Boise State University
ScholarWorks

Electrical and Computer Engineering Graduate Theses and Dissertations

Department of Electrical and Computer Engineering

5-8-2012

Biomedical Matching Gui Development

Nazia Sarang
Boise State University

BIOMEDICAL MATCHING GUI DEVELOPMENT

by

Nazia Sarang

A project

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

Boise State University

December 2011

© 2011
Nazia Sarang
ALL RIGHTS RESERVED

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the project submitted by

Nazia Sarang

Thesis Title: Biomedical Matching GUI Development

Date of Final Oral Examination: December 2011

The following individuals read and discussed the thesis submitted by student Nazia Sarang, and they evaluated her presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Dr. Elisa Barney Smith	Chair
Dr. Murali Medidi	Co-Chair
Dr. Teresa Cole	Member, Supervisory Committee
Dr. Alark Joshi	Member, Supervisory Committee

The final reading approval of the thesis was granted by Dr. Elisa Barney Smith, Chair. The thesis was approved for the Graduate College by John R. Pelton, Ph.D., Dean of the Graduate College.

Dedicated to my parents, Sayeed and Hafsa

ACKNOWLEDGMENTS

I wish to express my gratitude to my advisor Dr. Elisa Barney Smith for considering me for this project and guiding me all the way towards the achievement of my M.S. Degree. I also thank Dr. Amit Jain for having recommend me to Dr. Elisa Barney Smith. Dr. Barney Smith helped me understand the concepts of Computed Tomography and Digitally Reconstructed Radiograph and introduced me to the world of medical imaging. I wish to thank Dr. Alark Joshi for his immense help in understanding the concepts of the Visualization Toolkit and the concepts involved behind the display of a structure in a three-dimensional view. I also wish to thank Dr. Teresa Cole for helping me understand the concept of graphics. I am also thankful to all my faculty members for helping me learn the key computer science skills required for this project and the rest of my career.

I am grateful to my parents and friends for their invaluable support and encouragement. I also thank all the members of Signal Processing Lab for their tremendous help for completing my project. I owe my success to all my teachers, right from kindergarten to graduate school.

ABSTRACT

Biomedical researchers constantly search for new methods to diagnose the extent of joint injuries in live human subjects. In order to achieve this, the researchers need to know the accurate three dimensional kinematic data of bones and joints and to accurately quantify how bones in a joint move relative to one another during dynamic activities. Algorithms have been developed previously to estimate the exact spatial position and movement of the bones. These methods involved generation of digitally reconstructed radiographs (DRR) from a three dimensional CT scan image of human joint with known position and orientation and comparing it with the original two dimensional video fluoroscopy (video X-ray) image. This way, a DRR image could be found which looks similar to the fluoroscopy frame of the knee joint and thus the position and orientation of the bone could be discovered.

Previous research has been done for finding the dimensions and the orientation of the bone by making use of a sequential Monte Carlo method [1] and Swarm Intelligence techniques in a parallel computing environment [2]. These methods are a little hard to understand by a person not involved in image processing. This project is about developing a user friendly GUI to help people understand the scope of the problem on which researchers are working. The GUI displays the bones in both three dimensional and two dimensional space, allowing the human user to do the search which was previosuly done automatically by the computer using image feature information. The user of the GUI will estimate the position of a given bone, given images of a bone in 3D and 2D space viewed at perspectives chosen by the user.

TABLE OF CONTENTS

ABSTRACT vi

LIST OF FIGURES xi

LIST OF ABBREVIATIONS xii

1 Introduction 1

1.1 Fluoroscopic Analysis of Knee Joint Kinematics 1

1.2 Graphical User Interface 4

1.3 Prior Research 5

1.4 Project Statement 6

2 Biomedical Matching Software Model 7

3 Methods 10

3.1 Computed Tomography 10

3.2 Digitally Reconstructed Radiograph 10

3.3 Ray Tracing 11

3.4 Marching Cubes Algorithm 12

4 Visualization Toolkit: Software used for 3D Computer Graphics . . 16

4.1 VTK Installation 19

5 Design and Implementation 21
5.1 Graphical User Interface 22
5.2 3D Image Display 27
5.3 Conversion of 3D images to 2D images 30

6 Conclusions and Future Work 33
6.1 Graphical User Interface for Fluoroscopic Analysis 33
6.2 Future Scope 34

REFERENCES 35

A README FILE 37

B Installing the program 40

C Running the program with different data 42

D Code Documentation 44
D.1 Classes Documentation 44
D.2 Slider Class Reference 44
D.2.1 Detailed Description 44
D.2.2 Constructor Documentation 45
D.2.3 Function Documentation 46
D.2.4 Variable Documentation 48
D.3 ScrollbarPanel Class Reference 48
D.3.1 Detailed Documentation 48
D.3.2 Constructor Documentation 48
D.3.3 Function Documentation 49

D.3.4 Variable Documentation 49
D.4 GUI Class Reference 49
D.4.1 Detailed Documentation 49
D.4.2 Constructor Documentation 53
D.4.3 Function Documentation 53
D.4.4 Action Listeners Documentation 55
D.4.5 Variable Documentation 57
D.5 DRR Class Reference 59
D.5.1 Detailed Documentation 59
D.5.2 Function Documentation 60
D.5.3 Variable Documentation 62
D.6 RayTracing Class Reference 62
D.6.1 Detailed Documentation 62
D.6.2 Function Documentation 63
D.6.3 Variable Documentation 64
D.7 InsertionSort Class Reference 64
D.7.1 Detailed Documentation 64
D.7.2 Function Documentation 65
D.8 vtkMarchingCubes Class Reference 65
D.8.1 Detailed Documentation 65
D.8.2 Constructor Documentation 67
D.8.3 Function Documentation 67
D.8.4 Variable Documentation 68
D.9 BiomedicalMatching Class Reference 68
D.9.1 Detailed Description 68

