phase images. In other words, displacements of -10/3 and 10/3 mm are wrapped into the displacement of 0 mm and thus need to be restored back to actual values. Only a displacement in the range of -5/3-5/3 mm is unwrapped.

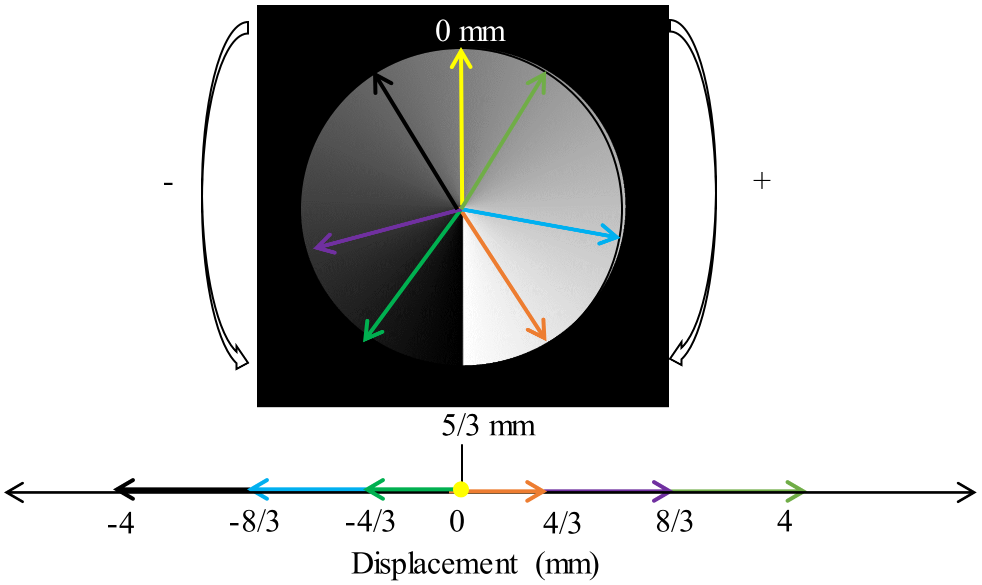
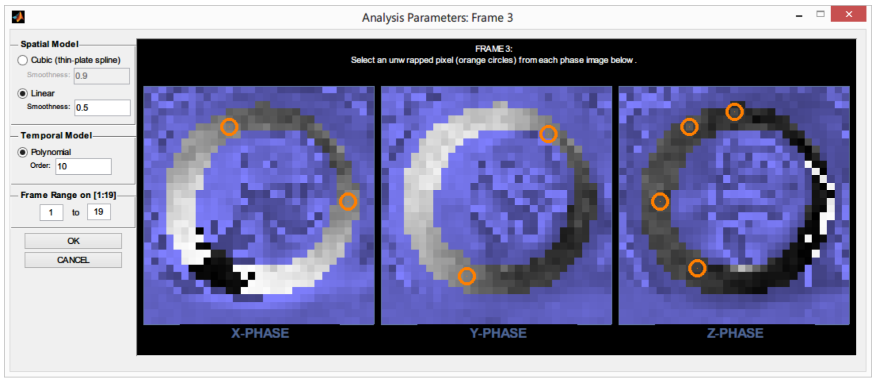


Figure 3.9: Schematic of phase and corresponding displacement when encoding frequency is 0.3 cyc/mm

With DENSEanalysis, first, drawn contours were used to extract myocardial pixels (non-blue pixels in Figure 3.10) from the pixels belong to blood and surrounding tissue (blue pixels in Figure 3.10) in each frame. Then, representative pixels encoded with unwrapped displacement were manually picked out from phase images for all slices (Figure 3.10). Last, an algorithm proposed by Spottiswoode [33] was used for phase unwrapping. Consequently, actual displacements of all myocardial voxels in each frame were obtained. They were relative to initial location of the voxels in the referential frame when the encoding occurred (Figure 3.1).


Analysis Parameters: Frame 3

Spatial Model
- ( ) Cubic (thin-plate spline) — Smoothness: 0.9
- (●) Linear — Smoothness: 0.5

Temporal Model
- (●) Polynomial — Order: 10

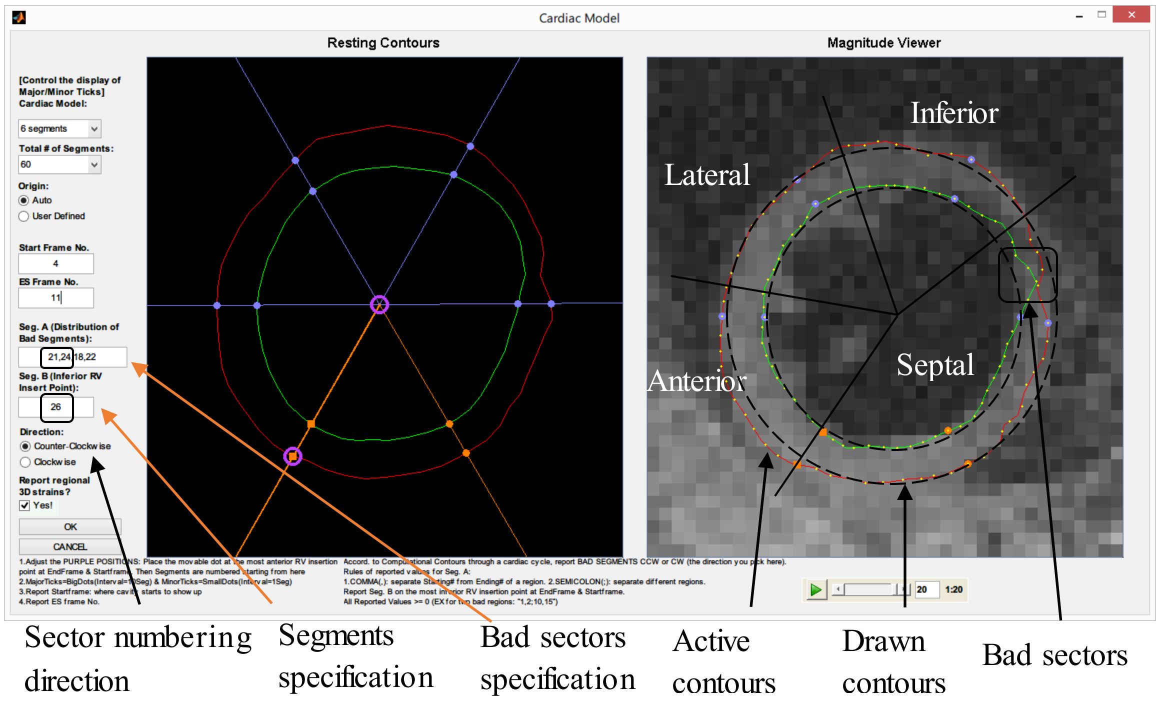
Frame Range on [1:19]: 1 to 19

FRAME 3:
Select an unwrapped pixel (orange circles) from each phase image below.

Figure 3.10: Manual selection of unwrapped pixels in DENSEanalysis

3.4.6 Discretization of the LV in the Referential Frame

A module of generating a cardiac model is provided in DENSEanalysis. A cardiac model of a specific slice consists of resting contours (left viewer in Figure 3.11) in the referential frame and computational contours (right viewer in Figure 3.11) in each imaging frame. First, resting contours were obtained via projecting the myocardial voxels in the unwrapped-pixels-located frame (Figure 3.10) back to the referential frame using the corresponding displacements. Next, computational contours were acquired via projecting the resting contours forward to each frame using the corresponding myocardial displacements. As shown in Figure 3.11, displacements in the frame 20 were used to move the resting contours to frame 20.


Cardiac Model

[Control the display of Major/Minor Ticks] Cardiac Model: 6 segments

Total # of Segments: 60

Origin: (●) Auto ( ) User Defined

Start Frame No.: 4

ES Frame No.: 11

Seg. A (Distribution of Bad Segments): 21,24;18,22

Seg. B (Inferior RV Insert Point): 26

Direction: (●) Counter-Clockwise ( ) Clockwise

Report regional 3D strains? ✓ Yes!

1.Adjust the PURPLE POSITIONS: Place the movable dot at the most anterior RV insertion point at EndFrame & Startframe. Then Segments are numbered starting from here
2.MajorTicks=BigDots(Interval=10Seg) & MinorTicks=SmallDots(Interval=1Seg)
3.Report Startframe: where cavity starts to show up
4.Report ES frame No.

Accord. to Computational Contours through a cardiac cycle, report BAD SEGMENTS CCW or CW (the direction you pick here).
Rules of reported values for Seg. A:
1.COMMA(,): separate Starting# from Ending# of a region. 2.SEMICOLON(;): separate different regions.
Report Seg. B on the most inferior RV insertion point at EndFrame & Startframe.
All Reported Values >= 0 (EX for two bad regions: "1,2;10,15")

Figure 3.11: Meshing user interface for a short-axis slice

The module was subsequently improved in MATLAB by adding a function of meshing the LV in the referential frame (Appendix E, Appendix G, Appendix H and Appendix I). The first step of left ventricular discretization was to set up mesh parameters and generate a 2D referential mesh for each SA slice. As shown in Figure 3.11, a custom mesh configuration interface was created with graphical user interface design environment (GUIDE) of MATLAB (Appendix E). Computational contours were set for a circumferential partition of 60 sectors numbered counterclockwise. Four nearby yellow dots constructed a sector. Two insertion points of the RV wall to the LV separate myocardium into septum and free wall. The first sector was placed on the anterior RV insertion point (the square dot in Figure 3.11). The sector number on the inferior RV insertion point was also specified. The sectors between these two sectors were classified into septal segment, while the rest of the sectors were divided into equal thirds which were classified as inferior, lateral, and anterior segment, respectively. The anterior segment lies by the side which is closer to the chest. Last but not least, bad sectors were marked. In Figure 3.11, computational end-diastolic contours have abnormal wall thickness from sector 21 to sector 24, resulting in enormous magnitude of negative strain. Therefore, it's necessary to take bad sectors out of play. On the other hand, bad sectors might be repaired by trying different unwrapped pixels from different frames.

For each rat left ventricle, once the mesh configuration was done for all short-axis slices, a referential volumetric mesh was built up by stacking 60 sectors of resting contours slice by slice so that the mesh was filled with 8-noded brick elements. There are 60 elements in the circumferential direction and 3 elements in the transmural direction (Figure 3.12). The number of elements in longitudinal direction was less than the number of SA slices by one. Segmentation attribution of each sector was transferred to the corresponding elements. The elements corresponding to bad sectors were also marked.

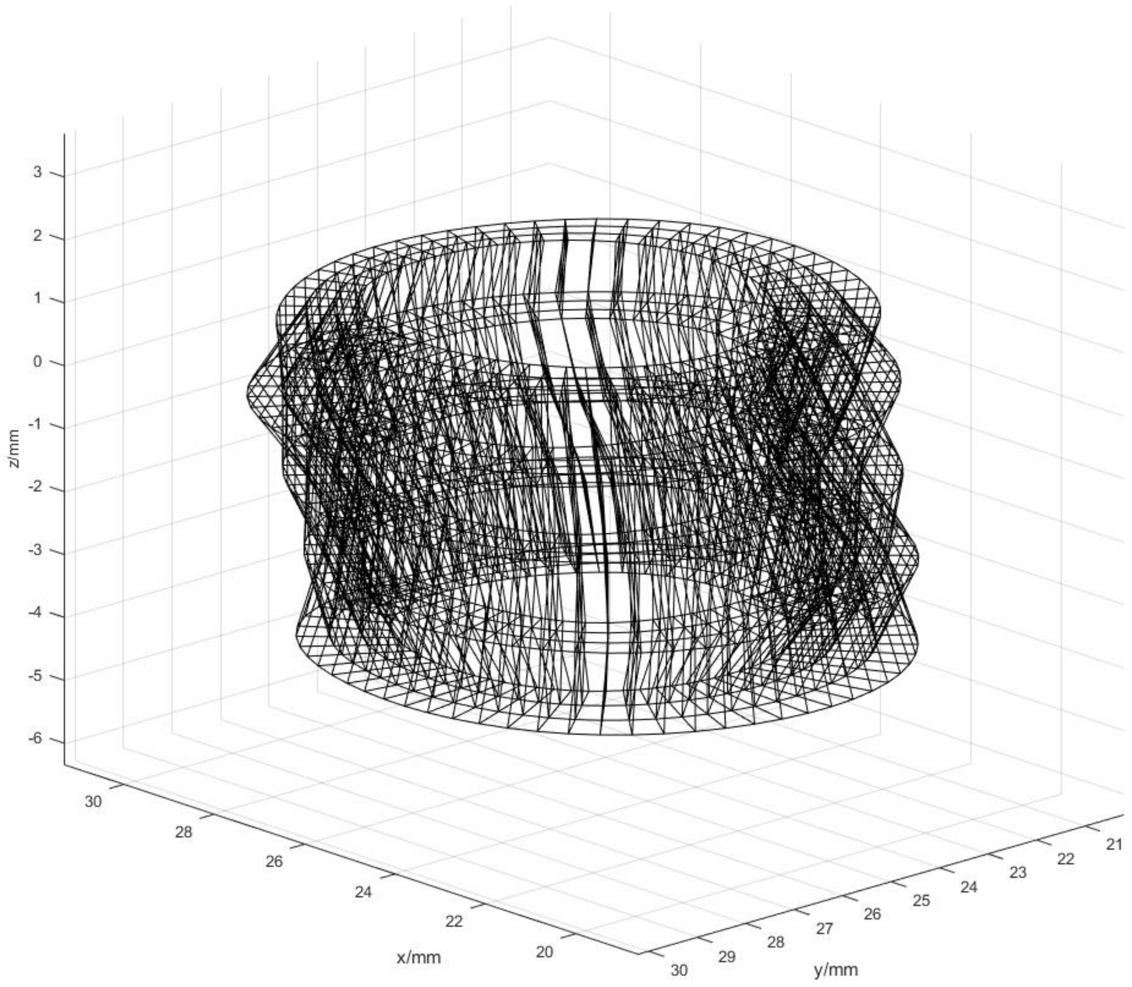


Figure 3.12: A 3D referential mesh of 60 × 3 × 4 elements in circumferential, transmural and longitudinal directions from the resting contours of LV scanned on Aug. 30, 2015

3.4.7 Converting to a Displacement Field and Mapping Referential Mesh

A MATLAB script written by Dr. Jonathan Suever was used for spatial interpolation of the myocardial displacements acquired in Chapter 3.4.5. For computational efficiency, the script fitted a continuous linear displacement field over the displacements that were within the region of the myocardial contours. The acquired displacement field was in terms of referential location of each node and frame number. Utilizing the queried displacements of all nodes, the referential mesh was deformed over time. Consequently, coordinates of all nodes in each frame were acquired. In addition, impressed by the temporal fitting result of Spottiswoode [31], a temporal smoothing was performed on coordinates of all nodes over time using fifth-order Fourier basis functions. All nodes in the same frame were assembled into a computational mesh. An example of a computational end-systolic mesh is shown in

Figure 3.13. Construction of computational meshes was implemented by a custom main function (Appendix J) of DENSE3D written in MATLAB.

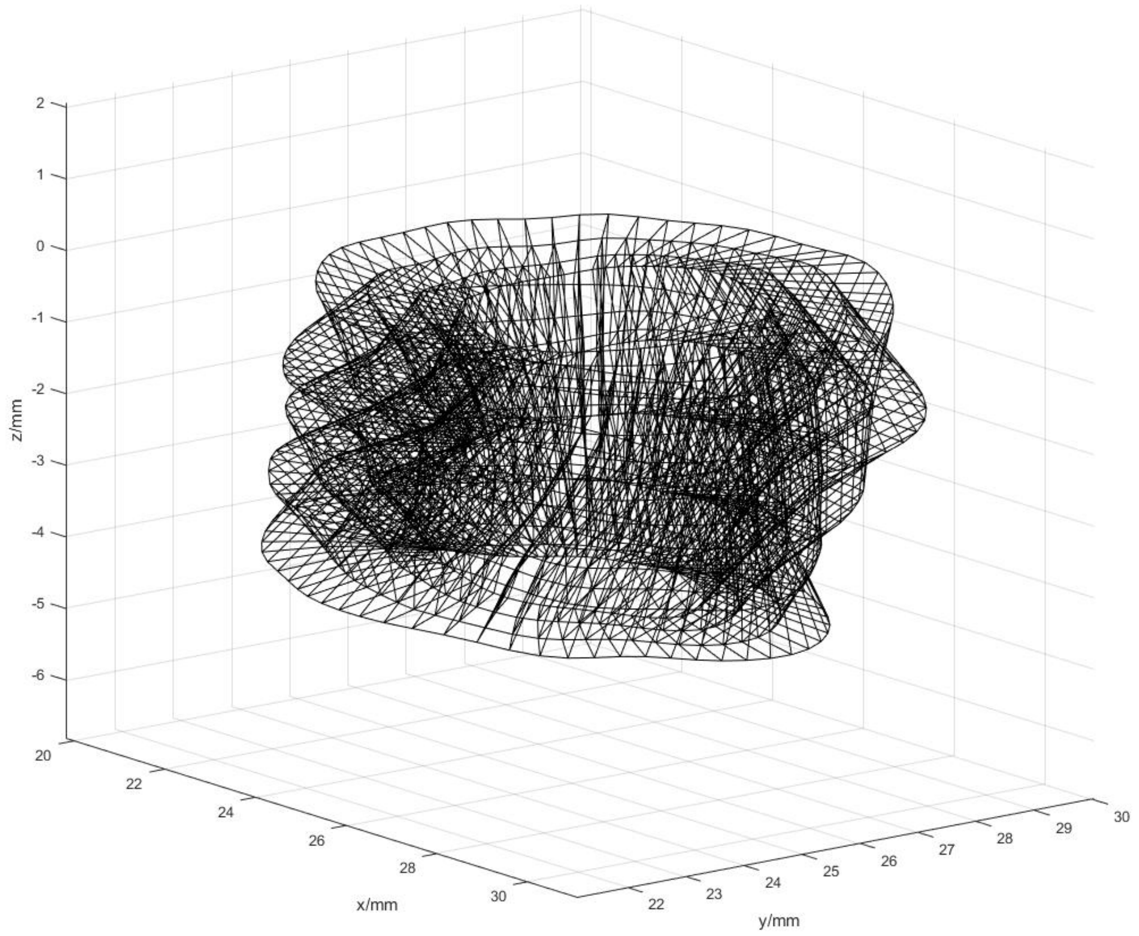


Figure 3.13: A computational end-systolic mesh of LV scanned on Aug. 30, 2015

3.4.8 Computing Mechanics

Considering a left ventricle in a specific frame (time t), the referential mesh is its referential configuration in the referential frame, while the computational mesh is its current configuration in the corresponding frame. The deformation $d\underline{x}$ of a current element P' located at $\underline{x}$ can be mapped into the deformation $d\underline{X}$ of a referential element P located at $\underline{X}$, where $\underline{X}$ and $\underline{x}$ are both defined in a 3D Cartesian coordinate system. Thus, the deformation gradient tensor $\underline{F}$ can be defined by:

$$\underline{F} = \frac{\partial \underline{x}(\underline{X}, t)}{\partial \underline{X}} \tag{3.4}$$

The Jacobian deformation J is equal to:

$$J = \det \underline{F} \tag{3.5}$$

The Lagrangian strain tensor in Cartesian coordinates can be written as:

$$\underline{E} = \frac{1}{2}(\underline{F}^T \underline{F} - \underline{I}) \tag{3.6}$$

where $\underline{I}$ is the identity tensor. Its components give us three normal strains and three shear strains in Cartesian coordinates.