D.9.2 Function Description 69
D.9.3 Variable Documentation 69

LIST OF FIGURES

1.1 Model to form simulated X-ray [1] 2

2.1 Biomedical Matching Software Model 8

2.2 Outline of the Whole Project 9

3.1 Generation of a DRR Image [10] 11

3.2 Ray Tracing Algorithm [11] 13

3.3 Marching Cube [4] 14

3.4 Marching Cubes Algorithm [5] 15

4.1 Visualization Model [15] 17

4.2 Dataset types found in VTK: (a) Polygonal data, (b) Structured points, (c) Structured grid, (d) Unstructured grid, (e) Unstructured points, (f) Object Diagram [15]. 18

4.3 Instance of CMake 20

5.1 Instance of a Graphical User Interface 22

5.2 Three Dimensional Joint 23

5.3 Help Window as displayed in the GUI 26

5.4 Three Dimensional Data as seen in Volview 30

5.5 Three Dimensional Joint as seen in Volview 31

LIST OF ABBREVIATIONS

GUI – Graphical User Interface

DRR – Digitally Reconstructed Radiograph

API – Application Programming Interface

CT – Computed Tomography

VTK – Visualization Toolkit

MRI – Magnetic Resonance Imaging

CHAPTER 1

INTRODUCTION

1.1 Fluoroscopic Analysis of Knee Joint Kinematics

Biomedical researchers work to find new methods to diagnose the extent of joint injuries in live human subjects. In order to achieve this, the researchers need to know the accurate three dimensional kinematic data of bones and joints and to accurately quantify how bones in a joint move relative to one another during dynamic activities. There are various methods involved in creating two dimensional and three dimensional images of an object. A couple of them are explained as follows. Fluoroscopy is a medical imaging technique that produces a sequence of real-time X-ray images of an object moving in two dimensions. These continuous series of images produced in the fluoroscopy technique display motion when projected on a monitor. This motion can be viewed as any movie just like the conventional television. Computed tomography (CT) is an imaging technique that produces a static, three dimensional image of the scanned object. It is desired to know the position of the 3D bone joint at every time frame in the fluoroscopy "movie", which would be very beneficial to biomedical researchers in their quest to find new diagnostic methods for various joint injuries in human bodies. By matching the three dimensional image of a human joint with its equivalent two dimensional image, we can know at what 3D position the bone in the 2D image is located at each time instance. The two dimensional DRR image

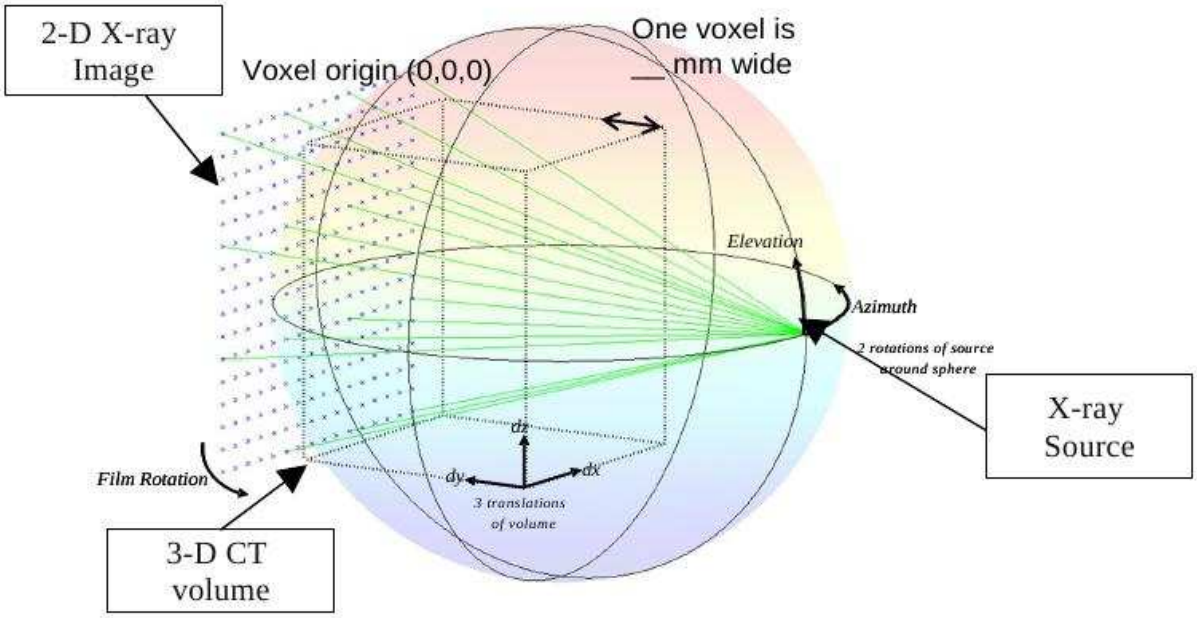

Figure 1.1: Model to form simulated X-ray [1]

can be generated from a three dimensional CT scan image using a model shown in Figure 1.1.

As shown in Figure 1.1, there is an X-ray source and an X-ray film positioned at equal radial distances on opposite sides of the origin. The CT cube (containing a sample of human hand or knee) is placed initially at the origin in between the X-ray source and the film. The CT cube can be moved in the x, y and z directions. The X-ray film can rotate while the X-ray source can move in the azimuth and elevation

dimensions. These form six degrees of freedom (x, y, and z coordinates of the cube, azimuth, elevation of the X-ray source and rotation of the film) that determine the resulting image. The azimuth can vary from $-180°$ to $180°$ and the elevation and rotation can each vary from $-90°$ to $90°$.

The CT cube seen in Figure 1.1 is converted into a 2D X-ray image and this conversion is done with the help of a method called Digitally Reconstructed Radiograph (DRR). The X-rays that are projected from the source to the film pass through the CT cube and produce a DRR image by determining the total bone density on that ray path though the CT cube. Different DRR images can be generated by changing the location of the X-ray source relative to the bone (CT image).

If a generated DRR image matches the 2D X-ray image, the six parameters representing the position of the cube, and the orientation of the X-ray source and the film can be known. Using this information and repeating the process for several fluoroscopy image frames, we can determine the position of the 3D bone over time. This helps in visualizing and analyzing the joint in motion.

The problem of finding the correct values of the 6 model parameters is similar to a search problem and using a brute force method to search for the target image will take a tremendous amount of time [1]. So, a better search solution is needed which can provide a significant speedup. Parallel computing methods have been used to estimate the six parameters given the CT cube and two dimensional data [2]. This work involved two swarm intelligence techniques in a parallel computing environment to create a test bed for fluoroscopic analysis and increase the speed of the process of finding the six parameters. The parallel programs were developed using two swarm intelligence techniques: Bees Algorithm and Particle Swarm Optimization technique. This method was successful in improving the speed of computation and achieving

considerable accuracy.

All these complexities are hard for a common user to understand by looking at the code. Hence this project aims at developing a GUI which will provide a user with an interface for viewing the three dimensional CT image and the DRR projection image of a bone in two dimensions. The user will complete the same process the automated search algorithm solves, but using the human 3D object knowledge. By contemplating how the user would describe a procedure to do that same task automatically, an appreciation for the scope of the problem will develop.

1.2 Graphical User Interface

In today's world a Graphical User Interface (GUI) is an essential part of any software application programming. It provides a user friendly interface for the underlying logical design of a stored program. All the software which we use on a daily basis includes some form of a GUI. The most common examples being the MP3 Players, Airline Self Ticketing and check-in and most software provided on Windows™ based computer platforms. The GUIs can be integrated with any sophisticated technology and all the complexities can be hidden from the user behind an attractive interface. GUIs can be used easily by any user rather than the application being confined only to those users who are aware of the programming and logic involved behind the application. The neat interface shown in the GUI with all the labeled buttons frees the user of reading any manuals before using the GUI. All these features make the GUIs more powerful and the performance of the application is considerably increased.

1.3 Prior Research

This project is an extension to the research done by Scott et al. [1] and Ramanatha [2]. The research conducted by Scott et al. was meant to find the three dimensional position of data over time by combining the three dimensional bone geometry data from static computed tomography (CT) images with time sequence information from two dimensional video fluoroscopy images [1]. The process involved creating virtual X-rays from the CT image through digitally reconstructed radiograph (DRR) projections. The need for manual initialization for matching the 3D and 2D data was eliminated using a Monte Carlo technique with a variable search range. This method was an optimization search as the search range decreased with the improvement in matching. This research was also implemented in a parallel environment using Swarm Intelligence algorithms, namely the Bees Algorithm and particle swarm optimization [2]. This research was successful in increasing the speed of the search process involved in fluoroscopic analysis.

However, a non-researcher finds it hard to understand the concepts behind the software and learn the functionality of the algorithms involved in matching the three dimensional bone data to two dimensional fluoroscopy image. Therefore, there is a need for a simple interface which describes the details behind the matching algorithms by making everything visual. The goal is to make the common person understand the mechanism of determining the three dimensional position data of the bones and joints by merely looking at the images of the bone in both two dimensions and three dimensions.

1.4 Project Statement

The purpose of this project is to create a graphical user interface which displays the bone in both three dimensions and two dimensions, and converting the three dimensional bone joint into a two dimensional image by specifying the alignment of the bone, azimuth and elevation of the X-ray source and rotation of the film array. The user can perform the search for the dimensions and the orientation of the bone by looking at the images. The Marching Cubes algorithm is used to display the bone in three dimensions with the help of the Visualization Toolkit.

This project will illustrate the use of a GUI to describe the image processing algorithms for matching three dimensional images of human joints with two dimensional fluoroscopic (video X-ray) image sequences. Further details of the biomedical image matching project are explained in Chapter 2. The GUI will display 2D and 3D images. Parameters are set with the help of the sliders provided in the GUI. The details of each part of the GUI will be explained in Chapter 3. The three dimensional image is displayed in the GUI with the help of the Visualization Toolkit and the Marching Cubes algorithm.

CHAPTER 2

BIOMEDICAL MATCHING SOFTWARE MODEL

In this chapter, the GUI model designed for the project is explained along with the process involved behind the GUI.

The part of the GUI responsible for converting the 3D image of the bone into its equivalent 2D image had to be converted from MATLAB to Java. Java was chosen for the main reason that it is fast and it can be run on any platform. The design of the GUI involved a lot of consideration to make it look informative and clear to any person who will be using it in the future. A number of sketches were prepared before the GUI was finalized to meet the needs of each component involved in the GUI, referring to the size of each component and the position of each component. Various components involved in the GUI are buttons, image displays, sliders and textboxes.

The design of the GUI is as shown in the Figure 2.1. As seen in Figure 2.1, the GUI shows two windows, where one window displays a three dimensional image of the joint obtained from the data and the other window shows two images shown in two dimensional form. One is a DRR image obtained from a set of parameters chosen by the software and the second is a DRR image obtained from a set of parameters chosen by the user.

The user through the GUI adjusts the model parameters to make the second 2D image display in the GUI. Then, a comparison is made to see if the two 2D images