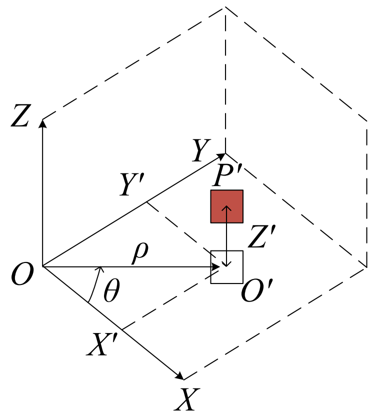


Figure 3.14: Transformation from Cartesian coordinates to polar coordinates

Depicted in Figure 3.14, point O is the center of the slice where element P' locates, point O' is the centroid of the element. The angle θ is defined by:

$$\theta = \tan^{-1}\frac{Y'}{X'} \tag{3.7}$$

The rotation matrix R which can transform Cartesian coordinate system into polar coordinate system can be written as:

(3.8)

Thus, the Lagrangian strain tensor in polar coordinates can be derived by:

$$\underline{E}_{polar} = R\underline{E}R^T = \begin{bmatrix} \varepsilon_{rr} & \varepsilon_{rc} & \varepsilon_{rl} \\ \varepsilon_{cr} & \varepsilon_{cc} & \varepsilon_{cl} \\ \varepsilon_{lr} & \varepsilon_{lc} & \varepsilon_{ll} \end{bmatrix} \tag{3.9}$$

where ε_{rr} is radial strain, ε_{cc} is circumferential strain, ε_{ll} is longitudinal strain, ε_{rc} , ε_{rl} and ε_{cl} are shear strains relative to two corresponding directions.

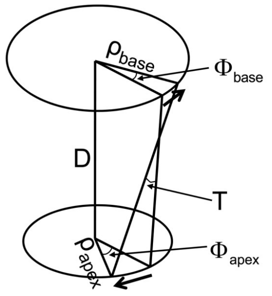


Figure 3.15: Definition of torsion T between basal and apical slices [32]

Torsion is described by the circumferential-longitudinal (CL) shear angle α_{CL} which takes radius ρ of a myocardial slice and distance D between slices into account (Figure 3.15). α_{CL} can be obtained by Equation (3.10) [32].

$$\alpha_{CL} = \sin^{-1}\frac{2\varepsilon_{cl}}{\sqrt{(1+2\varepsilon_{cc})(1+2\varepsilon_{ll})}} \tag{3.10}$$

In order to get Lagrangian strain in polar coordinates and CL shear angle of all elements in all frames, a custom module (Appendix K) was written in MATLAB.

3.4.9 Segmentation of the Middle Ventricle

Since the age of rats range from 5 to 7 months, a variety in heart sizes across subjects was noticeable. Considering this fact, mid-ventricular locations were estimated at the half length of the left ventricular cavity based on the long-axis endocardial contours. The layer of elements closest to mid-ventricular locations in the referential mesh were then located.

During the discretization of the LV (Chapter 3.4.6), the middle ventricle was automatically divided into septal, lateral, inferior and anterior segments circumferentially and transmural thirds (sub-endocardium, mid-wall, and sub-epicardium). A 12-segment mid-ventricular model is depicted in Figure 3.16 and segment names are summarized in Table 3.1. As American Heart Association (AHA) suggested [17], each segment can be assigned to one of the 3 major coronary arteries. Specifically, anterior and septal segments relate to the left anterior descending (LAD). Interior segments relate to right coronary artery (RCA). And lateral segments relate to the left circumflex coronary artery (LCX). The locations of coronary artery territories are demonstrated in Figure 3.17.

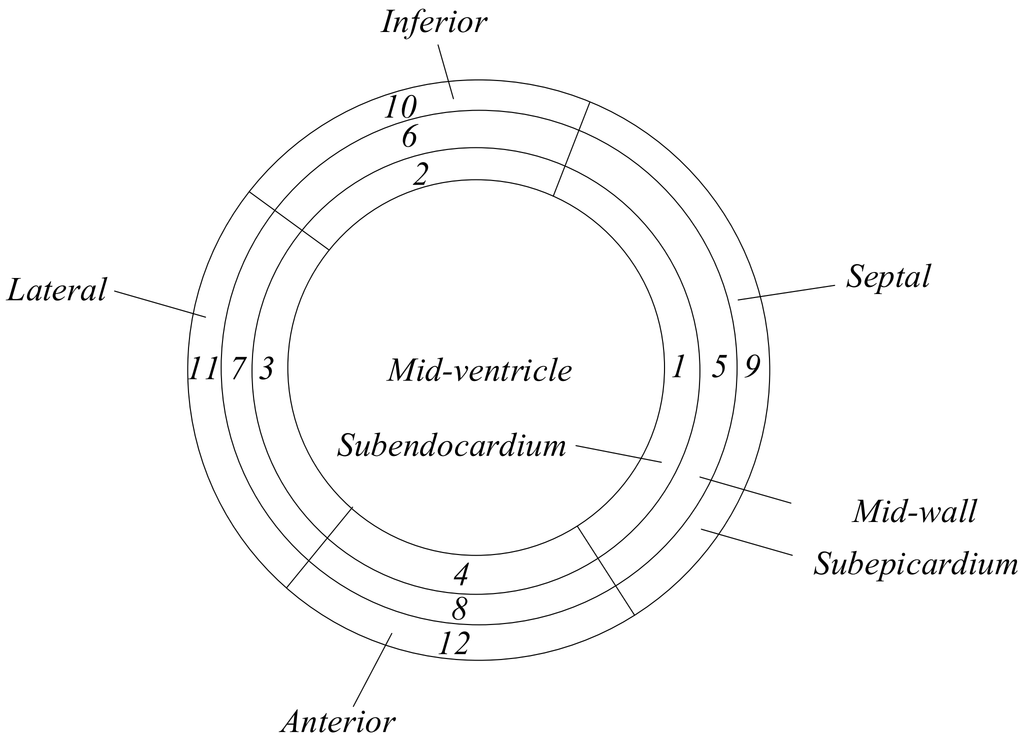
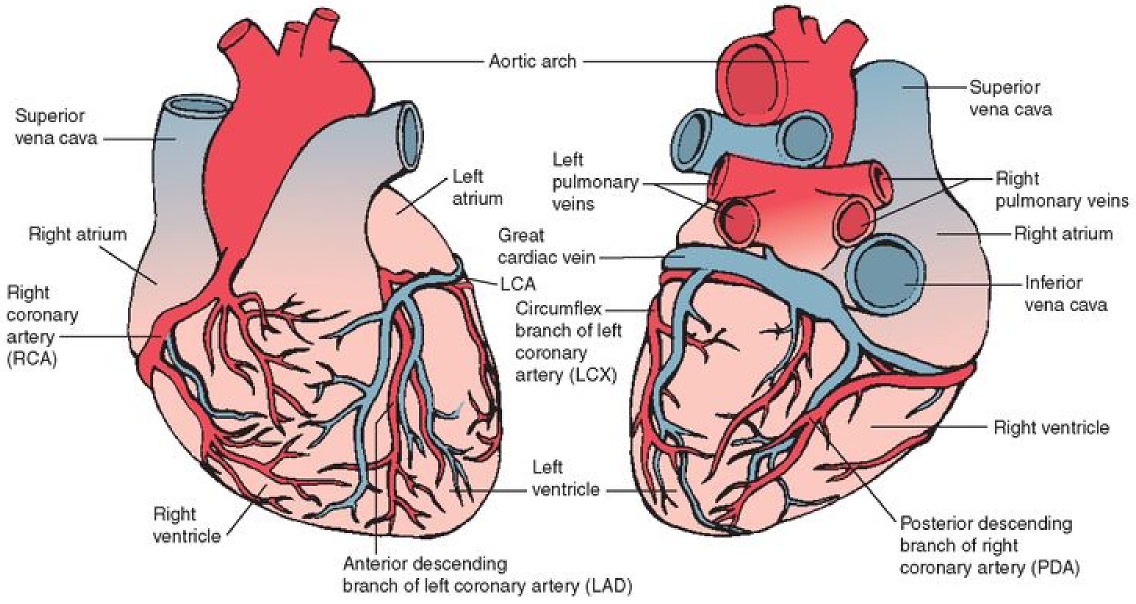


Figure 3.16: A 12-segment mid-ventricular model

Table 3.1: Notation of segments

Septal Segments	Inferior Segments	Lateral Segments	Anterior Segments
1. Septal sub-endocardium	2. Inferior sub-endocardium	3. Lateral sub-endocardium	4. Anterior sub-endocardium
5. Septal mid-wall	6. Inferior mid-wall	7. Lateral mid-wall	8. Anterior mid-wall
9. Septal sub-epicardium	10. Inferior sub-epicardium	11. Lateral sub-epicardium	12. Anterior sub-epicardium



(a) Anterior View (b) Inferior View

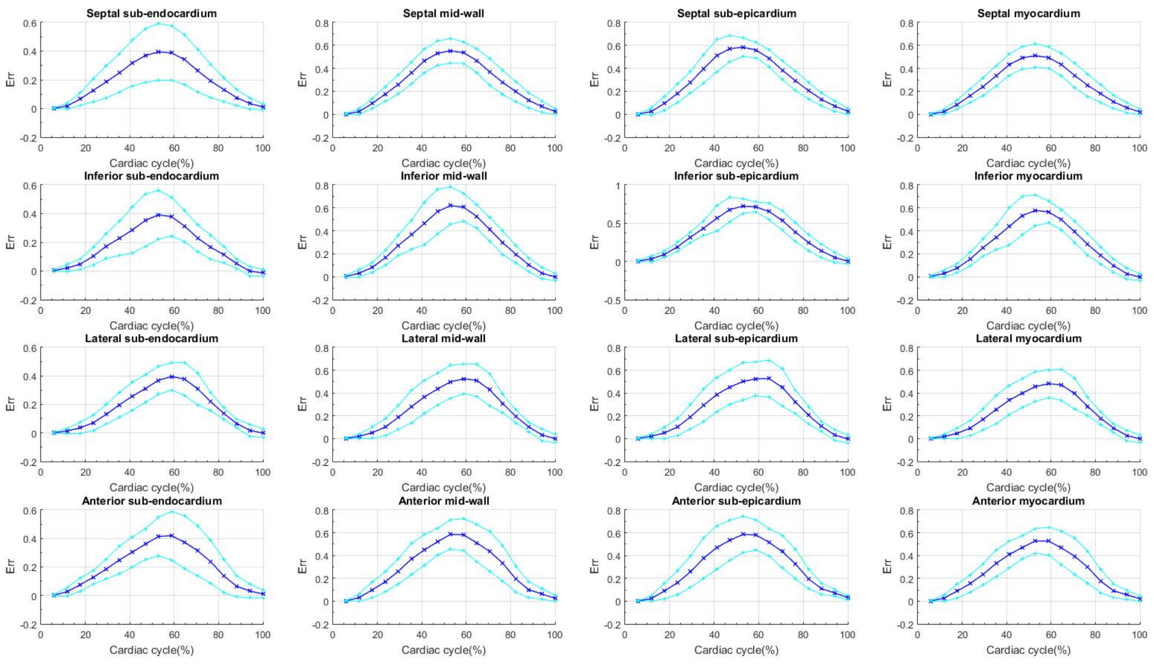
Figure 3.17: Coronary arteries and major veins of the heart [33]

In each frame, strain of each segment was evaluated by averaging the strains from the mid-ventricular good elements with the same segment attribution (generated during meshing the LV). Strain evaluation in each segment was implemented by the custom main function (Appendix J) of DENSE3D written in MATLAB.

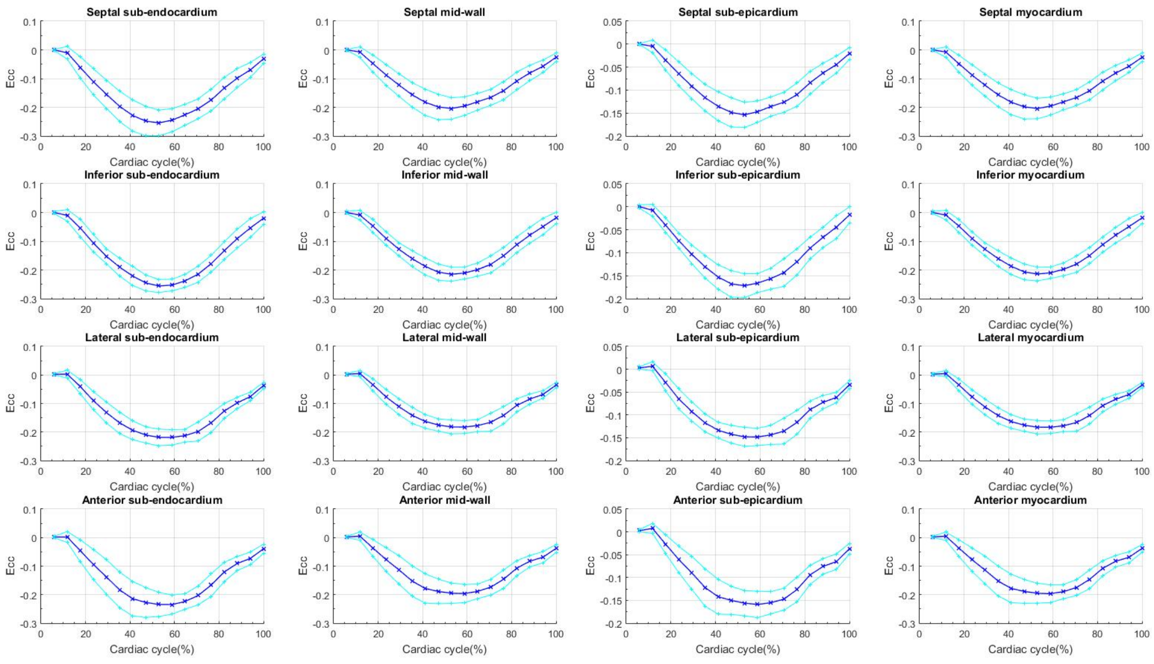
3.5 Results and Discussion

3.5.1 Mid-ventricular Lagrangian Strain

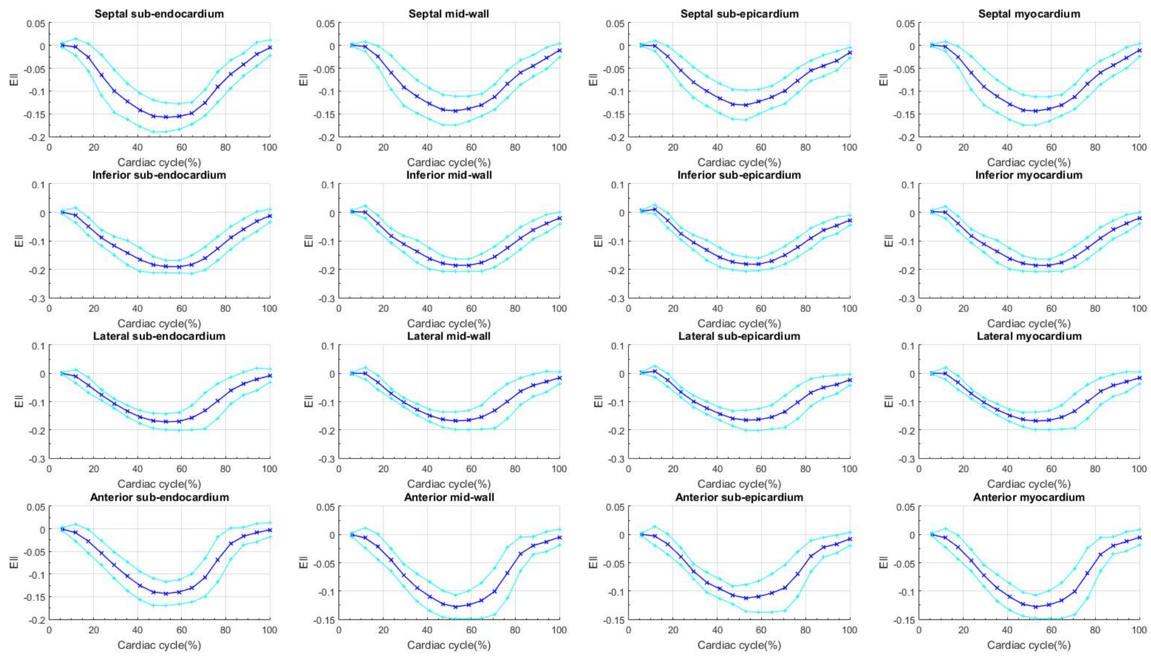
A custom script was written in MATLAB (Appendix L) for strain comparison across subjects. Since the subjects have different heart rates, strain in terms of frame were transferred into strain in terms of percentage of cardiac cycle. A temporal interpolation was done by fitting a cubic spline into scattered strain values distributed over time. Figure 3.18 summarizes six components of Lagrangian strain averaged across 10 rats in each of the 12 segments.