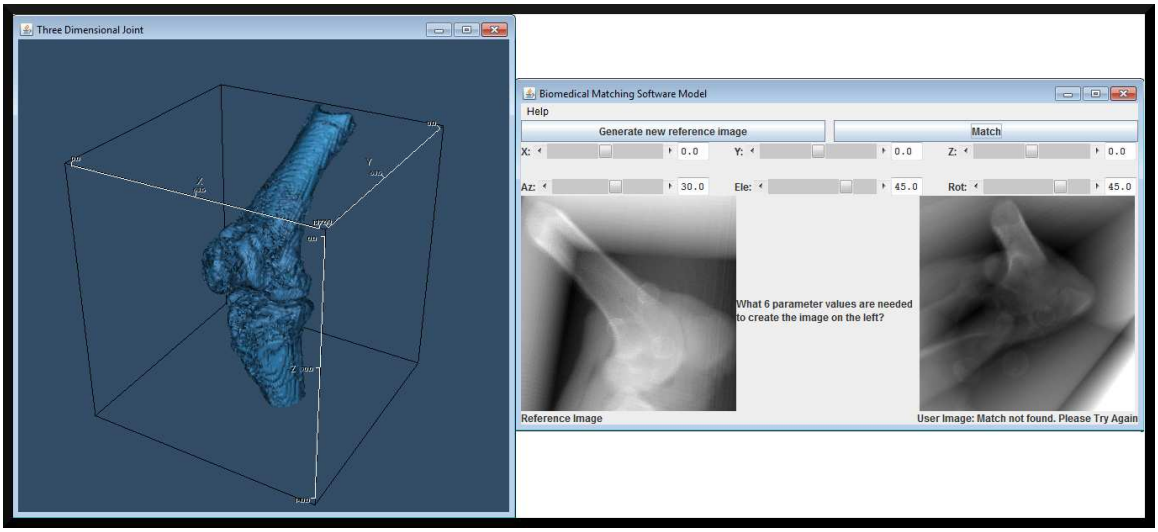

Figure 2.1: Biomedical Matching Software Model

match with each other and the user is shown the result of matching. If the two 2D images do not match, the user of the GUI can make some changes to the parameters and a different 2D image is displayed. The user can go on playing with the software until he gets a perfect matching 2D bone image. This whole process of guessing the values for the six degrees of freedom in an attempt to obtain a replica of the initial 2D image can be viewed as a game where any non-researcher can use it. This way, the user can replace the automatic search algorithms by selecting values for the parameters of his choice.

The working of the GUI can be explained in a diagram as shown in Figure 2.2. A DRR method is used to convert the three dimensional CT scan to a two dimensional image. The DRR method is run twice, first with the known initial parameters for the six degrees of freedom and then with the parameters chosen by the user for the six degrees of freedom. The six degrees of freedom being x, y, and z coordinates of the CT cube, azimuth, elevation of the X-ray source and rotation of the film. Once

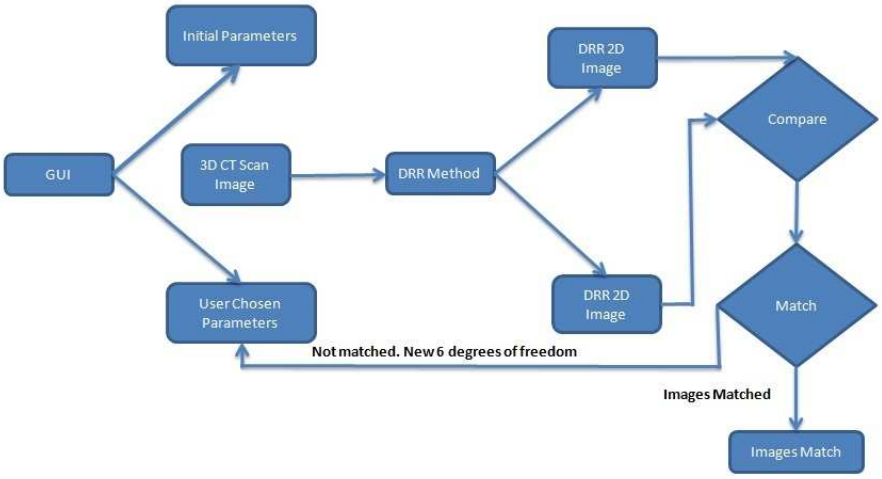

Figure 2.2: Outline of the Whole Project

the two DRR images are generated, a comparison is done on the two images. If they match, the iteration ends but if they do not match, the user should give new values for the six unknown parameters till the right parameters are guessed.

CHAPTER 3

METHODS

3.1 Computed Tomography

Computed Tomography is a medical imaging technique that is used to make pictures of structures inside the body showing every detail. The origin of the word "tomography" is from the Greek word "tomos" meaning "slice" or "section" and "graph" meaning "drawing" [6]. It combines a series of X-ray views taken from many different angles to produce cross-sectional images of the bones and soft tissues inside a human body. The resulting images can be viewed individually or can be formed into a 3D image by performing additional visualizations. These CT scan images provide much more information when compared to ordinary X-rays.

CT scans are used to visualize many organs inside the body with more precision. They are mainly used to examine people who have internal injuries from accidents. A CT scan can also visualize the brain and with the help of injected contrast material check for blockages or other problems in your blood vessels [7].

3.2 Digitally Reconstructed Radiograph

A Digitally Reconstructed Radiograph is a 2D radiograph which is generated from a CT scan. The resulting DRR image outputted from the method can be used as

a reference image for verifying the computer-designed treatments [8]. It basically verifies if the setup position of the body is correct before performing any radiation treatment [9]. The DRR method involves computing synthetic X-ray images, which are called digitally reconstructed radiographs. In this method, rays are cast through a CT image using a known camera geometry as shown in Figure 3.1 [10]. The two dimensional DRR image creation involves ray tracing which is discussed in the next section.

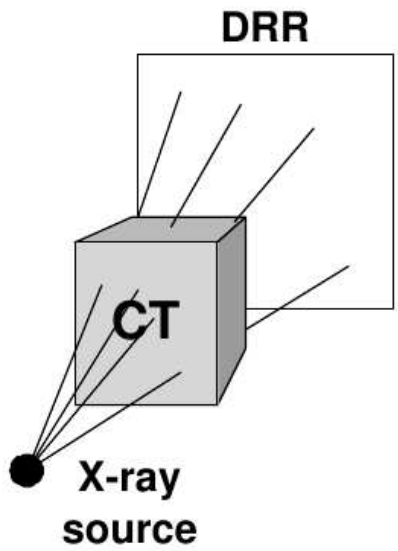

Figure 3.1: Generation of a DRR Image [10]

3.3 Ray Tracing

In computer graphics, ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane and simulating the effects of its encounters with virtual objects. This technique is capable of producing a very

high degree of visual realism, usually higher than that of typical scanline rendering methods, but at a greater computational cost [11].

The ray tracing algorithm described in Figure 3.2 explains the formation of an image when the ray of light passes through the pixels and bounces off the objects in the scene. This is used in computer graphics to render images of objects. Similarly, a DRR image is generated when X-rays are passed to the film, but for a DRR the "light" source is behind the object and the amount of "light" absorbed while penetrating the object is calculated for each ray.

Siddon's algorithm [3] describes the ray tracing method which constructs a DRR from a CT image by calculating the radiological path through a three - dimensional CT array. In this algorithm, the CT data is considered as consisting of intersection volumes of three orthogonal sets of equally spaced, parallel planes; defined as voxels. The intersection of the ray with the voxels are obtained as a subset of the intersections of the ray with the planes. For each voxel intersection length, the corresponding voxel indices are obtained and the products of the intersected length and particular voxel density are summed over all intersections to yield the radiological path.

3.4 Marching Cubes Algorithm

The Marching cubes algorithm was chosen to display the bone in three-dimensions. The Marching Cubes algorithm is a high resolution surface construction algorithm for viewing 3D data [4]. It creates triangle models of constant density surfaces (called isosurfaces) from 3D medical data. The two basic steps involved in the Marching Cubes algorithm are: (a) Locating the surface corresponding to a user-specified value and creating triangles. (b) Calculating normals to the surface at each vertex of each

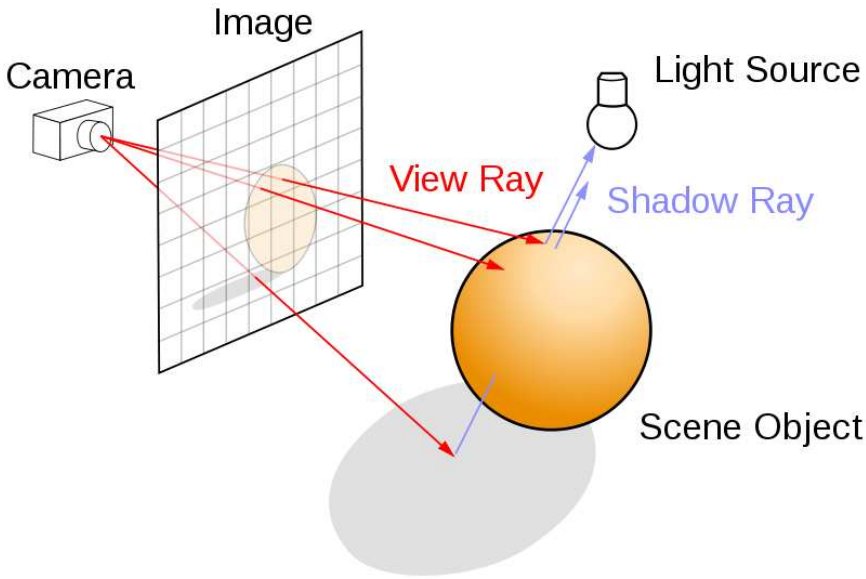

Figure 3.2: Ray Tracing Algorithm [11]

triangle to ensure a quality image of the surface.

Marching cubes uses a divide-and-conquer approach where the space is first divided into a series of small cubes. The algorithm then determines how the surface intersects the cube, then moves (or "marchs") to the next cube. The intersection of the surface with the cube can be seen in Figure 3.3. To find the surface intersection in a cube, the value one is assigned to a cube's vertex if the data value at that vertex exceeds (or equals) the value of the surface being constructed. These vertices are inside (or on) the surface. Cube vertices below the surface are assigned a value of zero and these vertices are outside the surface. This assumption is used to determine the topology of the surface within a cube. The sum total of all polygons generated in each cube forms the surface that approximates the one the data set describes.

Since there are eight vertices in each cube and two states, (inside and outside), there are only $2^8 = 256$ ways a surface can intersect the cube. Triangulating the 256

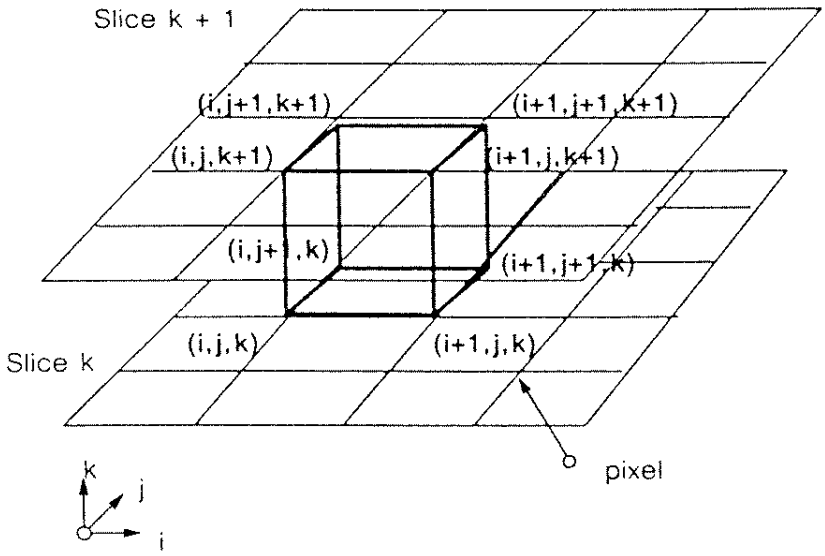

Figure 3.3: Marching Cube [4]

cases is possible but tedious and error-prone. So, the 256 cases are generalized to 15 patterns by rotations and symmetries. The 15 patterns can be seen in Figure 3.4