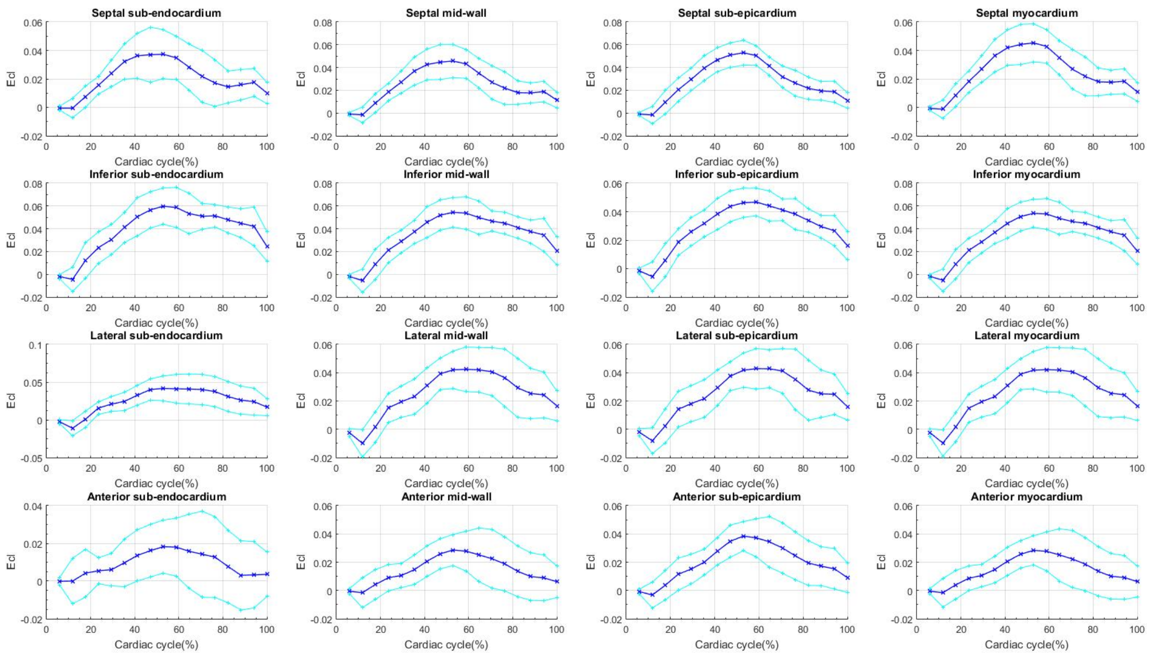
(a) Radial strain ε_{rr}



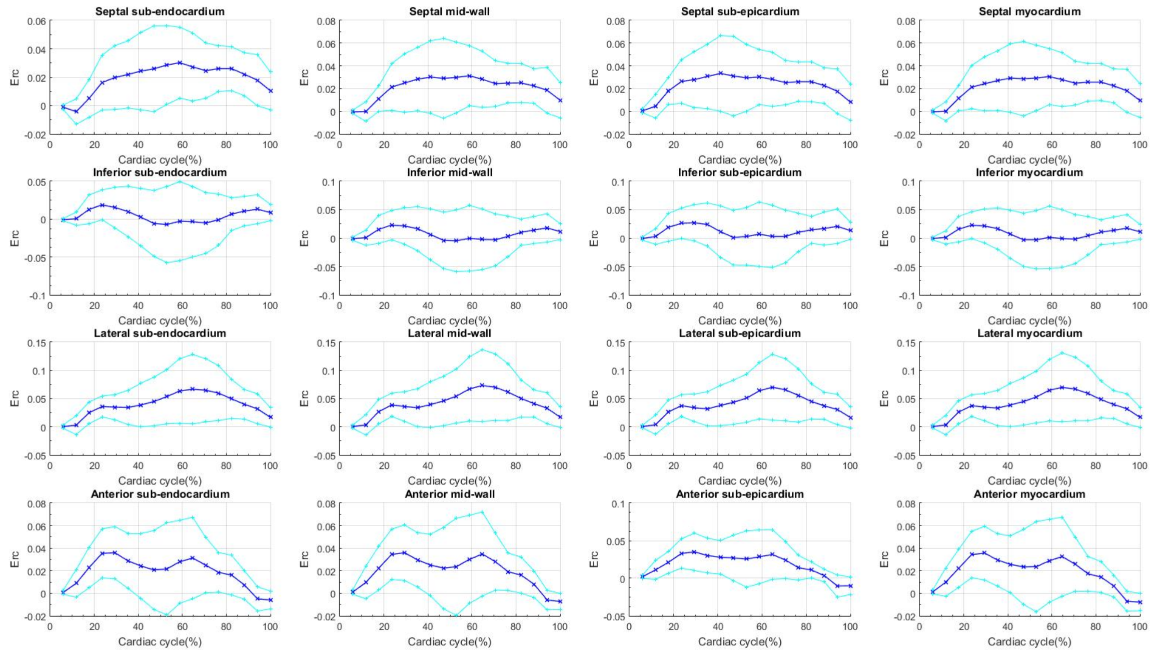
(b) Circumferential strain ε_{cc}



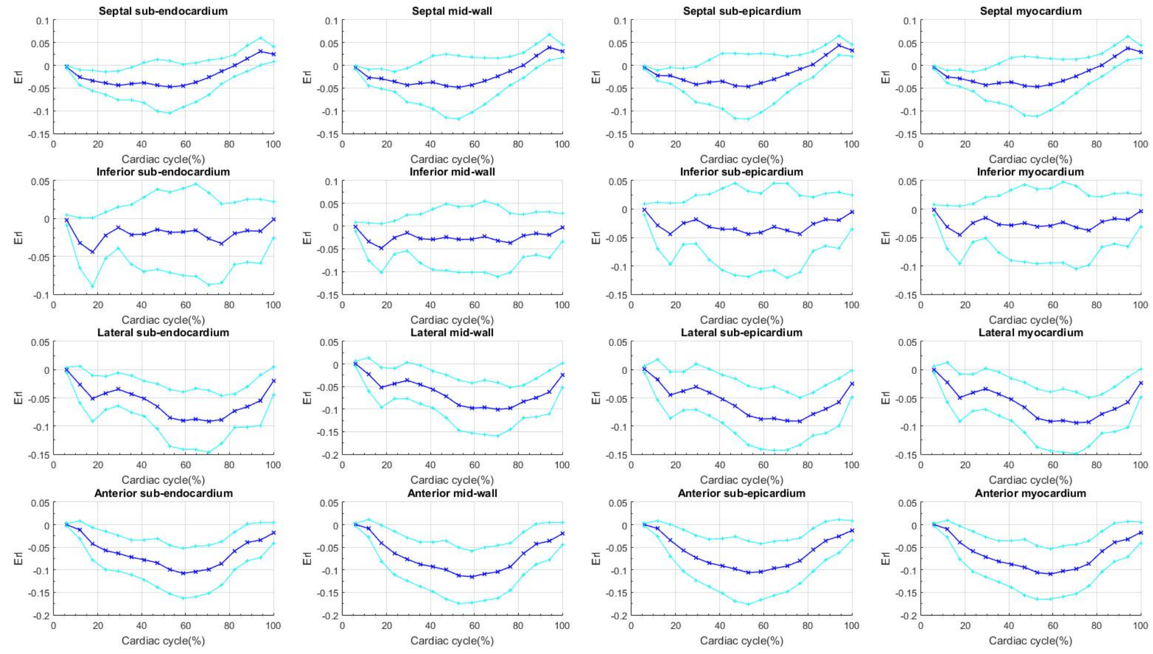
(c) Longitudinal strain ε_{ll}



(d) Circumferential-longitudinal shear strain ε_{cl}



(e) Radial-circumferential shear strain ε_{rc}



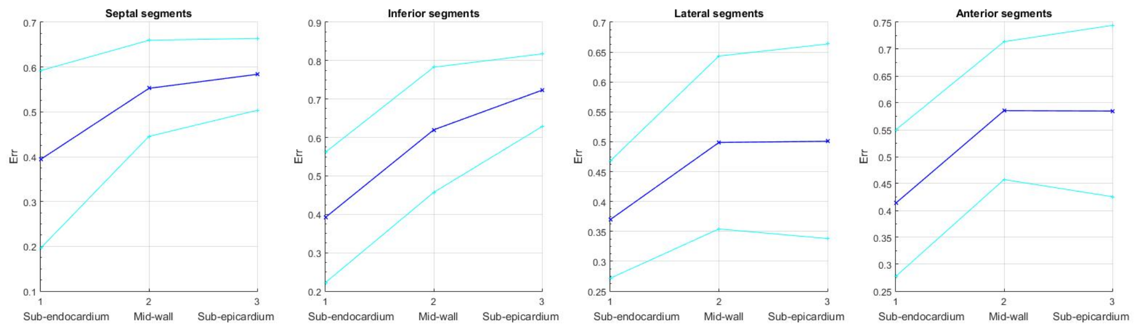
(f) Radial-longitudinal shear strain ε_{rl}

Figure 3.18: Lagrangian mid-ventricular strain averaged across subjects. The lighter lines above and below the average strain are plotted at ± 1 standard deviation.

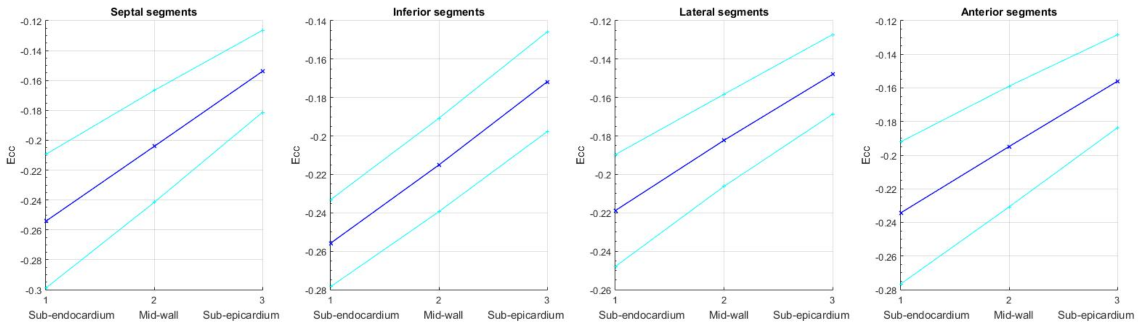
Since the material response of the myocardium is assumed to be nearly incompressible, Figure 3.18 reveals mechanical deformations of myocardium. During systole, positive increasing radial strain implied a thickening wall; negative decreasing circumferential strain implied a shortening along that direction; and negative decreasing longitudinal strain implied shortening length of LV due to atrioventricular plane displacement (AVPD). In the mid-wall, circumferential strain revealed the deformation of myocardial fiber, because the fiber angle is approximately aligned with its direction.

Figure 3.18 also indicates the repeatability is different for different strain components. Small differences are observed in circumferential and longitudinal strains across subjects, meaning circumferential and longitudinal strains in each segment are highly repeatable. As for radial strains, standard deviations were acceptable except in the sub-endocardium. For circumferential-longitudinal shear strain, differences across subjects were acceptable except in the anterior segments. However, radial-circumferential and radial-longitudinal shear strains were completely not repeatable.

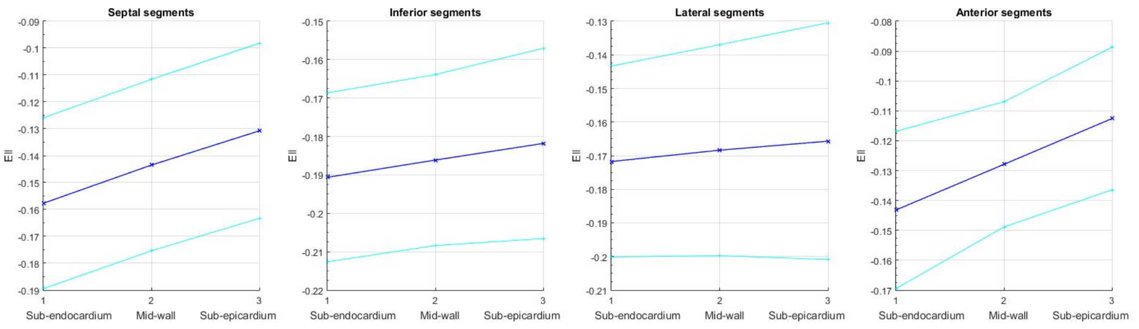
The transmural variation of end-systolic normal strain is depicted in Figure 3.19. In the free wall, since blood pool inside the cavity stresses on endocardium, strain is expected to be higher in the sub-endocardium than in the sub-epicardium. This phenomenon can be observed in circumferential and longitudinal strains. Those strains were approximately linearly decreasing across the wall. However, radial strain was increasing nonlinearly from endocardium to epicardium. The reverse of variation trend is possibly caused by the fact that imaging resolution was relatively low in the transmural direction considering wall thickness is much less than wall length. Consequently, displacement-encoded voxels in the transmural direction (normally 3-5 voxels) were not enough for reconstructing the actual variation of radial strain. What's worse, the spatial smoothing of 3-5 displacements across the wall might ruin the actual variation of radial strain.



(a) Transmural variation of radial strain ε_{rr}



(b) Transmural variation of circumferential strain ε_{cc}



(c) Transmural variation of longitudinal strain ε_{ll}

Figure 3.19: Transmural variation of normal strain at end systole. The lighter lines above and below the average normal strain are plotted at ±1 standard deviation.

3.5.2 Torsion of the LV

In order to calculate the torsion of the whole LV, using the custom script written in MATLAB (Appendix L), torsions in terms of time were also tranferred into torsions in terms of percentage of cardiac cycle, followed by being averaged over all elements. The results are depicted in Figure 3.20 and Figure 3.21.

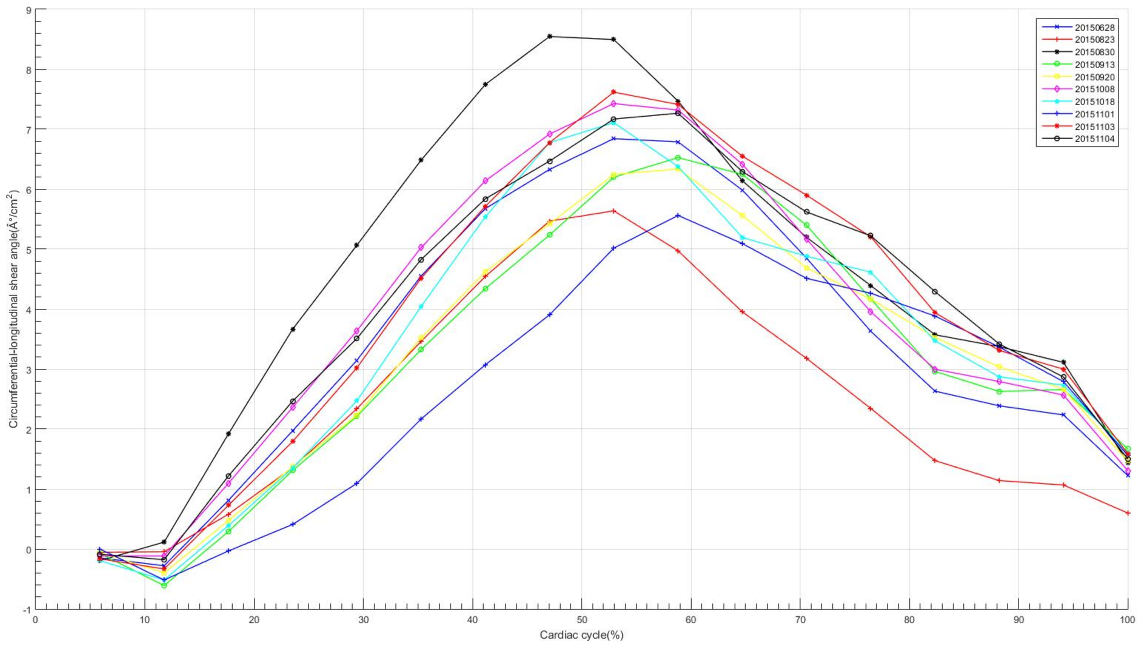


Figure 3.20: CL shear angles across 10 sudjects

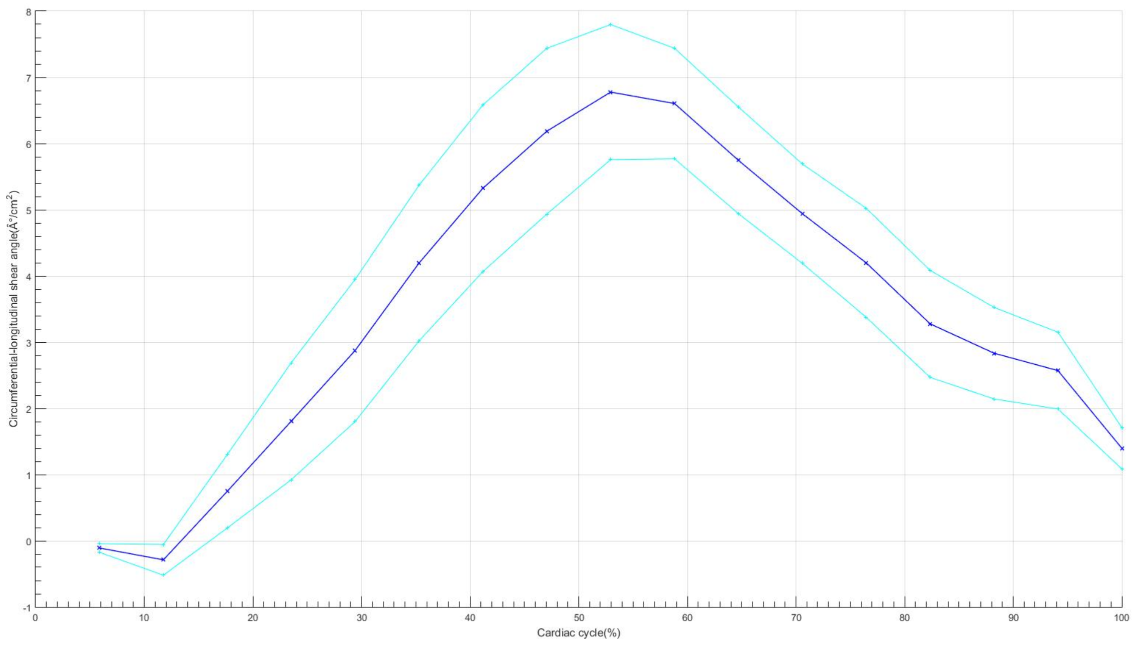


Figure 3.21: CL shear angle averaged across subjects. The lighter lines above and below the average CL shear angle are plotted at ± 1 standard deviation.

As shown in Figure 3.20, differences in CL shear angle across subjects were acceptable. Thus, CL shear angle is repeatable. Figure 3.21 shows positive values of CL shear angle over time, indicating the LV twist in the direction as Figure 3.22 pictures. The basal slice twisted 6.8±1 °/cm^2 with respect to the apical slice at end systole.

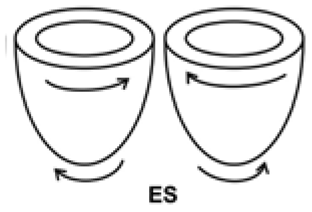


Figure 3.22: Positive direction of CL shear angle

3.6 Conclusion

A methodology for quantification of 3D Lagrangian strain using DENSE CMR images was proposed. Tools written in MATLAB were developed for the methodology. The LV was meshed, followed by displacements tracking for all elements. Lagrangian strain was then calculated for each element. Average strains of all mid-ventricular segments were compared and discussed individually. Torsion of the whole LV was also evaluated.

Statistical stain analysis indicated that circumferential and longitudinal strains were highly repeatable, while radial strain and CL shear strain were less repeatable than circumferential and longitudinal strains. Radial strain, circumferential strain, longitudinal strain, and CL shear angle provide a measurement of mechanical deformations in the cardiac wall over a cardiac cycle.