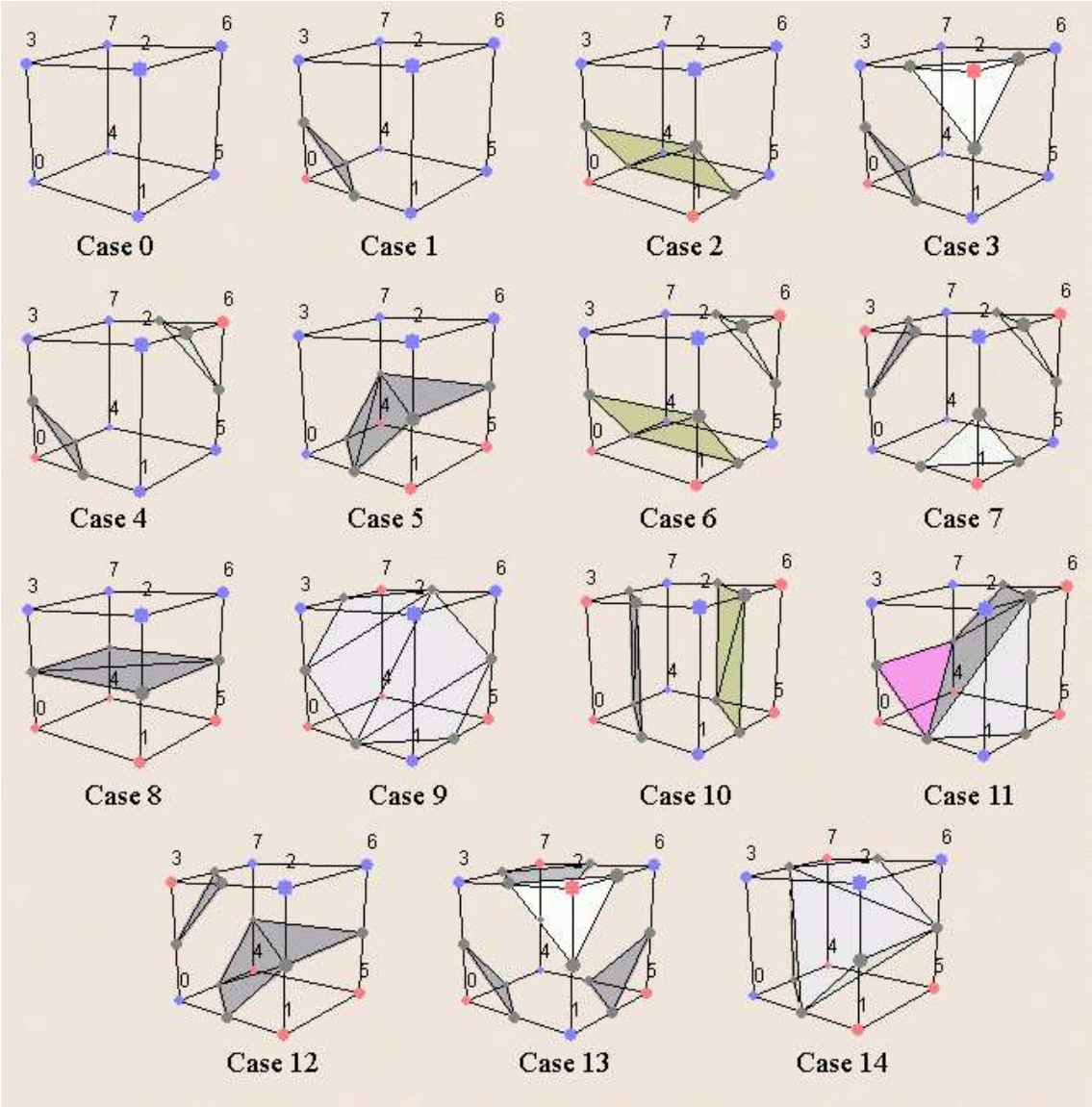

Figure 3.4: Marching Cubes Algorithm [5]

CHAPTER 4

VISUALIZATION TOOLKIT: SOFTWARE USED FOR 3D COMPUTER GRAPHICS

The Marching Cubes algorithm was programmed using the Visualization Toolkit. Initially, I made an attempt to display the 3D bone using Java programming language. A wrapper library in Java called as Java OpenGL is used to render 3D images in Java. I tried using Java OpenGL and started to program Marching Cubes algorithm. However, the limitations involved with this approach were that there were no classes and libraries which could be used to program the Marching Cubes algorithm. So, I made an another attempt in programming Marching Cubes algorithm using the Visualization Toolkit.

Visualization Toolkit is the software used in the project to display the three dimensional CT image. VTK is an open-source, freely available software system for 3D computer graphics, image processing and visualization. VTK consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java and Python. VTK supports a wide variety of visualization algorithms including: scalar, vector, tensor, texture, and volumetric methods. It also supports advanced modeling techniques such as implicit modeling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation [14]. VTK is a cross-platform software and runs on Linux, Windows, Mac and Unix platforms.

The Visualization Toolkit consists of two basic subsystems: a compiled C++ class library and an "interpreted" wrapper layer that lets you manipulate the compiled classes using the languages Java, Tcl, and Python. The advantage of this architecture is that you can build efficient (in both CPU and memory usage) algorithms in the compiled C++ language, and retain the rapid code development features of interpreted languages (which avoids the compile/link cycle and provides simple but powerful tools and access to GUI tools). The VTK model uses a data flow approach to transform information into graphical data. The data flow approach can be seen in Figure 4.1. In this approach modules are connected together into a network. The modules perform algorithmic operations on data as it flows through the network. The execution of this visualization network is controlled in response to demands for data (demand-driven) or in response to user input (event-driven). The advantage of this model is that it is flexible, and can be quickly adapted to different data types or new algorithmic implementations. As seen in Figure 4.1, the process objects A, B, C input and/or output one or more data objects. Data objects represent and provide access to data while process objects operate on the data. Objects A, B and C are source, filter, and mapper objects respectively.

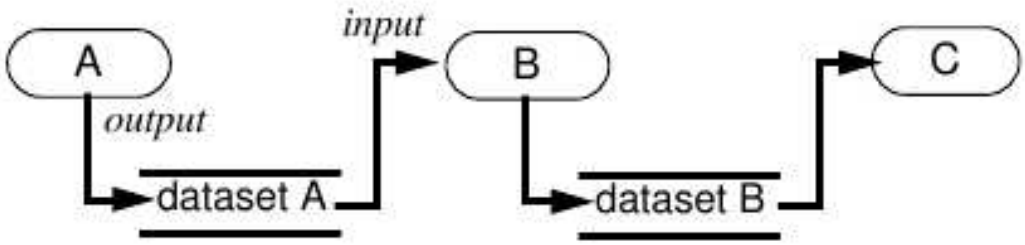

Figure 4.1: Visualization Model [15]

There are two basic types of objects involved in this approach: Process Objects and Data Objects. Data objects represent data of various types. Process objects are the modules or algorithmic portions of the visualization network. Data objects, also refererred to as datasets, represent and enable operations on the data that flows through the network. Figure 4.2 shows the dataset objects supported in VTK. Process objects can be classified into one of three types: sources, filters and mappers. Source objects initiate the network and generate one or more output datasets. Filters require one or more inputs and generate one or more output datasets. Mappers require one or more inputs and terminate the network.

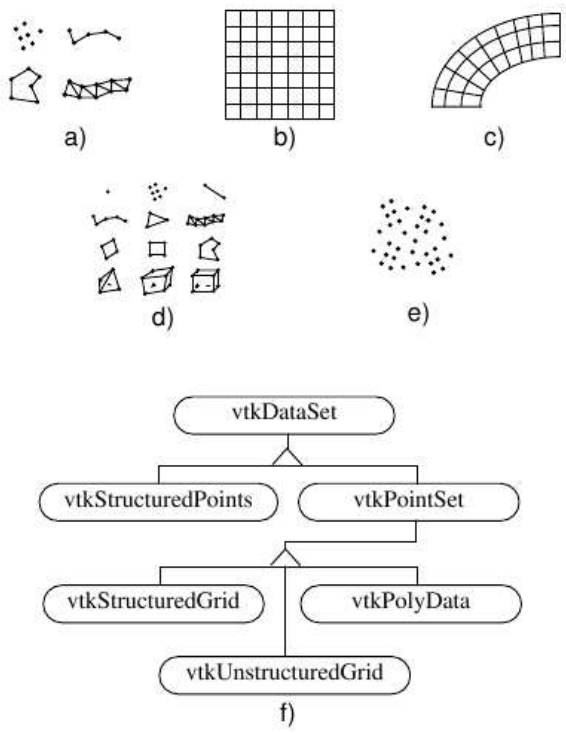

Figure 4.2: Dataset types found in VTK: (a) Polygonal data, (b) Structured points, (c) Structured grid, (d) Unstructured grid, (e) Unstructured points, (f) Object Diagram [15].

Datasets consist of geometric and topological structures (points and cells) as illustrated in Figure 4.2; they also have associated attribute data such as scalars or vectors. The attribute data can be associated with the points or cells of the dataset. Cells are topological organizations of points; cells form the atoms of the dataset and are used to interpolate information between points. The 3D image data to my program is of the form Structured Point Dataset. A structured point is a subclass of ImageData that is used to represent 2D images and 3D volumes. To start using VTK, it first needs to be installed in the way described in the next section.

4.1 VTK Installation

VTK installation in the Windows operating system involves installing CMake and the VTK source code. CMake is an open-source, cross-platform tool for configuring and managing the build process. CMake is downloaded from `http:/www.cmake.org`. Running CMake requires three basic pieces of information: which compiler to use, where the source code directory is and into which directory to place the object code, libraries, and binaries that the compilation process produces. CMake first detects the compiler present in the computer. Once the compiler is detected, VTK source code is downloaded and the path to it is fed into the CMake. An empty folder with the name VTKBin is created and the path to this folder is fed into the binaries path. This setup can be seen in Figure 4.3.

After the source code and binary directories have been selected, the `Configure` button is clicked and compiling starts. After the configuration is done successfully, the `Generate` button is clicked to generate all the libraries. Once VTK has been built, all libraries and executables produced will be located in the binary directory

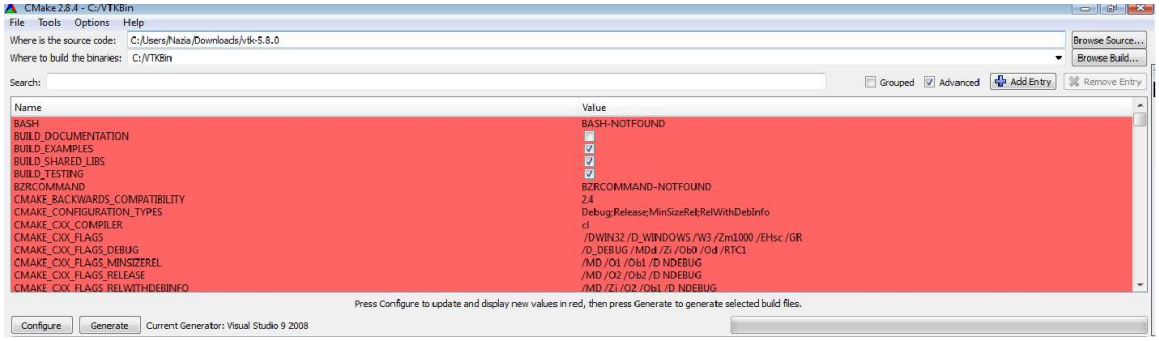

Figure 4.3: Instance of CMake

specified in CMake in a sub-folder called bin. After the installation, the VTK is ready to be used. Different programs can be written in Java with the vtk JAR file created by CMake. The Marching Cubes Algorithm discussed in Section 3.4 can be written in VTK as well.

CHAPTER 5

DESIGN AND IMPLEMENTATION

The Graphical User Interface (GUI) is designed in this project to illustrate the image processing algorithms involved in matching the three dimensional images of human joints with the two dimensional fluoroscopic images. The details of the GUI are explained in Chapter 2. Chapter 3 discusses all the algorithms that are used in the project. The part of the project involving 3D graphics is discussed in Chapter 4. In this chapter, the design and implementation details are provided. Appendix D discusses the code involved in the project. The whole project can be divided into the following components:

1. Graphical User Interface
    - Java Classes used for GUI
    - Event Handling
    - 2D Image Display
2. 3D Image Display
    - Visualization Toolkit
    - Marching Cubes Algorithm
3. Conversion of 3D image to 2D image

- Digitally Reconstructed Radiograph
- Ray Tracing Method

Each of these phases is described more elaborately in the next sections and everything is coded in the Java programming language.