Appendices

Appendix A – Abbreviations

AV	Atrioventricular
AVPD	Atrioventricular Plane Displacement
CMR	Cardiac Magnetic Resonance
CHF	Congestive Heart Failure
CSO	Contour Segmentation Objects
DENSE	Displacement ENcoding via Stimulated Echoes
DICOM	Digital Imaging and Communications in Medicine
Dob	Dobutamine
ECG	Electrocardiogram
ES	End Systole
ED	End Diastole
HLA	Horizontal Long-Axis
HARP	Harmonic Phase
LAD	Left Anterior Descending (Artery)
LCX	Left Circumflex
LV	Left Ventricle
MRI	Magnetic Resonance Imaging
PVR	Pulmonary Valve Replacement
RV	Right Ventricle
RVEDV	Right Ventricular End-Diastolic Volume
RVEF	Right Ventricular Ejection
RVSV	Right Ventricular Stroke Volume
RCA	Right Coronary Artery
SA	Short Axis
SNR	Signal-to-noise Ratio
TOF	Tetralogy of Fallot
VLA	Vertical Long-Axis
WEM	Winged Edge Mesh

Appendix B – Python Codes in MeVisLab

```
# change for different surface:
SurfaceName = 'ED_SA_RV'
# change for different scan:
VoxelSize = [1.406, 1.406, 8]
csoList=CTX.field(SurfaceName + ".outputCSOList").object()
numCSOs=csoList.getNumCSOs()
numCSO=0
print 'Total Number of CSOs', numCSOs
# change for different scan:
text_file = open("Post_" + SurfaceName + ".txt", "w")

#cycle through all CSOs
for i in range(numCSOs):
    cso=csoList.getCSOAt(i)
    numCSO=numCSO+1
    #CSO Lable
    #if cso.getTimePointIndex() == 2:
    if cso.getTimePointIndex() == 14:
        csopoints=cso.getPathPoints()
        #print out coordinates of path points
        for j in range(0,len(csopoints), 3):
            CTX.field("WorldVoxelConvert.worldPos").value = [csopoints[j], csopoints[j+1], csopoints[j+2]]
            voxelPos=CTX.field("WorldVoxelConvert.voxelPos").value
            XPos=voxelPos[0]*VoxelSize[0]
            YPos=voxelPos[1]*VoxelSize[1]
            ZPos=str(round(voxelPos[2], 1)*VoxelSize[2])
            text_file.write('{0:2f} {1:3f} {2:4s}\n'.format(XPos, YPos, ZPos))
text_file.close()
print 'Total Number of outputed CSOs', numCSO
```

Appendix C – Matlab User-Defined Function of Converter from DENSEanalysis to Correcterborders

```
%% CONVERT DENSE DATA TO CORRECTERBORDERS DATA ------------------------------------------
% Last Modified: 11:15 PM Wednesday, October 28, 2015
% Modified By: Zhanqiu Liu (lafeir.lew@gmail.com)

function [ROI,Sequence,Images,roi,seq,img] = dense2correcterborders(roi,seq,img)

    %% No "if" will be faster:
    if ~isfield(seq,'AnalysisUID')
        seq(1).AnalysisUID = '';
    end
    if ~isfield(seq,'AnalysisName')
        seq(1).AnalysisName = '';
    end
    if ~isfield(seq,'Orientation')
        seq(1).Orientation = '';
    end

    for k = 1:numel(seq)

        %% probably NOT initialized by "rgs = cat(1, rgs, roi.ROIGroup.loadobj(rois(k)));" inside "CBdata.m"
            seq(k).AnalysisUID = seq(k).DENSEanalysisUID;
            seq(k).AnalysisName = seq(k).DENSEanalysisName;

        if isempty(seq(k).Orientation)
            if ~isempty(strfind(seq(k).DENSEanalysisName, 'SA'))
                seq(k).Orientation = 'ShortAxis';
            elseif ~isempty(strfind(seq(k).DENSEanalysisName, 'LA'))
                seq(k).Orientation = 'TwoChamber';
            else
                error(sprintf('%s:invalidInput',mfilename),'%s',...
                    'Invalid DENSE seq(',sprintf('%d',k),').Orientation: ',sprintf('%s',seq(k).Orientation),'. Cannot
recognize the type for SA or LA.');
                [ROI,Sequence,Images,roi,seq,img] = deal([]);
                return
            end
        end

    end



    ii = 0;
    nrois = numel(roi);
    for k = 1:nrois

        nFrames = size(roi(k).Position,1);

        %% Transfer voxel position of points
        switch roi(k).Type
            case 'SA'
                if strcmp(seq(roi(k).SeqIndex(1)).Orientation,'ShortAxis')
                ii = ii + 1;
                for frame = 1:nFrames
                    ROI(ii).ROIs(frame, 1) = struct('Nodes', roi(k).Position{frame, 1}, 'Periodic', true, 'Orientation',
'ShortAxis', 'Type', 'LVEpicardium');
                    ROI(ii).ROIs(frame, 2) = struct('Nodes', roi(k).Position{frame, 2}, 'Periodic', true, 'Orientation',
'ShortAxis', 'Type', 'LVEndocardium');
                end
                else
                    msgbox(strcat('Incorrect ROI type for Seq. #',num2str(k),'should be SA instead of LA'),'Error')
                end
            case 'LA'
                if strcmp(seq(roi(k).SeqIndex(1)).Orientation,'TwoChamber')
                ii = ii + 1;
                for frame = 1:nFrames
                    ROI(ii).ROIs(frame, 1) = struct('Nodes', roi(k).Position{frame, 1}, 'Periodic', false, 'Orientation',
'TwoChamber', 'Type', 'LVEpicardium');
                    ROI(ii).ROIs(frame, 2) = struct('Nodes', roi(k).Position{frame, 2}, 'Periodic', false, 'Orientation',
'TwoChamber', 'Type', 'LVEndocardium');
                end
                else
                    msgbox(strcat('Incorrect ROI type for Seq. #',num2str(k),'should be LA instead of SA'),'Error')
                end
            case 'SAFull'
                if strcmp(seq(roi(k).SeqIndex(1)).Orientation,'ShortAxis')
```

```
                        ii = ii + 1;
                        for frame = 1:nFrames
                            ROI(ii).ROIs(frame, 1) = struct('Nodes', roi(k).Position{frame, 1}, 'Periodic', true, 'Orientation',
'ShortAxis', 'Type', 'Epicardium');
                            ROI(ii).ROIs(frame, 2) = struct('Nodes', roi(k).Position{frame, 2}, 'Periodic', true, 'Orientation',
'ShortAxis', 'Type', 'RVEndocardium');
                            ROI(ii).ROIs(frame, 3) = struct('Nodes', roi(k).Position{frame, 3}, 'Periodic', true, 'Orientation',
'ShortAxis', 'Type', 'LVEndocardium');
                        end
                        else
                            msgbox(strcat('Incorrect ROI type for Seq. #',num2str(k),'should be SA instead of LA'),'Error')
                        end
                    case 'LAFull'
                        if strcmp(seq(roi(k).SeqIndex(1)).Orientation,'TwoChamber')
                        ii = ii + 1;
                        for frame = 1:nFrames
                            ROI(ii).ROIs(frame, 1) = struct('Nodes', roi(k).Position{frame, 1}, 'Periodic', false, 'Orientation',
'TwoChamber', 'Type', 'Epicardium');
                            ROI(ii).ROIs(frame, 2) = struct('Nodes', roi(k).Position{frame, 2}, 'Periodic', false, 'Orientation',
'TwoChamber', 'Type', 'RVEndocardium');
                            ROI(ii).ROIs(frame, 3) = struct('Nodes', roi(k).Position{frame, 3}, 'Periodic', false, 'Orientation',
'TwoChamber', 'Type', 'LVEndocardium');
                        end
                        else
                            msgbox(strcat('Incorrect ROI type for Seq. #',num2str(k),'should be LA instead of SA'),'Error')
                        end
                    case 'curve'
if ismember(roi(k).Name, {'LVEpicardium','RVEndocardium','RVEpicardium','Epicardium','Infarct','Reference'})
    flag_curve = false;
    for jj = [1:k-1, k+1:nrois]
        if isequal(roi(jj).SeqIndex,roi(k).SeqIndex) && ismember(roi(jj).Type,{'SA','LA','SAFull','LAFull'})
            if strncmpi(roi(jj).Type,'SA', 2)
                Orientation = 'ShortAxis';
            else
                Orientation = 'TwoChamber';
            end
            if any(regexp(roi(jj).Type,'Full$'))
                nContours = 3;
            else
```

```
                                        nContours = 2;
                                end
                                col = size(ROI(jj).ROIs,2)+1;
                                if col < nContours+1
                                        col = nContours + 1;
                                end
                                for frame = 1:nFrames
                                        ROI(jj).ROIs(frame, col) = struct('Nodes', roi(k).Position{frame, 1}, 'Periodic', true, 'Orientation', Orientation,
'Type', roi(k).Name);
                                end
                                ROI(jj).ROIs(1, col).UID = roi(k).UID;
                                flag_curve = true;
                                break
                        end
                end
                if flag_curve
                        continue
                else
                        ii = ii + 1;
                        if strfind(seq(roi(k).SeqIndex(1)).Orientation, 'SA')
                                Orientation = 'ShortAxis';
                        else
                                Orientation = 'TwoChamber';
                        end
                        for frame = 1:nFrames
                                ROI(ii).ROIs(frame, 1) = struct('Nodes', roi(k).Position{frame, 1}, 'Periodic', true, 'Orientation', Orientation, 'Type',
roi(k).Name);
                        end
                        ROI(ii).ROIs(1, 1).UID= roi(k).UID;
                end
else
        msgbox( {strcat('"',roi(k).Name,'" is NOT a unsupported name for the type "curve" in Correcterborders.'),' Supported names of the
type "curve":',' LVEpicardium, RVEndocardium, RVEpicardium, Epicardium, Infarct, Reference '},'Warning');
        continue
end


                                otherwise
                                        msgbox( {strcat('"',roi(k).Type,'" is NOT a supported type of ROI in Correcterborders:'),' You can create a new
ROI and select a different type in DENSE2D'},'Warning');
```

```
                continue
            end


            ROI(ii).Name = roi(k).Name;
            ROI(ii).UID = roi(k).UID;
            %% dimension mismatch if Sequence exists
            Sequence(ii) = seq(roi(k).SeqIndex(1));
            Images(ii) = img(roi(k).SeqIndex(1));

            %% Initialized at the end of function "load" inside "CBdata.m"
            ROI(ii).Orientation = Sequence(ii).Orientation;
        end

    end
```

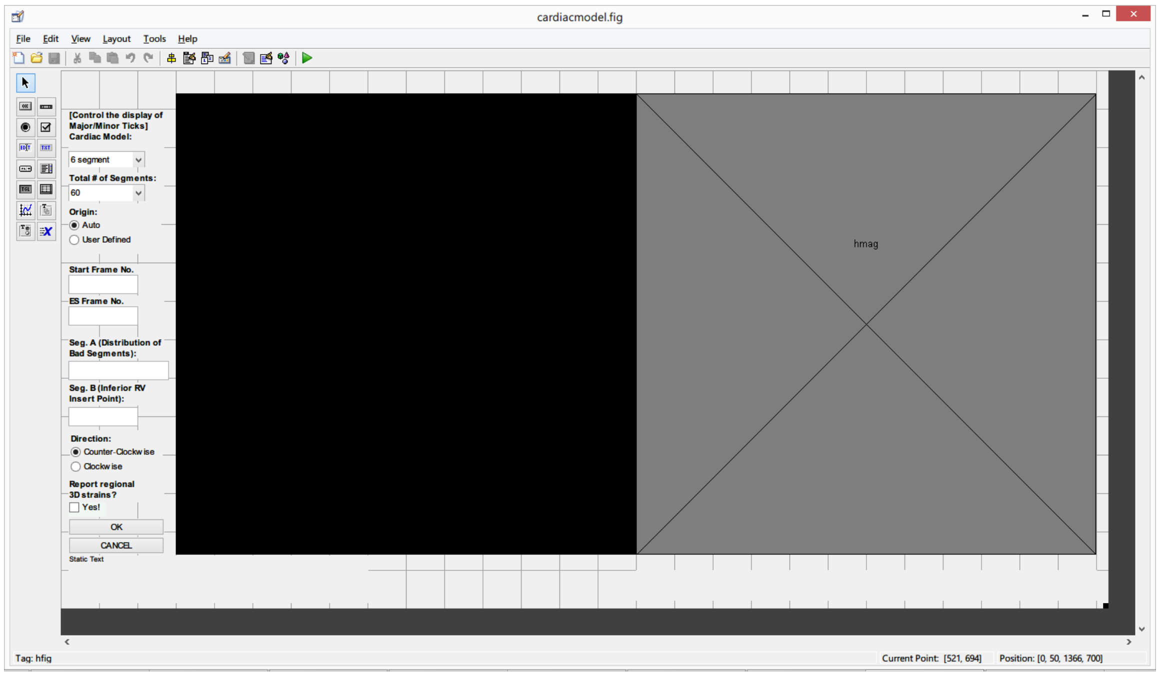
Appendix D – Matlab User-Defined Function of Converter from Correcterborders to DENSEanalysis

```
%% CONVERT CORRECTERBORDERS DATA TO DENSE DATA ------------------------------------------
% Last Modified: 11:15 PM Wednesday, October 28, 2015
% Modified By: Zhanqiu Liu (lafeir.lew@gmail.com)

% function [roi,seq,img] = correcterborders2dense(ROI,Sequence,Images,roi,seq,img)
function [roi,seq,flag_savecdb] = correcterborders2dense(ROI,Sequence,roi,seq)

    flag_savecdb = false;

    newslice = numel(roi) + 1;

    for k = 1:numel(ROI)

        [nFrames,nContours] = size(ROI(k).ROIs);

        if isempty(ROI(k).ROIs)
            error(sprintf('%s:invalidInput',mfilename),'%s',...
                'Invalid Correcterborders ROI(',sprintf('%d',k),'): no voxel points inside.');
            roi = struct([]);
            return
        end



        index = find(strcmp(ROI(k).UID, {roi.UID}));
        if numel(index) == 0
            error(sprintf('%s:invalidInput',mfilename),'%s',...
                'Invalid Correcterborders ROI(',sprintf('%d',k),').UID: from DENSE2D to CORRECTERBORDERS, UID is changed. Try to Match ROIs using Sequence().AnalysisName and seq().DENSEanalysisName.');
            roi = struct([]);
            return
        elseif numel(index) >= 2
            error(sprintf('%s:invalidInput',mfilename),'%s',...
                'Invalid Correcterborders ROI(',sprintf('%d',k),').UID: from DENSE2D to CORRECTERBORDERS, a same UID has been repeated for ',sprintf('%d',numel(index)),' times. Try to Match ROIs using Sequence().AnalysisName and seq().DENSEanalysisName.');
        roi = struct([]);
        return
end

if strcmpi(roi(index).Type,'SAFull') || strcmpi(roi(index).Type,'LAFull')
        flag_biv = true;
else
        flag_biv = false;
end



for ii = 1:nContours %size(ROI(k).ROIs,2)
        %% Transfer voxel position of points
        switch ROI(k).ROIs(1, ii).Type
        case 'LVEpicardium'
                if ~flag_biv
                        for frame = 1:nFrames
                                roi(index).Position{frame, 1} = ROI(k).ROIs(frame, ii).Nodes;
                                nPoints = size(roi(index).Position{frame, 1},1);
                                if nPoints == 0
                                        nPoints = 1;
                                end
                                roi(index).IsCurved{frame, 1} = ones(nPoints,1);
                                roi(index).IsCorner{frame, 1} = zeros(nPoints,1);
                                %% NOT working since create a cell arrray of logical in each element!
                                % repmat({true}, [nPoints,1])
                                % repmat({false}, [nPoints,1])
                        end
                        flag_newslice = false;
                else
                        flag_newslice = true;
                end
        case 'LVEndocardium'
                if ~flag_biv
                        for frame = 1:nFrames
                                roi(index).Position{frame, 2} = ROI(k).ROIs(frame, ii).Nodes;
                                nPoints = size(roi(index).Position{frame, 2},1);
                                if nPoints == 0
                                        nPoints = 1;
```

```
                    end
                    roi(index).IsCurved{frame, 2} = ones(nPoints,1);
                    roi(index).IsCorner{frame, 2} = zeros(nPoints,1);
                end
            else
                for frame = 1:nFrames
                    roi(index).Position{frame, 3} = ROI(k).ROIs(frame, ii).Nodes;
                    nPoints = size(roi(index).Position{frame, 1},1);
                    if nPoints == 0
                        nPoints = 1;
                    end
                    roi(index).IsCurved{frame, 3} = ones(nPoints,1);
                    roi(index).IsCorner{frame, 3} = zeros(nPoints,1);
                end
            end
            flag_newslice = false;
        case 'Epicardium'
            if flag_biv
                for frame = 1:nFrames
                    roi(index).Position{frame, 1} = ROI(k).ROIs(frame, ii).Nodes;
                    nPoints = size(roi(index).Position{frame, 1},1);
                    if nPoints == 0
                        nPoints = 1;
                    end
                    roi(index).IsCurved{frame, 1} = ones(nPoints,1);
                    roi(index).IsCorner{frame, 1} = zeros(nPoints,1);
                end
                flag_newslice = false;
            else
                flag_newslice = true;
            end
        case 'RVEndocardium'
            if flag_biv
                for frame = 1:nFrames
                    roi(index).Position{frame, 2} = ROI(k).ROIs(frame, ii).Nodes;
                    nPoints = size(roi(index).Position{frame, 1},1);
                    if nPoints == 0
                        nPoints = 1;
                    end
                    roi(index).IsCurved{frame, 2} = ones(nPoints,1);
                    roi(index).IsCorner{frame, 2} = zeros(nPoints,1);
                end
                flag_newslice = false;
            else
                flag_newslice = true;
            end
        case {'RVEpicardium','Infarct','Reference'}
            flag_newslice = true;
        otherwise
            flag_newslice = false;
            flag_savecdb = true;
        end

        if flag_newslice
            if isfield(ROI(k).ROIs(1, ii),'UID') && ~isempty(ROI(k).ROIs(1, ii).UID)
                idx = find(strcmp(ROI(k).ROIs(1, ii).UID, {roi.UID}));
                if numel(idx) == 0
                    error(sprintf('%s:invalidInput',mfilename),'%s',...
                        'Invalid Correcterborders ROI(',sprintf('%d',k),').ROIs.(1, ',sprintf('%d',ii),').UID: from DENSE2D to CORRECTERBORDERS, UID is changed. Try to Match ROIs using Sequence().AnalysisName and seq().DENSEanalysisName.');
                    roi = struct([]);
                    return
                elseif numel(idx) >= 2
                    error(sprintf('%s:invalidInput',mfilename),'%s',...
                        'Invalid Correcterborders ROI(',sprintf('%d',k),').ROIs.(1, ',sprintf('%d',ii),').UID: from DENSE2D to CORRECTERBORDERS, a same UID has been repeated for ',sprintf('%d',numel(idx)),' times. Try to Match ROIs using Sequence().AnalysisName and seq().DENSEanalysisName.');
                    roi = struct([]);
                    return
                end
                for frame = 1:nFrames
                    roi(idx).Position{frame, 1} = ROI(k).ROIs(frame, ii).Nodes;
                    nPoints = size(roi(idx).Position{frame, 1},1);
                    if nPoints == 0
                        nPoints = 1;
                    end
                    roi(idx).IsCurved{frame, 1} = ones(nPoints,1);
```

```
                    roi(idx).IsCorner{frame,1} = zeros(nPoints,1);
                end
            else
                for frame = 1:nFrames
                    roi(newslice).Position{frame, 1} = ROI(k).ROIs(frame, ii).Nodes;
                    nPoints = size(roi(newslice).Position{frame, 1},1);
                    if nPoints == 0
                        nPoints = 1;
                    end
                    roi(newslice).IsCurved{frame, 1} = ones(nPoints,1);
                    roi(newslice).IsCorner{frame, 1} = zeros(nPoints,1);
                end
                roi(newslice).IsClosed = repmat({true}, [nFrames,1]);
                roi(newslice).SeqIndex = roi(index).SeqIndex;
                roi(newslice).Type = 'curve';
                roi(newslice).Name= ROI(k).ROIs(1,ii).Type;
                roi(newslice).UID = dicomuid;
                newslice = newslice + 1;
            end
        end
    end


    %% Detect if some contours are missing for some frames:
    tmp = roi(index).Position;
    emptyCells = cellfun(@isempty, tmp);
    tmp(all(emptyCells,2),:) = [];
    nContours = sum(sum(cellfun(@isempty, tmp)),2);
    if ~isempty(tmp) && nContours~=0
        [row,col] = find(~emptyCells);
        msgbox({strcat('Uncompleted DENSE roi(',sprintf('%d',k),').Position: '), strcat(sprintf('% d',nContours), ' contours
are missing at frame No. ', sprintf('% d',row))},'Error');
        roi = struct([]);
        return
    end

    % Update viewports and window / level data
    idx = roi(index).SeqIndex(1);
    % contrast:
    seq(idx).Window     = Sequence(k).Window;
    seq(idx).Level      = Sequence(k).Level;
    % zoom:
    seq(idx).Viewport = Sequence(k).Viewport;

end

end
```

Appendix E – Design of GUI of Mesh Configuration in Matlab



[Control the display of Major/Minor Ticks] Cardiac Model: 6 segment

Total # of Segments: 60

Origin: [✓ Auto] User Defined

Start Frame No.

ES Frame No.

Seg. A (Distribution of Bad Segments):

Seg. B (Inferior RV Insert Point):

Direction: [✓ Counter-Clockwise] Clockwise

Report regional 3D strains? Yes!

Tag: hfig

Current Point: [521, 694] Position: [0, 50, 1366, 700]

Appendix F – Main Function of Meshing Module in Matlab

```
function options = mainFcn(api)


    % colors
    api.clrA = [1 0.5 0];
    api.clrB = [0.5 0.5 1.0];
    api.clrP = [0.75 0.25 1];
    api.clrH = [1 1 0];



    % COMMON INITIALIZATION-----------------------------------

    % display magnitude image
    api.him = image('parent',api.hmag,...
        'cdata',zeros(api.Isz),...
        'cdatamapping','scaled');
    set(api.hmag,'clim',[0 1]);
    set(api.hfig,'colormap',gray(256),'renderer','zbuffer');

    % link axes / set display limits
```

```
linkaxes([api.hrest,api.hmag]);
axis(api.hmag,api.drng);

% axes titles
htitle(1) = title(api.hrest,'Resting Contours');
htitle(2) = title(api.hmag,'Magnitude Viewer');
set(htitle,'fontweight','bold','fontsize',12);



% display resting contours (static)
heprest = line('parent',api.hrest,...
    'xdata',api.RestingContour{1}(:,1),...
    'ydata',api.RestingContour{1}(:,2),...
    'color','r','hittest','off');
henrest = line('parent',api.hrest,...
    'xdata',api.RestingContour{2}(:,1),...
    'ydata',api.RestingContour{2}(:,2),...
    'color','g','hittest','off');

% display frame contours
api.hep = copyobj(heprest,api.hmag);
api.hen = copyobj(henrest,api.hmag);

%% @ 'Resting Contours'
% minor spokes
api.hspokeminor = patch(...
    'parent',           api.hrest,...
    'vertices',         NaN(1,2),...
    'faces',            1,...
    'edgecolor',        'flat',...
    'facecolor',        'none',...
    'facevertexcdata',  NaN(1,3),...
    'linestyle',        '--',...
    'hittest',          'off');



% major spokes
api.hspokemajor = copyobj(api.hspokeminor,api.hrest);
set(api.hspokemajor,'linestyle','-');
```

```
% primary spoke
api.hspoke1 = copyobj(api.hspokeminor,api.hrest);
set(api.hspoke1,'linestyle','-','linewidth',2,'edgecolor',api.clrA);



% major intersections
api.hcross = patch(...
    'parent',              api.hrest,...
    'vertices',            NaN(1,2),...
    'faces',               1,...
    'edgecolor',           'none',...
    'facecolor',           'none',...
    'markerfacecolor',     'flat',...
    'markeredgecolor',     'flat',...
    'marker',              'o',...
    'markersize',          6,...
    'facevertexcdata',     NaN(1,3),...
    'hittest',             'off');

% primary intersection
api.hcross1 = copyobj(api.hcross,api.hrest);
set(api.hcross1,'marker','s');

 %% @ 'Magnitude Viewer'

% copy intersection object to magnitude display
api.hcrossmag    = copyobj(api.hcross, api.hmag);
api.hcross1mag = copyobj(api.hcross1,api.hmag);

% minor intersections
api.hcrossmagnitude = patch(...
    'parent',              api.hmag,...
    'vertices',            NaN(1,2),...
    'faces',               1,...
    'edgecolor',           'none',...
    'facecolor',           'none',...
    'markerfacecolor',     'yellow',...
    'markeredgecolor',     'yellow',...
    'marker',               'o',...
    'markersize',          2,...
    'facevertexcdata',   NaN(1,3),...
    'hittest',              'off');

api.hcross1magnitude = patch(...
    'vertices',             NaN(1,2),...
    'faces',                 1,...
    'edgecolor',           'none',...
    'facecolor',           'none',...
    'markerfacecolor',   'red',...
    'markeredgecolor',   'red',...
    'marker',               'o',...
    'markersize',          1,...
    'facevertexcdata',   NaN(1,3),...
    'hittest',              'off');




% interactive points
Npt = 2;

api.hpoint = NaN(Npt,1);
api.constrainFcn = cell(Npt,1);
for k = 1:Npt
    api.hpoint(k) = line(...
        'parent',         api.hrest,...
        'color',           api.clrP,...
        'marker',          'o',...
        'markersize',    15,...
        'linewidth',      3);

    pb = struct(...
        'enterFcn',       @(varargin)pointEnter(api.hpoint(k),api.hfig),...
        'traverseFcn',   [],...
        'exitFcn',         @(varargin)pointExit(api.hpoint(k),api.hfig));
    iptSetPointerBehavior(api.hpoint(k),pb);

    set(api.hpoint(k),'ButtonDownFcn',...
        @(varargin)pointDrag(api.hpoint(k),api.hfig));
end



% SHORT AXIS INITIALIZATION-------------------------------
if strcmpi(api.Type,'SA')

    % redraw function
    api.redrawFcn = @redrawSA;

    % constraint functions
      % rest_endo:
    api.constrainFcn{1} = @(varargin)constrainInContour(...
        varargin{:},api.RestingContour{2});
      % rest_epi:
    api.constrainFcn{2} = @(varargin)constrainOnContour(...
        varargin{:},api.RestingContour{1});

    % auotmated point locations
    pos = api.autoorigin;
    set(api.hpoint(1),'xdata',pos(1),'ydata',pos(2),'hittest','off');
    pos = api.RestingContour{1}(1,:);
    set(api.hpoint(2),'xdata',pos(1),'ydata',pos(2));

    % inputted point locations
    if ~isempty(api.PositionA)
        pos = api.constrainFcn{1}(api.hpoint(1),api.PositionA,api);
        set(api.hpoint(1),'xdata',pos(1),'ydata',pos(2),'hittest','on');
        set(api.huser,'value',1);
    end

    if ~isempty(api.PositionB)
        pos = api.constrainFcn{2}(api.hpoint(2),api.PositionB,api);
        set(api.hpoint(2),'xdata',pos(1),'ydata',pos(2));
    end

    % Nmodel popup
    str = {'6 segments','4 segments'};
```

```
        val = find(api.Nmodel==[6,4]);
        set(api.hmodel,'String',str,'Value',val);

        % Nseg popup
        seg = [1,api.Nmodel:api.Nmodel:api.MaxSegments];
        val = find(api.Nseg==seg);
        set(api.hnseg,'String',num2cell(seg),'Value',val);

        % note to user
        note = sprintf('%s\n','1.Adjust the PURPLE POSITIONS: Place the movable dot at the most anterior RV insertion point at EndFrame & Startframe. Then Segments are numbered starting from here','2.MajorTicks=BigDots(Interval=10Seg) & MinorTicks=SmallDots(Interval=1Seg)','3.Report Startframe: where cavity starts to show up','4.Report ES frame No.');
        note2 = sprintf('%s\n','Accord. to Computational Contours through a cardiac cycle, report BAD SEGMENTS CCW or CW (the direction you pick here).','Rules of reported values for Seg. A:','1.COMMA(,): separate Starting# from Ending# of a region. 2.SEMICOLON(;): separate different regions.','Report Seg. B on the most inferior RV insertion point at EndFrame & Startframe.','All Reported Values >= 0 (EX for two bad regions: "1,2;10,15")');


    % LONG AXIS INITIALIZATION--------------------------------
    else

        % connector line
        pos = [api.RestingContour{1}(1,:); api.RestingContour{2}(end,:);
               NaN,NaN;
               api.RestingContour{2}(1,:); api.RestingContour{1}(end,:)];
        hconnect = line('parent',api.hrest,...
            'xdata',pos(:,1),'ydata',pos(:,2),...
            'color','b','hittest','off');
        api.hconnect = copyobj(hconnect,api.hmag);

        % redraw function
        api.redrawFcn = @redrawLA;

        % constraint functions
        api.constrainFcn{1} = @(varargin)...
            constrainInContour(varargin{:},api.RestingContour{2});
        api.constrainFcn{2} = @(varargin)...
            constrainInContour(varargin{:},api.RestingContour{2});
```

```
% primary location
C    = api.RestingContour{2};
d1 = (C(1,1)-C(:,1)).^2 + (C(1,2)-C(:,2)).^2;
d2 = (C(end,1)-C(:,1)).^2 + (C(end,2)-C(:,2)).^2;
[val,idx] = max(d1+d2);
posA = C(idx,:);

% secondary location
posB = mean(api.RestingContour{2}([1 end],:));

% automated point locations
fac = [0.01 0.75];
pos = NaN(2,2);
for k = 1:2
    pos(k,:) = fac(k)*(posB-posA) + posA;
end
set(api.hpoint,{'xdata'},num2cell(pos(:,1)),...
    {'ydata'},num2cell(pos(:,2)));

% inputted point locations
if ~isempty(api.PositionA)
    pos = api.constrainFcn{1}(api.hpoint(1),api.PositionA,api);
    set(api.hpoint(1),'xdata',pos(1),'ydata',pos(2));
end

if ~isempty(api.PositionB)
    pos = api.constrainFcn{2}(api.hpoint(2),api.PositionB,api);
    set(api.hpoint(2),'xdata',pos(1),'ydata',pos(2));
end

% Nmodel popup
set(api.hmodel,'String',{'SA:n slices=LA:(2n+1)segments'},...
    'Value',1,'enable','off');

% Nseg popup
seg = [1,(api.Nmodel+1):api.Nmodel:api.MaxSegments];
val = find(api.Nseg==seg);
set(api.hnseg,'String',num2cell(seg),'Value',val,'enable','on');
```

```
        % additional control setup
        set(api.huser,'value',1);
        set(api.hauto,'enable','off');

        % note to user
        note = sprintf('%s','Adjust the purple positions. ',...
                'The orange lines define the first segment. ',...
                'Segments are numbered from [1:3:5:7:...] ',...
                'in the direction specified.');
end



% ADDITIONAL SETUP--------------------------------------

% display note
[note,pos] = textwrap(api.hnote,{note});
set(api.hnote,'string',note);
[note2,pos] = textwrap(api.hnote2,{note2});
set(api.hnote2,'string',note2);

% Clockwise setup
if api.Clockwise
        set(api.hclock,'value',1);
else
        set(api.hcounterclock,'value',1);
end



% setup callbacks
set(api.hmodel,'Callback',...
        @(varargin)modelCallback(api.hfig));
set(api.hnseg,'Callback',...
        @(varargin)nsegCallback(api.hfig));
set(api.hclockpanel,'SelectionChangeFcn',...
        @(src,evnt)clockCallback(api.hfig,evnt));
set(api.horiginpanel,'SelectionChangeFcn',...
        @(src,evnt)originCallback(api.hfig,evnt));
set(api.hok,'Callback',...
@(varargin)okCallback(api.hfig,get(api.hsegA,'String'),get(api.hsegB,'String'),str2double(get(api.hStartFrame,'String')),str2double(get
(api.hESframe,'String'))));

    set(api.hcancel,'Callback',...
        @(varargin)cancelCallback(api.hfig));
    set(api.hfig,'CloseRequestFcn',...
        @(varargin)figCloseRequestFcn(api.hfig));

    % initialize pointer manager
    iptPointerManager(api.hfig,'enable');

    % initialize playbar
    api.hplaybar = playbar(api.hppanel);
    api.hplaybar.Min = api.FramesForAnalysis(1);
    api.hplaybar.Max = api.FramesForAnalysis(2);
    hlisten_playbar = addlistener(api.hplaybar,...
        'NewValue',@(varargin)playbackFcn(api.hfig));

    % position playbar
    pos     = getpixelposition(api.hplaybar.Parent);
    plsz    = [200 30];
    margin = 5;
    p = [(pos(3)+1)/2 - (plsz(1)+1)/2, 1+margin, plsz];
    setpixelposition(api.hplaybar,p);

    % update figure
    api.cross = [];
    guidata(api.hfig,api);
    api.redrawFcn(api.hfig);



    % WAIT & CLEANUP-----------------------------------------

    % wait for figure to finish
    waitfor(api.hfig,'userdata')

    % cleanup
    delete(hlisten_playbar);
    % output
    if ~ishandle(api.hfig) || ~isequal(get(api.hfig,'userdata'),'complete')
            % CANCEL/FIGURECLOSE BUTTON CALLBACKS
        options = [];
    else
            tmp = get(api.hsegA,'String'); tmp = regexpi(tmp,',|;','split');
            segA = cellfun(@str2double,tmp); % segA = unique(segA);
        segB = str2double(get(api.hsegB,'String'));
            api = guidata(api.hfig);
            propA = get(api.hpoint(1),{'xdata','ydata'});
            propB = get(api.hpoint(2),{'xdata','ydata'});
            options = struct(...
                'Nmodel',           api.Nmodel,...
                'Nseg',             api.Nseg,...
                'Clockwise',        api.Clockwise,...
                'StartFrame',       str2double(get(api.hStartFrame,'String')),...
                'ESframe',      str2double(get(api.hESframe,'String')),...
                'Strains3D',        get(api.hStrains3D,'Value'),...
                'PositionA',        cat(2,propA{:}),...
                'PositionB',        cat(2,propB{:})); %,...
                % 'SegDistribution',       ({segA,segB,segC}),...
                % 'SegDistribution',       api.SegDistribution,...
                % 'Strains3D',       api.Strains3D);
            options.SegDistribution = {segA,segB};
     end

end
```

Appendix G – Callback Function of "OK" Button of Meshing Module in Matlab

```
function okCallback(hfig,segA,segB,StartFrame,ESframe)
        if ~isempty(segB)
                segB = str2double(segB);
                if isnan(segB) || segB<=0
                        msgbox('Reported values of Seg. B must be Numerical & Positive!','Error');
                        return;
                end
        end
        if ~isempty(segA)
                segA = regexpi(segA,',|;','split'); segA = cellfun(@str2double,segA);
                if sum(isnan(segA))
                        msgbox('Reported values of Seg. A must be Numerical!','Error');
                        return;
                end
                if sum(segA<=0)
                        msgbox('Reported values of Seg. A must be Positive!','Error');
                        return;
                end
                if sum(floor(segA)~=segA)
                        msgbox('The number of reported values of Seg. A must be Integers!','Error');
                        return;
                end
                if mod(numel(segA),2)~=0
                        msgbox('The number of reported values of Seg. A must be Even!','Error');
                        return;
                end
                segA = reshape(segA,2,[])'; tmp = [segA(:,1), segA(:,1)];
                if sum((segA - tmp) < 0)
                        msgbox('For a reported region: Starting# <= Ending#!','Error');
                        return;
                end
        end

        if isnan(StartFrame) || isnan(ESframe) || StartFrame<1 || StartFrame>ESframe
                msgbox('Reported numbers of StartFrame or ESframe must be numerical! Or the rank of report values must be
satisfied!','Error');
                return;
        end

      set(hfig,'userdata','complete');
end
```

Appendix H – Sector Partition Function of Meshing Module in Matlab

```
function redrawSA(hfig)


    % gather guidata
    api = guidata(hfig);


    % drawing parameters
    N        = api.Nmodel;
    Ntotal = api.Nseg;
    if Ntotal<N, Ntotal=N; end


    flag_clockwise = api.Clockwise;
    fac = Ntotal/N;
    C0 = api.RestingContour;


    % current origin
    prop = get(api.hpoint(1),{'xdata','ydata'});
    origin = cat(2,prop{:});


    % current spoke location
    prop = get(api.hpoint(2),{'xdata','ydata'});
    pos = cat(2,prop{:});



    % primary spoke angle
    theta0 = atan2(pos(2)-origin(2),pos(1)-origin(1));


    % minor spoke angles
    if flag_clockwise
        theta = linspace(0,2*pi,Ntotal+1);
    else
        theta = linspace(2*pi,0,Ntotal+1);
    end
    theta = theta(1:end-1) + theta0;


    % spoke vertices/faces/colors
    lim = api.drng;
    rad = 2*max([lim(2)-lim(1),lim(4)-lim(3)]);
```

```
[x,y] = pol2cart(theta,rad);
x = x(:)+origin(1);
y = y(:)+origin(2);

vertices = [origin; x,y];
faces    = [ones(1,Ntotal); 2:Ntotal+1]';

fvcd = [NaN(1,3); ones(Ntotal,1)*api.clrB];
fvcd([2 2+fac],:) = ones(2,1)*api.clrA;

% update spoke appearance
idxmajor = 1:fac:Ntotal; % fac = Ntotal/N;
idxminor = setdiff(1:Ntotal,idxmajor);
faces = {faces(1,:); faces(idxmajor,:); faces(idxminor,:)};

set([api.hspoke1,api.hspokemajor,api.hspokeminor],...
    'vertices',vertices,{'faces'},faces(:),'facevertexcdata',fvcd);


% remove too many-spokes
if Ntotal > 36
    set(api.hspokeminor,'visible','off');
else
    set(api.hspokeminor,'visible','on');
end

% determine intersections of major spokes with contours
vertices = repmat({NaN(N,2)},[2 1]);
pts = [origin; x(idxmajor),y(idxmajor)];
for ck = 1:2
    for sk = 1:N
        idx = [1 sk+1];
        [xi,yi] = intersections(pts(idx,1),pts(idx,2),...
            C0{ck}(:,1),C0{ck}(:,2));
        if ~isempty(xi)
            vertices{ck}(sk,:) = [xi(1),yi(1)];
        end
    end
end
```

```
% check if origin is on the endocardial border
[tfin,tfon] = inpolygon(origin(1),origin(2),C0{2}(:,1),C0{2}(:,2));
if tfon
    vertices{2}(:) = NaN;
end

% intersection vertices/faces/colors
vertices = cat(1,vertices{:});
faces = (1:2*N)';

fvcd = [ones(2,1)*api.clrA; ones(N-2,1)*api.clrB];
fvcd = [fvcd; fvcd];

% update major intersections
idx1 = [1 N+1];
idx    = setdiff(1:2*N,idx1);
set([api.hcross1,api.hcross],...
    'vertices',vertices,...
    {'faces'}, {faces(idx1,:); faces(idx,:)},...
    'facevertexcdata',fvcd);

% save intersections to api
api.cross = vertices;
api.faces = faces;
api.fvcd    = fvcd;

  % determine intersections of minor spokes with contours
vertices = repmat({NaN(Ntotal,2)},[2 1]);
pts = [origin; x(1:Ntotal),y(1:Ntotal)];
for ck = 1:2
    for sk = 1:Ntotal
        idx = [1 sk+1];
        [xi,yi] = intersections(pts(idx,1),pts(idx,2),...
            C0{ck}(:,1),C0{ck}(:,2));
        if ~isempty(xi)
            vertices{ck}(sk,:) = [xi(1),yi(1)];
        end
    end
```

```
end

% check if origin is on the endocardial border
[tfin,tfon] = inpolygon(origin(1),origin(2),C0{2}(:,1),C0{2}(:,2));
if tfon
    vertices{2}(:) = NaN;
end

% intersection vertices/faces/colors
vertices = cat(1,vertices{:});
faces = (1:2*Ntotal)';

fvcd = [ones(2,1)*api.clrA; ones(Ntotal-2,1)*api.clrB];
fvcd = [fvcd; fvcd];

% update minor intersections
idx1 = [1 Ntotal+1];
idx  = setdiff(1:2*Ntotal,idx1);
set([api.hcross1magnitude,api.hcrossmagnitude],...
    'vertices',vertices,...
    {'faces'},{faces(idx1,:); faces(idx,:)},...
    'facevertexcdata',fvcd);

% save intersections to api
api.crosslite = vertices;
api.faceslite = faces;
api.fvcdlite  = fvcd;

% update & playback
guidata(api.hfig,api);
playbackFcn(api.hfig);

% update OK button

if isempty(api.cross(:)) || any(isnan(api.cross(:)))
    set(api.hok,'enable','off');
else
    set(api.hok,'enable','on');
end
```

```
end
```

Appendix I – Playback Function of Playbar of Meshing Module in Matlab

```
function playbackFcn(hfig)

    % gather gui data
    api = guidata(hfig);

    % current frame
    fr = api.hplaybar.Value;

    % update image & contours
    set(api.him,'cdata',api.Mag(:,:,fr));
    set(api.hep,'xdata',api.Contour{fr,1}(:,1),...
        'ydata',api.Contour{fr,1}(:,2));
    set(api.hen,'xdata',api.Contour{fr,2}(:,1),...
        'ydata',api.Contour{fr,2}(:,2));

    % update major spoke intersections
    if ~isempty(api.cross)

        N = size(api.cross,1)/2;
        tf = ~any(isnan(api.cross),2);
        if ~any(tf)
            vertices = NaN(2*N,2);
        else
            dx = NaN(1,2*N);
            dy = NaN(1,2*N);

            pts = api.cross(tf,:)';
            pts(3,:) = fr;
            dx(tf) = fnvalmod(api.spldx,pts([2 1 3],:));
            dy(tf) = fnvalmod(api.spldy,pts([2 1 3],:));

            vertices = api.cross + [dx(:),dy(:)];
        end

        idx1 = [1 N+1];
        idx    = setdiff(api.faces,idx1);
        set([api.hcross1mag,api.hcrossmag],...
        'vertices',          vertices,...
        {'faces'},           {api.faces(idx1,:);
                                 api.faces(idx,:)},...
        'facevertexcdata',   api.fvcd);

end

  % update minor spoke intersections
if ~isempty(api.crosslite)

    N = size(api.crosslite,1)/2;
    tf = ~any(isnan(api.crosslite),2);
    if ~any(tf)
        vertices = NaN(2*N,2);
    else
        dx = NaN(1,2*N);
        dy = NaN(1,2*N);

        pts = api.crosslite(tf,:)';
        pts(3,:) = fr;
        dx(tf) = fnvalmod(api.spldx,pts([2 1 3],:));
        dy(tf) = fnvalmod(api.spldy,pts([2 1 3],:));

        vertices = api.crosslite + [dx(:),dy(:)];
    end

    idx1 = [1 N+1];
    idx   = setdiff(api.faceslite,idx1);
    set([api.hcross1magnitude,api.hcrossmagnitude],...
        'vertices',          vertices,...
        {'faces'},           {api.faceslite(idx1,:);
                                 api.faceslite(idx,:)},...
        'facevertexcdata',   api.fvcdlite);

end

% update connector line
if strcmpi(api.Type,'LA')
    pos = [api.Contour{fr,1}(1,:); api.Contour{fr,2}(end,:); NaN,NaN;
```

```
                        api.Contour{fr,2}(1,:); api.Contour{fr,1}(end,:)];
                set(api.hconnect,'xdata',pos(:,1),'ydata',pos(:,2));
        end

    end
```

Appendix J – Main Function of DENSE3D in Matlab

```
function self = DENSE3D(varargin)
    % DENSE3D - Object for computing 3D mechanics from DENSE
    %
    % USAGE:
    %     obj = DENSE3D(files, param/value)
    %
    % INPUTS:
    %     files:        Cell Array, cell array containing the paths to
    %                   all MAT files for analysis (from DENSEanalysis)
    %
    %     The Following parameter/value pairs are allowed:
    %
    %     Output:       String, Path to a MAT file to store the results
    %     Layers:       Integer, Indicates how many transmural layers
    %                   (Default = 5)
    %     Interp:       Integer, Amount of interpolation to use in z
    %                   direction (Default = 1)
    %     Regions:      Integer, indicates how many elements we want
    %                   circumferentially (Default = 66)
    %     Type:         Function Handle or String, Type of RBF
    %                   interpolation to use (Default = 'Linear')
    %     Constant:     Scalar, RBF Constant (Default = 0)
    %     Smooth:       Scalar, RBF Smoothing Factor (Default = 0)
    %     Insertion:    Matrix/String, 'Base' or 'Mid' or 'Apex' or 'Whole' or 'Average' or n-by-2 Matrix of specific Insertion
Points(Default = blank)
    %     SegDistro:    String 'Average' OR 1-by-2 Cell Array: 1-by-2n Matrix of BadSegDistribution + Inferior RV Insert Pt(Default
= blank)
    %     Bold:    If it's 0, only those from SOI will be ignored; If it's 1, bad segments from SOI, slice above & slice below will be
ignored; Otherwise bad segments from all silces will be ignored (by Default)


    ip = inputParser();
    ip.addRequired('Files', @(x)ischar(x) || iscell(x));
    ip.addParamValue('Output', '',    @(x)ischar(x));
    ip.addParamValue('Layers', 3,    @(x)isscalar(x));
    ip.addParamValue('Interp', 1,    @(x)isscalar(x)); %&& isinteger(x));
    ip.addParamValue('Regions', 60,    @(x)isscalar(x)); %&& isinteger(x));
    ip.addParamValue('Type', RBFInterpolator.LINEAR, @(x)ischar(x) || isa(x, 'function-Handle'));
ip.addParamValue('Constant', 0, @(x)isscalar(x));
ip.addParamValue('Smooth', 0, @(x)isscalar(x) && x >= 0);
ip.addParamValue('Insertion', ''); %,   @(x)ischar(x));
ip.addParamValue('SegDistro', {});
% ip.addParamValue('Bold', true,   @(x)islogical(x));
ip.addParamValue('Bold', 2,   @(x)isscalar(x));
ip.parse(varargin{:});

res = ip.Results;

if isa(res.Type, 'function_handle')
        rbf.Type      = res.Type;
else
        rbf.Type      = RBFInterpolator.(upper(res.Type));
end

rbf.Constant      = res.Constant;
rbf.Smooth        = res.Smooth;

% Specified Data to be loaded from DENSE2D:
fields = {'ImageInfo',...
                'AnalysisInfo',...
                'SequenceInfo',...
                'TransmuralStrainInfo',...
                'ROIInfo'};

data = cellfun(@(x)load(x, fields{:}, '-mat'), res.Files);
nPartitions = numel(data);

% recognize Input Files type: SA or LA?
nSA = repmat(false,1,nPartitions); nLA = repmat(false,1,nPartitions);
for ii = 1:nPartitions
        if ~isempty(strfind(data(ii).SequenceInfo(1,1).DENSEanalysisName, 'SA'))
                nSA(ii) = true;
        elseif ~isempty(strfind(data(ii).SequenceInfo(1,1).DENSEanalysisName, 'LA'))
                nLA(ii) = true;
        else
                error(sprintf('%s:invalidInput',mfilename),'%s',...
                        'Invalid Input #',sprintf('%d',ii),': ',sprintf('%s',data(ii).SequenceInfo(1,1).DENSEanalysisName),'. Cannot
recognize the type for SA or LA.');
                return
            end
        end

        if sum(nLA) > sum(nSA)
            data = data(nLA);
        else
            data = data(nSA);
        end
        % clear nSA, nLA;
        nPartitions = numel(data);

        locs = cellfun(@(x)x(1).SliceLocation, {data.SequenceInfo});

        [~, ind] = sort(locs, 'descend');

        data = data(ind);

        iminfo    = cat(1, data.ImageInfo);
        aninfo    = cat(1, data.AnalysisInfo);
        sqinfo    = cat(3, data.SequenceInfo);
        sqinfo    = squeeze(sqinfo(1,1,:));
        rois      = cat(1, data.ROIInfo);

        doflip = false;

        % Compute the RV insertion point for Septum
        switch lower(res.Insertion)
            case 'mid'
                if mod(nPartitions,2) == 0
                flag_aveStrain = false;
                ind = nPartitions/2;
                insertion = mean(vertcat(aninfo(ind).PositionB,aninfo(ind+1).PositionB), 1);
                if mod(res.Interp,2) == 0
                res.reportslice = ind + res.Interp/2*(1/(res.Interp+1));
                else
                msgbox('When #ofSlices is even and ROI is mid-ventricle: DENSE3D Input "Interp" must be even!','Error');
```

```
return
end
switch res.Bold
case 0
SegDistro = vertcat(aninfo(ind).SegDistribution,aninfo(ind+1).SegDistribution);
case 1
if nPartitions > 4
    SegDistro = vertcat(aninfo(ind-1).SegDistribution,aninfo(ind).SegDistribution,aninfo(ind+1).SegDistribution,aninfo(ind+2).SegDistribution);
else
    SegDistro = vertcat(aninfo.SegDistribution);
end
otherwise
SegDistro = vertcat(aninfo.SegDistribution);
end
clear tmp;
tmp{1} = [SegDistro{:,1}]; tmp{1}(isnan(tmp{1})) = [];
tmp{2} = round(mean([aninfo(ind).SegDistribution{2},aninfo(ind+1).SegDistribution{2}]));
SegDistro = tmp;
else
flag_aveStrain = true;
ind = round(nPartitions/2);
insertion = aninfo(ind).PositionB;
res.reportslice = [ind-1/(res.Interp+1), ind];
switch res.Bold
case 0
SegDistro = aninfo(ind).SegDistribution;
case 1
if nPartitions > 3
    SegDistro = vertcat(aninfo(ind-1).SegDistribution,aninfo(ind).SegDistribution,aninfo(ind+1).SegDistribution);
else
    SegDistro = vertcat(aninfo.SegDistribution);
end
otherwise
SegDistro = vertcat(aninfo.SegDistribution);
end
clear tmp;
tmp{1} = [SegDistro{:,1}]; tmp{1}(isnan(tmp{1})) = [];
```

```
        tmp{2} = aninfo(ind).SegDistribution{2};
        SegDistro = tmp;
        end
case 'base'
        insertion = aninfo(1).PositionB;
        res.reportslice = 1;
        switch res.Bold
        case 0
                SegDistro = aninfo(1).SegDistribution;;
        case 1
                if nPartitions > 2
                        SegDistro = vertcat(aninfo(1).SegDistribution,aninfo(2).SegDistribution);
                else
                        SegDistro = vertcat(aninfo.SegDistribution);
                end
        otherwise
                SegDistro = vertcat(aninfo.SegDistribution);
        end
        clear tmp;
        tmp{1} = [SegDistro{:,1}]; tmp{1}(isnan(tmp{1})) = [];
        tmp{2} = aninfo(1).SegDistribution{2};
        SegDistro = tmp;
case 'apex'
        insertion = aninfo(end).PositionB;
        SegDistro = aninfo(end).SegDistribution;
        res.reportslice = nPartitions-1/(res.Interp+1);
        switch res.Bold
        case 0
                SegDistro = aninfo(end).SegDistribution;
        case 1
                if nPartitions > 2
                        SegDistro = vertcat(aninfo(end-1).SegDistribution,aninfo(end).SegDistribution);
                else
                        SegDistro = vertcat(aninfo.SegDistribution);
                end
        otherwise
                SegDistro = vertcat(aninfo.SegDistribution);
        end
        clear tmp;
```

```
                    tmp{1} = [SegDistro{:,1}]; tmp{1}(isnan(tmp{1})) = [];
                    tmp{2} = aninfo(end).SegDistribution{2};
                    SegDistro = tmp;
                case 'average'
                % Compute the average RV insertion point
                    insertion = mean(vertcat(aninfo.PositionB), 1);
                    SegDistro = vertcat(aninfo.SegDistribution);
                    clear tmp;
                    tmp{1} = [SegDistro{:,1}]; tmp{1}(isnan(tmp{1})) = [];
                    tmp{2} = max([SegDistro{:,2}]);
                    SegDistro = tmp;
                    res.reportslice = 1 : (1/(res.Interp+1)) : nPartitions-1/(res.Interp+1);
                case ''
                    insertion = mean(vertcat(aninfo.PositionB), 1);
                    res.reportslice = [];
                otherwise
                    if isnumeric(res.Insertion)
                        [~,col] = size (res.Insertion);
                        if col ~= 2
                            error(sprintf('%s:invalidInput',mfilename),'%s','Incorrect size of inputed ParamValue Insertion: Row
Number',sprintf(' %d',col),' is odd!');
                            return
                        end
                        insertion = mean(res.Insertion, 1);
                        if isfield(aninfo, 'SegDistribution')
                            SegDistro = vertcat(aninfo.SegDistribution);
                            clear tmp;
                            tmp{1} = [SegDistro{:,1}]; tmp{1}(isnan(tmp{1})) = [];
                            tmp{2} = max([SegDistro{:,2}]);
                            SegDistro = tmp;
                            res.reportslice = 1 : (1/(res.Interp+1)) : nPartitions-1/(res.Interp+1);
                        else
                            res.reportslice = [];
                        end
                    else
                        insertion = mean(vertcat(aninfo.PositionB), 1);
                        res.reportslice = [];
                    end
        end
```

```
    % clear aninfo;

    if ~isempty(res.SegDistro)
        if ischar(res.SegDistro)
            % if strcmp(res.SegDistro,'average')
            switch lower(res.SegDistro)
                case 'average'
                    SegDistro = {NaN, round(res.Regions/4)};
                case 'whole'
                    SegDistro = {NaN, SegDistro{2}};
                % otherwise
            end
        else
            if numel(res.SegDistro) ~= 2 %|| cellfun(@floor,res.SegDistro) ~= res.SegDistro
                error(sprintf('%s:invalidInput',mfilename),'%s','Incorrect     size     of     inputed     ParamValue
SegDistro:',sprintf(' %s',res.SegDistro),'--it should be 2!');
                return
            else
                SegDistro = res.SegDistro;
            end
        end
    end

    cons = cat(3, rois.Contour);
    resting = squeeze(cons(1,:,:))';

    orientation = [1 0 0 0 1 0];

    con3D = resting;


    for slice = 1:nPartitions
        IPP = [0 0 sqinfo(slice).SliceLocation];

        rest_endo = resting{slice,2};
        rest_epi = resting{slice,1};

        [X,Y,Z] = im2world(rest_endo(:,1), rest_endo(:,2),...
                                        IPP, orientation, sqinfo(slice).PixelSpacing);
        con3D{slice,2} = [X(:), Y(:), Z(:)];

        [X,Y,Z] = im2world(rest_epi(:,1), rest_epi(:,2),...
                                        IPP, orientation, sqinfo(slice).PixelSpacing);

        con3D{slice,1} = [X(:), Y(:), Z(:)];

    end
    IPP = [0 0 sqinfo(1).SliceLocation];
    [insertion(1),insertion(2),insertion(3)] = im2world(insertion(1), insertion(2), IPP, orientation, sqinfo(1).PixelSpacing);

    % Compute resting contour & indice:
    contype = rois(1).ROIType;
    msh = contour2patch(con3D(:,2), con3D(:,1), ...
                        contype, ...
                        res.Regions, ...
                        res.Interp, ...
                        res.Layers, ...
                        insertion(1:2));
    % tmp = [fieldnames(msh)' fieldnames(tmp)'; struct2cell(msh)' struct2cell(tmp)'];
    % msh=struct(tmp{:});

    minframe = max(arrayfun(@(x)x.FramesForAnalysis(1), aninfo));
    maxframe = min(arrayfun(@(x)x.FramesForAnalysis(2), aninfo));

    frames = minframe:maxframe;

    % initialize RBF:
    nFrames = numel(frames);
    RBF = cell(nFrames, 1);

    for frame = minframe:maxframe
        disps = zeros(0, 3);
        points = zeros(0, 3);

        for slice = 1:nPartitions
            Xunwrap = iminfo(slice).Xunwrap(:,:,frame);
            Yunwrap = iminfo(slice).Yunwrap(:,:,frame);
            if nPartitions == 1
                Zunwrap = zeros(size(iminfo(slice).Zunwrap(:,:,frame)));
            else
                Zunwrap = iminfo(slice).Zunwrap(:,:,frame);
            end

            seq = sqinfo(slice);

            % Find Array elements of Xunwrap that are NOT NaN
            valid = ~isnan(Xunwrap);

            % Find #ofRow & #ofCol of Array elements of Xunwrap that are NOT NaN
            [row, col] = find(valid);

            tmppos = [0 0 seq.SliceLocation];

            % Get world coordinates of all unwrap voxel at time=0:
            [X,Y,Z] = im2world(col, row, tmppos, orientation, seq.PixelSpacing);

            points = cat(1, points, [X(:), Y(:), Z(:)]);

            % Convert displacements to mm
            dx = Xunwrap(valid) .* iminfo(slice).Multipliers(1) .* seq(1).PixelSpacing(2);
            dy = Yunwrap(valid) .* iminfo(slice).Multipliers(2) .* seq(1).PixelSpacing(1);
            dz = Zunwrap(valid) .* iminfo(slice).Multipliers(3) .* seq(1).PixelSpacing(1);

            if doflip
                dz = -dz;
            end

            disps = cat(1, disps, [dx(:), dy(:), dz(:)]);
        end

        % Get world coordinates of all voxel at time=frame:
        points = points - disps;
        RBF{frame} = RBFInterpolator(points, disps, rbf);
    end
```

```
        % Deform meshes using the Lagrangian displacements
        vert = msh.vertices;

        defmesh = repmat(msh, [max(frames) + 1, 1]);

        for frame = frames
                dX = RBF{frame}.query(vert);
                defmesh(frame+1).vertices = vert + dX;
        end

        % Perform fourier smoothing
        FF = fourierfit([0 frames], cat(3, defmesh.vertices), 5);

        smoothmesh = defmesh;

        for k = 1:size(FF, 3)
                smoothmesh(k).vertices = squeeze(FF(:,:,k));
        end

        % Compute strain components for each element:
        inp = struct('faces',           {msh.faces},...
                        'vertices',     {msh.vertices},...
                        'time',         frames,...
                        'RBF',          {RBF});

        msh.strains = hexahedralstrain(inp);

        % Compute strain components region by region:
        if ~exist('SegDistro', 'var') || isnan(SegDistro{2})
                msgbox('No Regional Strains Componets are computed!','Warning');
        else
                fields = {'XX', 'XY', 'XZ', 'YX', 'YY', 'YZ', 'ZX', 'ZY', 'ZZ', 'RR', 'RC', 'RL', 'CR', 'CC', 'CL', 'LR', 'LC', 'LL', 'p1', 'p2', 'p3',
'torsion', 'J'};
                nElemt = floor((res.Regions-SegDistro{2})/3);

                SegD = repmat(true,1,res.Regions);
                if ~isnan(SegDistro{1})
                        for ii = 1:2:numel(SegDistro{1})
                                SegD(SegDistro{1}(ii):SegDistro{1}(ii+1)) = false;
                        end
                end

                clear SegD2;
                tmp = repmat(false,1,res.Regions); tmp(1:SegDistro{2}) = true; SegD2{1} = tmp;
                tmp = repmat(false,1,res.Regions); tmp(SegDistro{2}+1:SegDistro{2}+nElemt) = true; SegD2{2} = tmp;
                tmp = repmat(false,1,res.Regions); tmp(SegDistro{2}+nElemt+1:res.Regions-nElemt) = true; SegD2{3} = tmp;
                tmp = repmat(false,1,res.Regions); tmp(res.Regions-nElemt+1:res.Regions) = true; SegD2{4} = tmp;

                % SegD&SegD2{ii}:
                SegDist = cellfun(@(x)SegD&x,SegD2,'UniformOutput',false);

                slice = 0;
                % for absind = 1: (1/(res.Interp+1)) : max(msh.absind)
                for absind = res.reportslice
                %Longitudinal location: base -> apex
                        % compare doubles with tolerance:
                        ind = abs(msh.absind-absind)<1e-2;
                        slice = slice + 1;
                        for layer = 1: res.Layers %Transmurally: endo -> epi
                                ind2 = (layer == msh.layerid);
                                index = ind&ind2;
                                for ii = 1:numel(fields)
                                    tmp = msh.strains.(fields{ii})(index,:);
                                    %Regionally: septal -> interior -> lateral -> anterior
                                    msh.regionalstrains.(fields{ii}){slice,layer}                                    =
[mean(tmp(SegDist{1},:));mean(tmp(SegDist{2},:));mean(tmp(SegDist{3},:));mean(tmp(SegDist{4},:))];
                                end
                        end
                        % place Average strains at last: endo -> epi + Average
                        for ii = 1:numel(fields)
                                tmp = vertcat(msh.regionalstrains.(fields{ii}){slice,:});
                                msh.regionalstrains.(fields{ii}){slice,layer+1}(1,:) = mean(tmp(1:4:end,:));
                                msh.regionalstrains.(fields{ii}){slice,layer+1}(2,:) = mean(tmp(2:4:end,:));
                                msh.regionalstrains.(fields{ii}){slice,layer+1}(3,:) = mean(tmp(3:4:end,:));
                                msh.regionalstrains.(fields{ii}){slice,layer+1}(4,:) = mean(tmp(4:4:end,:));
                        end
                end
```

```
            if exist('flag_aveStrain', 'var') && flag_aveStrain
                clear tmp tmp2;
                for jj = 1:numel(fields)
                for layer = 1:4
                for region = 1:4
                    for slice = 1:numel(res.reportslice)
                        tmp2(slice,:) = msh.regionalstrains.(fields{jj}){slice,layer}(region,:);
                    end
                    tmp.(fields{jj}){1,layer}(region,:) = mean(vertcat(tmp2),1);
                end
                end
                end
                msh.regionalstrains = tmp;
            end

            % Save results:
            msh.regionalstrains.nElemts = cellfun(@sum,SegDist);
            msgbox(strcat('Strains in',sprintf(' %d',msh.regionalstrains.nElemts),' elements are average respectively for Septal, Inferior,
lateral, Anterior wall!'),'Warning');

        end

        % Save results:
        msh.regionalstrains.SegDistro = res.SegDistro;
        msh.regionalstrains.insertion = res.Insertion;
        msh.regionalstrains.reportslice = res.reportslice;
        % msh.LAlength = res.LAlength;
        msh.Layers = res.Layers;
        msh.Interp = res.Interp;
        msh.Regions = res.Regions;
        self.Mesh = msh;
        self.Data = data;

        % Output results:
        if ~isempty(res.Output)
            save(outfile, '-struct', 'out');
        end
end
```

Appendix K - Computing Mechanics Module in Matlab

```
function strain = hexahedralstrain(api)
    % Fields
    %     faces:          faces
    %     vertices:    vertices
    %     times:          times
    %     RBF:              rbf
    %     Origin:         origin
    %     Orientation:clockwise
    face = api.faces;

    % Make sure that we do not double any vertices
    vert = api.vertices;
    time = api.time;

    if ~isfield(api,'PositionA')
        % Basically the 2D centroid
        origin = mean(vert(:,[1 2]),1);
    else
        origin = api.PositionA;
    end

    Ntime = numel(time);
    [Nface, dim] = size(face);

    % Vertex trajectories
    vtrj = NaN([size(vert), Ntime]);

    % Only check points that are associated with a face
    for k = 1:Ntime
        % Compute dispalcements using rbf(radial basis function)
        dX = api.RBF{k}.query(vert);
        vtrj(:,:,k) = vert + dX;
    end


    % Put resting configuration at the beginning
    vtrj = cat(3, vert, vtrj);
```

```
dt = time(2) - time(1);
time = [time(1) - dt, time];

% Temporal smoothing
vtrj = fourierfit(time, vtrj, 5);



% Now compute the centroid of each element by averaging vertices
pos = NaN(Nface, size(vert, 2));
for k = 1:size(vert, 2)
    tmp = vert(:,k);
    tmp = tmp(face);
    pos(:,k) = mean(tmp, 2);
end

% Now move the centroids through time
ptrj = NaN([size(pos), Ntime]);

pts = pos';

for k = 1:Ntime
    pts(3,:) = time(k);
    % compute dx, dy, dz
    dX = api.RBF{k}.query(pos);
    ptrj(:,:,k) = pos + dX;
end

% Put resting configuration at the beginning
ptrj = cat(3, pos, ptrj);

ptrj = fourierfit(time, ptrj, 5);

theta = cart2pol(pos(:,1) - origin(1), pos(:,2) - origin(2));

ct = cos(theta);
st = sin(theta);

%% Actually compute the strains %%
```

```
tmp = NaN([Nface Ntime]);

strain = struct(...
        'vertices',        vert,...
        'faces',               face,...
        'orientation',    theta,...
        'ptrj',                ptrj,...
        'vtrj',                vtrj,...
        'XX',      tmp,...
        'YY',      tmp,...
        'ZZ',      tmp,...
        'XY',      tmp,...
        'YX',      tmp,...
        'XZ',      tmp,...
        'ZX',      tmp,...
        'YZ',      tmp,...
        'ZY',      tmp,...
        'RR',      tmp,...
        'CC',      tmp,...
        'LL',      tmp,...
        'RC',      tmp,...
        'CR',      tmp,...
        'RL',      tmp,...
        'LR',      tmp,...
        'CL',      tmp,...
        'LC',      tmp,...
        'p1',      tmp,...
        'p2',      tmp,...
        'p3',      tmp,...
        'torsion',      tmp,...
        'J',      tmp);

dX = zeros([3 dim Nface]);

  % Use the Resting Contour after Temporal Fourier Smoothing to get dist:
for k = 1:Nface
        tmp = vtrj(face(k,:),:,1) - ptrj(k*ones(dim,1),:,1);
        dX(:,:,k) = tmp';
```

```
end

% strain calculation
for fr = 1:Ntime

    % Loop through each face
    for k = 1:Nface
        % difference matrix: current configuration
        tmp = vtrj(face(k,:),:,fr) - ptrj(k*ones(dim,1),:,fr);
        dx = tmp';

        Fave = dx/dX(:,:,k);

            % Jacobian
            strain.J(k, fr) = det(Fave);

        % Lagrangian x/y/z strain tensor
        E = 0.5 * (Fave' * Fave - eye(3));

        % Rotate the coordinate system
        Rot = [ct(k) st(k) 0; -st(k) ct(k) 0; 0 0 1];

        % Lagrangian radial/circumferential/longitudinal strain tensor
        Erot = Rot*E*Rot';

            % Torsion: circumferential-longitudinal (CL) shear angle
            strain.torsion(k, fr) = asind(2*Erot(6)/sqrt((1+2*Erot(5))*(1+2*Erot(9))));

        % priniciple strains
        [v,d] = eig(E, 'nobalance');

        % Record the output
        fields = {'XX', 'XY', 'XZ', 'YX', 'YY', 'YZ', 'ZX', 'ZY', 'ZZ'};

        for i = 1:numel(fields)
            strain.(fields{i})(k, fr) = E(i);
        end

        fields = {'RR', 'RC', 'RL', 'CR', 'CC', 'CL', 'LR', 'LC', 'LL'};
```

```
                for i = 1:numel(fields)
                    strain.(fields{i})(k, fr) = Erot(i);
                end

                if all(d == 0)
                    strain.p1(k,fr) = 0;
                    strain.p2(k,fr) = 0;
                    strain.p3(k,fr) = 0;
                else
                    strain.p1(k,fr) = d(end);
                    strain.p2(k,fr) = d(2,2);
                    strain.p3(k,fr) = d(1,1);
                end
            end
        end
    end
```

Appendix L – MATLAB Script for Strains and Torsion Comparison across Subjects

```
% Last Modified: 1:06 PM Thursday, October 15, 2015
% Modified By: Zhanqiu Liu (lafeir.lew@gmail.com)

%%%Matlab Run
%set currunt Dir.
Insertion = input('The Longitudinal Ventricular Location of interest[press enter for "Mid", "1" for Base, "2" for "Apex", "3" for "Whole", "4" for "Average"; or Input a n-by-2 Matrix]: ');
if isempty(Insertion)
    Insertion = 'Mid';
end
switch Insertion
    case 1
        Insertion = 'Base';
    case 2
        Insertion = 'Apex';
    case 3
        Insertion = 'Whole';
    case 4
        Insertion = 'Average';
    otherwise
        if isnumeric(Insertion)
            [~,col] = size(Insertion);
            if col ~= 2
                error(strcat('Incorrect size of inputed ParamValue Insertion: #ofColumns is',sprintf(' %d',col),'--it should be even!'));
                return
            end
        end
end
SegDistro = input('Different kinds of regional distribution of circumferential elements [press enter for ignoring bad segments(specified in DENSE2D), "1" for including bad segments, "2" for four evenly-divided regions; or Input a 1-by-2 Cell Array]: ');
if isempty(SegDistro)
    SegDistro = '';
end
switch SegDistro
    case 1
        SegDistro = 'Whole';
        case 2
                SegDistro = 'Average';
        otherwise
                if isnumeric(SegDistro)
                        if numel(SegDistro) ~= 2
                                error(strcat('Incorrect size of inputed ParamValue SegDistro: #ofColumns is',sprintf('%d',col),'--it should be
2!'));
                                return
                        end
                end
end

tmp = input('1-by-3 DENSE3D Inputs(Meshing Parameters Matrix)--"[#Segments/slice,Transmural Interp,Z Interp]"[press enter for
"[60,3,2]" by default]:');
if isempty(tmp)
        Regions = 60; Layers = 3; Interp = 2;
else
        Regions = tmp(1); Layers = tmp(2); Interp = tmp(3);
end
button = questdlg({'Do you wish to use a User Interface to select Datasets of interest?'},'Options');
if strcmp(button,'Yes')
        clear files folders;
        ii = 0;
        while true
                [uifile, uipath, uipopup] = uigetfile({'*.mat', 'Select DENSE3D workspaces (*.mat)'},'Open',pwd);
                if ~uipopup %isequal(uifile,0) || isequal(uipath,0)
                        break;
                else
                        ii = ii + 1;
                        files{ii} = fullfile(uipath,uifile);
                        tmp = strsplit(uipath, '\');
                        folders{ii} = tmp{end-1};
                end
        end
else
        % Upper level of path:
        % path = strsplit(pwd, '\'); path = {path{1:end-1}}; path = strjoin(path, '\');
        tmp = dir(pwd); folders = {tmp.name}; folders = folders([tmp.isdir] == 1); folders = {folders{3:end}};
        tmp = arrayfun(@num2str, 1:numel(folders), 'UniformOutput', false);
        tmp = strcat(tmp, '.', folders);
        ind = input(strcat([sprintf('%s\n',tmp{:}),'Input the numbers of Datasets of interest(use Space/Space/Semicolon(;) to seperate):']));
        ind = reshape(ind,[],1); ind = round(abs(ind)); ind(ind==0) = [];
        folders = folders(ind);
        files                                                                                                     =
fullfile(pwd,folders,strcat('DENSE3Dobject',Insertion,SegDistro,num2str(Regions),num2str(Layers),num2str(Interp),'.mat'));
end



nFiles = numel(files);
clear data;
for ii = 1:nFiles
        if ~exist(files{ii},'file')
                Files = readpath(fullfile(pwd,folders{ii}))
                Files = fullfile(pwd,folders{ii},Files);
                % Files = readpath()
                dbstop if error
                DENSE3Dobject                                                                                     =
DENSE3D(Files,'Layers',Layers,'Interp',Interp,'Regions',Regions,'Insertion',Insertion,'SegDistro',SegDistro)
        save(strcat(pwd,'\',folders{ii},'\','DENSE3Dobject',Insertion,SegDistro,num2str(Regions),num2str(Layers),num2str(Interp)),'DE
NSE3Dobject');
        end
        tmp = load(files{ii}); %, 'DENSE3Dobject', '-mat');
        nFrames(ii) = tmp.DENSE3Dobject.Data(1).SequenceInfo(1, 1).CardiacNumberOfImages;
        data{ii} = tmp.DENSE3Dobject.Mesh.regionalstrains;
        torsion{ii} = mean(tmp.DENSE3Dobject.Mesh.strains.torsion,1);
end



%%% 3D Regional and Transmural Strain
%%% 6 segments
regions = {'Septal','Inferior','Lateral','Anterior'};
transmural = {' Sub-endocardium',' Mid-wall',' Sub-epicardium',' Myocardium'};
fields = {'RR', 'CC', 'LL', 'RC', 'RL', 'CL'};
Marker = {'x','+','*','o','s','d','p','+','*','o','s','d','p','x'};
Color = {'b','r','k','g','y','m','c','b','r','k','g','y','m','c'};

% Validate inputed datasets:
[ind, ~] = size(data{1}.(fields{1}));
```

```
[ind2, ~] = size(data{1}.(fields{1}){1,1});
for ii = 1:nFiles
        [nSlices, nLayers] = size(data{ii}.(fields{1}));
        if nSlices ~= ind
                error(strcat('The number of Slices for each Inputed Datasets is different:',sprintf(' %d',nSlices),' NOT EQUAL TO',sprintf(' %d',ind)));
                return
        end
        [nRegions, ~] = size(data{ii}.(fields{1}){1,1});
        if nRegions > 4 || nRegions ~= ind2
                error(strcat('The number of regions for Input #',sprintf('%d',ii),' is invalid:',sprintf('%d',nRegions),'----Only support that myocardium is divided into 4 regions.'));
                return
        end
        if nLayers > 4
                if mod((nLayers-1),2)~=0
                        layer = [1,ceil((nLayers-1)/2),nLayers-1,nLayers];
                        for jj = 1:numel(fields)
                                for slice = 1:nSlices
                                        data{ii}.(fields{jj}) = data{ii}.(fields{jj})(slice,layer);;
                                end
                        end
                else
                        clear tmp2;
                        for jj = 1:numel(fields)
                                for slice = 1:nSlices
                                        tmp = vertcat(data{ii}.(fields{jj}){slice,(nLayers-1)/2},data{ii}.(fields{jj}){slice,(nLayers-1)/2+1});
                                        tmp2{slice,1} = data{ii}.(fields{jj}){slice, 1};
                                        tmp2{slice,2}(1,:) = mean(tmp(1:4:nLayers,:));
                                        tmp2{slice,2}(2,:) = mean(tmp(2:4:nLayers,:));
                                        tmp2{slice,2}(3,:) = mean(tmp(3:4:nLayers,:));
                                        tmp2{slice,2}(4,:) = mean(tmp(4:4:nLayers,:));
                                        tmp2{slice,3} = data{ii}.(fields{jj}){slice, nLayers-1};
                                        tmp2{slice,4} = data{ii}.(fields{jj}){slice, nLayers};
                                end
                                data{ii}.(fields{jj}) = tmp2;
                        end
                end
        end
```

```
end

% Normalized:
minFrame = min(nFrames);
queryPoints = [1/minFrame:1/minFrame:1];
clear normalDataH;
for ii = 1:nFiles
    if nFrames(ii) == minFrame
        normalDataH{ii} = data{ii};
        torsionH(ii,:) = torsion{ii};
    else
        samplePoints = [1/nFrames(ii):1/nFrames(ii):1];
        for jj = 1:numel(fields)
            for slice = 1:nSlices
                for layer = 1:4
                    for region = 1:nRegions
                        sampleValues = data{ii}.(fields{jj}){slice,layer}(region,:);
                        normalDataH{ii}.(fields{jj}){slice,layer}(region,:)   =   pchip(samplePoints,   sampleValues,
queryPoints);
                    end
                end
            end
        end
        torsionH(ii,:) = pchip(samplePoints, torsion{ii}, queryPoints);
    end
end

button = questdlg({'Do you wish to compute Mean+Std. Dev.?'},'Options');
if strcmp(button,'Yes')
ind = queryPoints*100;
for jj = 1:numel(fields)
    for slice = 1:nSlices
        figure('units','normalized','outerposition',[0 0 1 1],'Visible','off');
        for region = 1:nRegions
            for layer = 1:4
                subplot(nRegions,4,4*(region-1)+layer);
                xlabel('Cardiac Cycle(%)');
                ylabel(strcat(regions{region},'-',transmural{layer}));
                grid on;
```

```
                    hold on;
                    tmp = [];
                    for ii = 1:nFiles
                        tmp = [tmp; normalDataH{ii}.(fields{jj}){slice,layer}(region,:)];
                    end
                    tmp2 = mean(tmp);
                    tmp3 = std(tmp);
                    ave.(fields{jj}){slice,layer}(region,:) = tmp2;
                    stdev.(fields{jj}){slice,layer}(region,:) = tmp3;
                    plot(ind,tmp2,'LineWidth',1,'color','b','Marker','x','MarkerSize', 5);
                    plot(ind,tmp2-tmp3,'LineWidth',0.5,'color','c','Marker','+','MarkerSize', 3);
                    plot(ind,tmp2+tmp3,'LineWidth',0.5,'color','c','Marker','+','MarkerSize', 3);
                    set(gca,'XMinorTick','on','YMinorTick','on');
                end
            end
            axes('Position',[0 0 1 1],'Xlim',[0 1],'Ylim',[0 1],'Box','off','Visible','off','Units','normalized', 'clipping' , 'off');
            text(0.5, 1,{strcat(Insertion,SegDistro,' slice#',num2str(slice),'-ventricular Strain E',lower(fields{jj})),strcat('Distrib. of
Circumf. Elemt:',SegDistro,', #Segments/slice:',num2str(Regions),', Transmural Interp:',num2str(Layers),', Z
Interp:',num2str(Interp)),'Dark:Mean VS Light:±Standard Deviation'},'HorizontalAlignment','center','VerticalAlignment', 'top');

        hgexport(gcf,strcat(strjoin(folders,'_VS_'),Insertion,SegDistro,num2str(Regions),num2str(Layers),num2str(Interp),'_E',lower(fi
elds{jj}),num2str(slice)),hgexport('factorystyle'), 'Format', 'jpeg');
        end
end


else
for jj = 1:numel(fields)
        for slice = 1:nSlices
            figure('units','normalized','outerposition',[0 0 1 1],'Visible','off');
            for region = 1:nRegions
                for layer = 1:4
                    subplot(nRegions,4,4*(region-1)+layer);
                    xlabel('Cardiac Cycle(frames)');
                    ylabel(strcat(regions{region},transmural{layer}));
                    grid on;
                    hold on;
                    for ii = 1:nFiles
                        plot(normalDataH{ii}.(fields{jj}){slice,layer}(region,:),'LineWidth',1,'color', Color{ii},'Marker',
Marker{ii},'MarkerSize',5); %,'MarkerEdgeColor','b');
```

```
                    end
                    set(gca,'XMinorTick','on','YMinorTick','on');
                    % hold off;
                end
            end
            axes('Position',[0 0 1 1],'Xlim',[0 1],'Ylim',[0 1],'Box','off','Visible','off','Units','normalized', 'clipping' , 'off');
            text(0.5, 1,[strcat(Insertion,' slice#',num2str(slice),'-ventricular Strain E',lower(fields{jj})),strcat('Distrib. of Circumf. Elemt:',SegDistro,', #Segments/slice:',num2str(Regions),', Transmural Interp:',num2str(Layers),', Z Interp:',num2str(Interp)),strcat(Color(1:nFiles), Marker(1:nFiles),'.', folders)],'HorizontalAlignment','center','VerticalAlignment', 'top');

        hgexport(gcf,strcat(strjoin(folders,'_VS_'),Insertion,SegDistro,num2str(Regions),num2str(Layers),num2str(Interp),'_E',lower(fields{jj}),num2str(slice)),hgexport('factorystyle'), 'Format', 'jpeg');
        end
end

end

% Plot Torsion:
ind = queryPoints*100;
clear tmp;
figure('units','normalized','outerposition',[0 0 1 1],'Visible','off');
xlabel('Cardiac Cycle(%)');
ylabel('Circumferential-longitudinal shear angle(°/cm^2)');
grid on;
hold on;
for ii = 1:nFiles
        tmp{ii} = plot(ind,torsionH(ii,:),'LineWidth',1,'color', Color{ii},'Marker', Marker{ii},'MarkerSize',5); %,'MarkerEdgeColor','b');
end
set(gca,'XMinorTick','on','YMinorTick','on');
legend([tmp{:}], folders);
hgexport(gcf,strcat('Torsion',Insertion,SegDistro,num2str(Regions),num2str(Layers),num2str(Interp),'_E',lower(fields{jj}),num2str(slice)),hgexport('factorystyle'), 'Format', 'jpeg');
% Mean+Std. Dev.:
torsionAve = mean(torsionH);
torsionStd = std(torsionH);
figure('units','normalized','outerposition',[0 0 1 1],'Visible','off');
xlabel('Cardiac Cycle(%)');
ylabel('Circumferential-longitudinal shear angle(°/cm^2)');
grid on;
```

```
hold on;
plot(ind,torsionAve,'LineWidth',1,'color','b','Marker','x','MarkerSize', 5);
plot(ind,torsionAve-torsionStd,'LineWidth',0.5,'color','c','Marker','+','MarkerSize', 3);
plot(ind,torsionAve+torsionStd,'LineWidth',0.5,'color','c','Marker','+','MarkerSize', 3);
set(gca,'XMinorTick','on','YMinorTick','on');
hgexport(gcf,strcat('TorsionStd',Insertion,SegDistro,num2str(Regions),num2str(Layers),num2str(Interp),'_E',lower(fields{jj}),num2str
(slice)),hgexport('factorystyle'), 'Format', 'jpeg');

% Figures in Thesis:
ind = queryPoints*100;
for jj = 1:numel(fields)
    for slice = 1:nSlices
        figure('units','normalized','outerposition',[0 0 1 1],'Visible','off');
        for region = 1:nRegions
            for layer = 1:4
                subplot(nRegions,4,4*(region-1)+layer);
                title(strcat(regions{region},transmural{layer}));
                xlabel('Cardiac Cycle(%)');
                ylabel(strcat('E',lower(fields{jj})));
                grid on;
                hold on;
                tmp = [];
                for ii = 1:nFiles
                    tmp = [tmp; normalDataH{ii}.(fields{jj}){slice,layer}(region,:)];
                end
                tmp2 = mean(tmp);
                tmp3 = std(tmp);
                ave.(fields{jj}){slice,layer}(region,:) = tmp2;
                stdev.(fields{jj}){slice,layer}(region,:) = tmp3;
                plot(ind,tmp2,'LineWidth',1,'color','b','Marker','x','MarkerSize', 5);
                plot(ind,tmp2-tmp3,'LineWidth',0.5,'color','c','Marker','+','MarkerSize', 3);
                plot(ind,tmp2+tmp3,'LineWidth',0.5,'color','c','Marker','+','MarkerSize', 3);
                set(gca,'XMinorTick','on','YMinorTick','on');
            end
        end

    hgexport(gcf,strcat('Strain',Insertion,SegDistro,num2str(Regions),num2str(Layers),num2str(Interp),'_E',lower(fields{jj}),num2s
tr(slice)),hgexport('factorystyle'),'Format','jpeg');
    end
```

```
end


% Transmural trend:
frame = 9;
ind = [1:3];
for jj = 1 : 3
        for slice = 1:nSlices
                figure('units','normalized','outerposition',[0 0 1 0.5],'Visible','off');
                for region = 1:nRegions
                        tmp = []; tmp2 = [];
                        for layer = 1:3
                                tmp = [tmp,ave.(fields{jj}){slice,layer}(region,frame)];
                                tmp2 = [tmp2,stdev.(fields{jj}){slice,layer}(region,frame)];
                        end
                        subplot(1,nRegions,region);
                        title(strcat(regions{region},' Segments'));
                        xlabel('Sub-endocardium                Mid-wall                Sub-epicardium');
                        ylabel(strcat('E',lower(fields{jj})));
                        grid on;
                        hold on;
                        plot(tmp,'LineWidth',1,'color','b','Marker','x','MarkerSize', 5);
                        plot(tmp-tmp2,'LineWidth',0.5,'color','c','Marker','+','MarkerSize', 3);
                        plot(tmp+tmp2,'LineWidth',0.5,'color','c','Marker','+','MarkerSize', 3);
                        % set(gca,'XMinorTick','off','YMinorTick','on');
                        set(gca,'XTick',[1 2 3],'YMinorTick','on');
                end
                hgexport(gcf,strcat('Transmural
trend',Insertion,SegDistro,num2str(Regions),num2str(Layers),num2str(Interp),'_E',lower(fields{jj}),num2str(slice)),hgexport('factorys
tyle'), 'Format', 'jpeg');
        end
end
```

References

[1] CDC, "Underlying Cause of Death, 1999-2014 Request," Data are from the Multiple Cause of Death Files, 1999-2014, as compiled from data provided by the 57 vital statistics jurisdictions through the Vital Statistics Cooperative Program, [Online]. Available: http://wonder.cdc.gov/ucd-icd10.html. [Accessed 30 March 2016].

[2] CDC, "Underlying Cause of Death, 1999-2014 Request," Data are from the File of Cause of Death caused by heart diseases, 1999-2014, as compiled from data provided by the 57 vital statistics jurisdictions through the Vital Statistics Cooperative Program, [Online]. Available: http://wonder.cdc.gov/ucd-icd10.html. [Accessed 30 March 2016].

[3] A. U. C. J. C. J. L. D. K. T. G. W. C. S. Ford ES, "Explaining the decrease in U.S. deaths from coronary disease, 1980–2000," N Engl J Med, 2007.

[4] "Bio 100: Heart Anatomy," [Online]. Available: http://www.biosbcc.net/b100cardio/htm/heartant.htm.

[5] wikipedia, "Ventricle (heart)," [Online]. Available: https://en.wikipedia.org/wiki/Ventricle_(heart).

[6] wikipedia, "Endocardium," [Online]. Available: https://en.wikipedia.org/wiki/Endocardium.

[7] wikipedia, "Wiggers Diagram," [Online]. Available: https://en.wikipedia.org/wiki/File:Wiggers_Diagram.png.

[8] wikipedia, "Heart systole," [Online]. Available: https://en.wikipedia.org/wiki/File:Heart_systole.svg.

[9] wikipedia, "Heart diastole," [Online]. Available: https://en.wikipedia.org/wiki/File:Heart_diastole.png.

[10] Center for Teaching and Learning, Columbia, "MECHANICAL PROPERTIES OF THE HEART I & II," [Online]. Available: http://ccnmtl.columbia.edu/projects/heart/exercises/MechPropHeart/lecture.html.

[11] Association AH, "Heart disease and stroke statistics 2011 update: A report from the american heart association," 2011.

[12] Wikipedia, "Heart normal," [Online]. Available: https://en.wikipedia.org/wiki/File:Heart_normal.svg.

[13] Wikipedia, "Heart tetralogy fallot," [Online]. Available: https://en.wikipedia.org/wiki/File:Heart_tetralogy_fallot.svg.

[14] Wikipedia, "Tetralogy of Fallot," [Online]. Available: https://en.wikipedia.org/wiki/Tetralogy_of_Fallot.

[15] Warnes, Carole A., "The Adult With Congenital Heart Disease," *Journal of the American College of Cardiology,* vol. 46, no. 1, p. 1–8, July 2005.

[16] American Heart Association, American College of Cardiology, and Society of Nuclear Medicine, "Standardization of cardiac tomographic imaging," *Circulation,* vol. 86, p. 338–339, 1992.

[17] Cerqueira M, "Standardized myocardial segmentation and nomenclature for tomographic imaging of the heart: a statement for healthcare professionals from the Cardiac Imaging Committee of the Council on Clinical Cardiology of the American Heart Association," *Circulation,* vol. 105, p. 539–542, 2002.

[18] Edwards Lifesciences LLC, "Normal Hemodynamic Parameters and Laboratory Values," 2009. [Online]. Available: www.edwards.com.

[19] F. B. M. Scott H. Faro, "Functional MRI: basic principles and clinical applications," in *Spiral-Echo Planar Imaging Methods*, Springer Science & Business Media, 2006, pp. 53-54.

[20] Meyer CH, "Echo-planar imaging: theory, technique, and application," in *Spiral echo-planar imaging*, Berlin and Heidelberg, Springer Verlag, 1998, p. 633–658.

[21] S. D. R. S. B. H. W. Anthony H. Aletras, "DENSE: Displacement Encoding with Stimulated Echoes in Cardiac Functional MRI," *J Magn Reson,* vol. 137, no. 1, p. 247–252., 1999.

[22] Zhong, X., Helm, P. A., & Epstein, F. H., "Balanced multipoint displacement encoding for DENSE MRI," *Magnetic resonance in medicine,* vol. 61, no. 4, pp. 981-988, 2009.

[23] Aletras AH, Ding SJ, Balaban RS, Wen H, "DENSE: Displacement encoding with stimulated echoes in cardiac functional MRI," *Journal of Magnetic Resonance,* vol. 137, no. 1, pp. 247-252, 1999.

[24] S. P. K. C. M. B. D. K. P. A. C. M. R. C. F. H. E. B. K. F. Christopher M Haggerty, "Reproducibility of cine displacement encoding with stimulated echoes (DENSE) cardiovascular magnetic resonance for measuring left ventricular strains, torsion, and synchrony in mice," *Journal of Cardiovascular Magnetic Resonance,* vol. 15, no. 1, p. 71, 2013.

[25] X. Y. Jia Zhong, "Strain and Torsion Quantification in Mouse Hearts under Dobutamine Stimulation using 2D Multi-Phase MR DENSE," *Magn Reson Med,* vol. 64, no. 5, p. 1315–1322, November 2010.

[26] X. Y. Wei Li, "Quantification of Myocardial Strain at Early Systole in Mouse Heart: Restoration of Undeformed Tagging Grid with Single-Point HARP," *J Magn Reson Imaging,* vol. 32, no. 3, p. 608–614, Sep 2010.

[27] A. S. X. Y. J. E. S. Yong Chen, "Altered in vivo left ventricular torsion and principal strains in hypothyroid rats," *Am J Physiol Heart Circ Physiol,* vol. 299, no. 5, p. H1577–H1587, Nov 2010.

[28] J. C. S. J. J. S. A. P. V. B. S. A. W. X. Y. Wei Liu, "HARP MRI Tagging for Direct Quantification of Lagrangian Strain in Rat Hearts After Myocardial Infarction," *J Biomech Eng,* vol. 126, no. 4, p. 523–528, Aug 2004.

[29] Aaron T. Hess, Xiaodong Zhong, Bruce. S. Spottiswoode, Frederick. H. Epstein, Ernesta M. Meintjes, "Myocardial 3D strain calculation by combining cine DENSE and cine SENC imaging," *Magn Reson Med,* vol. 62, no. 1, p. 77–84, July 2009.

[30] [Online]. Available: http://www.criticalecho.com/sites/default/files/images/3.3.png.

[31] Spottiswoode BS, Zhong X, Hess AT, Kramer CM, Meintjes EM, Mayosi BM, Epstein FH., "Tracking myocardial motion from cine DENSE images using spatiotemporal phase unwrapping and temporal fitting.," *IEEE Trans Med Imaging,* vol. 26, no. 1, pp. 15-30, Jan 2007.

[32] Iris K Rüssel, Sandra R Tecelão, Joost PA Kuijer, Robert M Heethaar, J Tim Marcus, "Comparison of 2D and 3D calculation of left ventricular torsion as circumferential-longitudinal shear angle using cardiovascular magnetic resonance tagging," *J Cardiovasc Magn Reson,* vol. 11, no. 1, p. 8, 2009.

[33] sky, "The Coronary Arteries and Major Veins of the Heart (Anterior and Inferior Views)," [Online]. Available: http://biology-forums.com/index.php?action=gallery;sa=view;id=8048.

[34] wikipedia, "Pericardium," [Online]. Available: https://en.wikipedia.org/wiki/Pericardium.

[35] Slart RH, Bax JJ, de Jong RM, et al, "Comparison of gated PET with MRI for evaluation of left ventricular function in patients with coronary artery disease," *Nucl Med,* vol. 45, pp. 176-182, 2004.

[36] Edwards WD, Tajik AJ, Seward JB, "Standardized nomenclature and anatomic basis for regional tomographic analysis of the heart," in *Mayo Clin Proc*, 1981.

[37] ELSTER LLC, "Magnetism - Questions and Answersin MRI," 2015 . [Online]. Available: http://mriquestions.com/polar-plots.html.

[38] W. D. G. C. M. K. F. H. E. Daniel Kim, "Myocardial Tissue Tracking with Two-dimensional Cine Displacement-encoded MR Imaging: Development and Initial Evaluation," *Radiology,* vol. 230, no. 3, pp. 862-71, 2004.

[39] F. H. Sheehan , S. Ge , G. W. Vick , K. Urnes , W. S. Kerwin , E. L. Bolson , T. Chung , J. P. Kovalchin , D. J. Sahn , M. Jerosch-Herold and A. H. Stolpen, "Three-dimensional shape analysis of right ventricular remodeling in repaired tetralogy of Fallot," *Am. J. Cardiol.,* vol. 101, no. 1, pp. 107-113, 2008.

[40] Spottiswoode BS, Zhong X, Hess AT, Kramer CM, Meintjes EM, Mayosi BM, Epstein FH, "Tracking Myocardial Motion from Cine DENSE Images using Spatiotemporal Phase Unwrapping and Temporal Fitting," *EEE Transactions on Medical Imaging,* vol. 26, no. 1, p. 15–30, 2007.

Vita

Zhanqiu Liu received his Bachelor of Engineering degree in Aircraft Manufacturing Engineering at Harbin Institute of Technology, China. During his undergraduate research, he proposed a 3-DOF articulating portable working platform for astronauts during their extravehicular missions. Besides, he developed a prototype of multi-freedom morphing wing activated by shape memory alloys. He was also involved in the project of development of six-wheel lunar rover with foldable rocker-bogie suspension.

He is enrolled in Department of Mechanical Engineering in 2013 fall at University of Kentucky. At UK, he joined Professor Jonathan F. Wenk's Computational Biomechanics Laboratory on February 2015. At present, he majorly focuses on finite element modeling of biventricular mechanics and development of programs for computer-assisted optimization of therapies.