5.1 Graphical User Interface

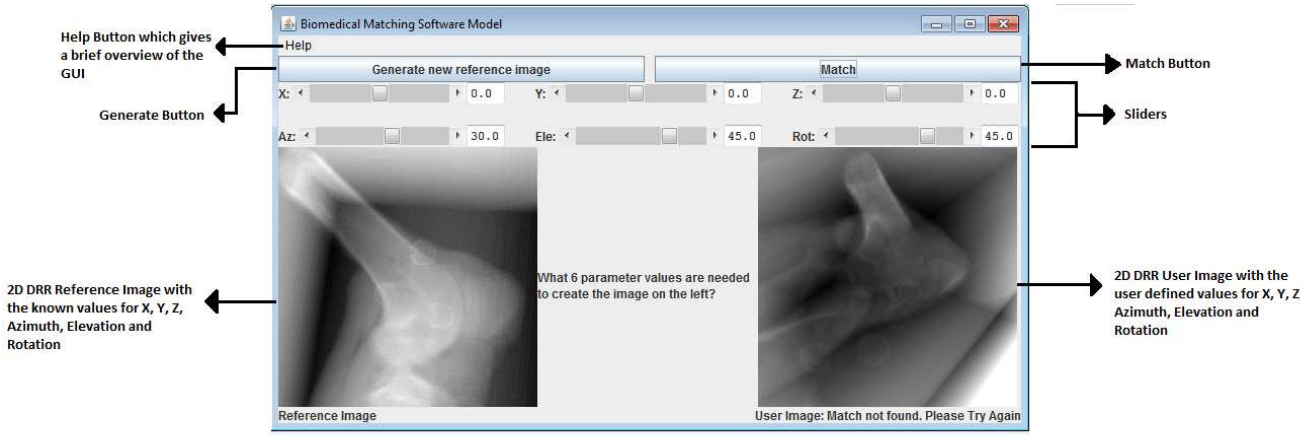

Figure 5.1: Instance of a Graphical User Interface

The GUI developed in the project is labelled and shown in Figure 5.1 and Figure 5.2. As labelled in Figure 5.1, there is a "Help" button. Along with it, there are two push buttons labelled as "Generate new Reference Image" and "Match". There are 6 sliders provided in the GUI which are used to vary the values of the six parameters discussed in Chapter 2. The six variables as defined before are as the position of an image in terms of x, y and z coordinates in the Cartesian coordinate system, and the orientation of the X-ray source and film defined in terms of azimuth, elevation and rotation. Apart from this, there are two image displays shown in the

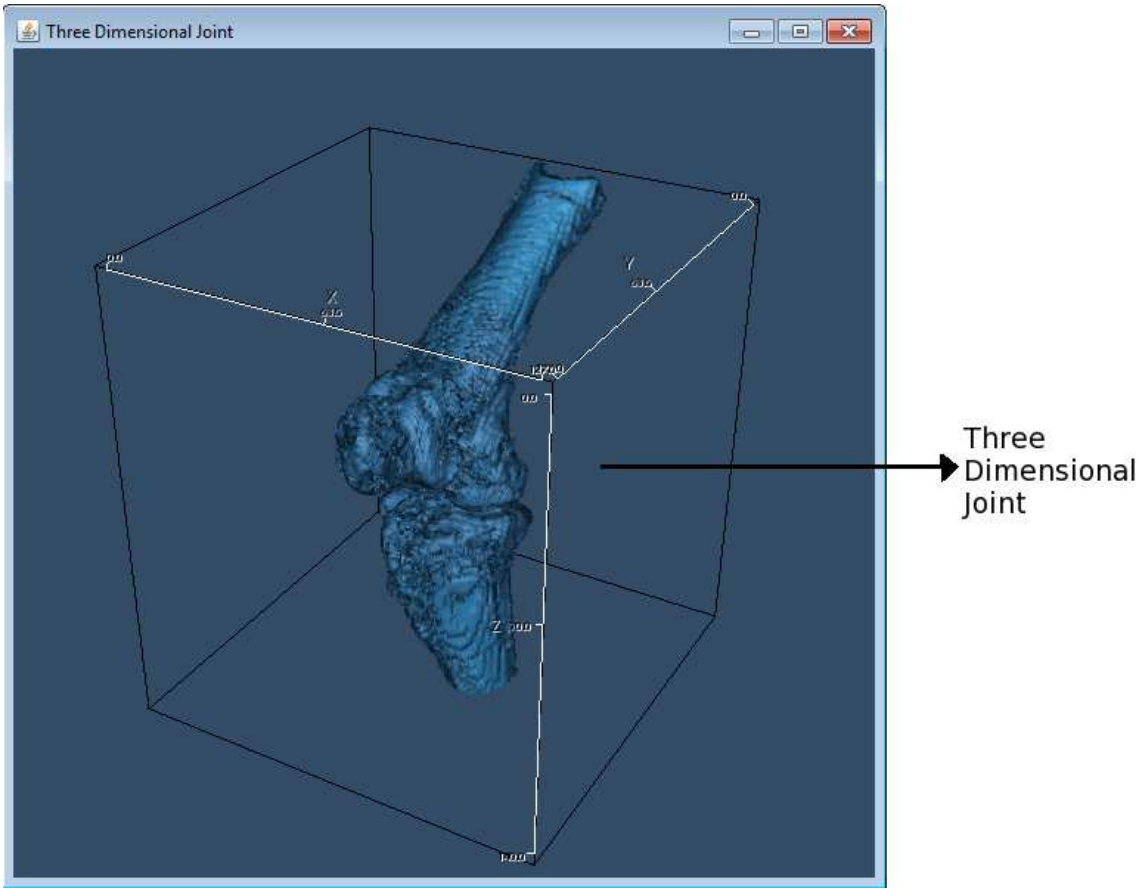

Figure 5.2: Three Dimensional Joint

GUI. One display shows a two dimensional DRR image with the known values for the six degrees of freedom and the other display shows a two dimensional DRR image with the user defined values for the six degrees of freedom. Among the two images, one image is labelled as the reference image. The reference image gets this name because it is created from the 3D image and a set of pre specified and fixed values of the six parameters. These six parameters are changed each time the "Generate new reference image" button is pressed. When the user feeds in his new values for the six parameters, a new 2D image is created. The image on the right is called the

user defined image. Figure 5.2 shows the bone in three dimensions. This image is generated with the help of Visualization Toolkit described in Chapter 4.

The functionality of the GUI and the methods involved behind every component of the GUI are discussed in detail as follows.

- **Java Classes used for the GUI**

  Several classes were used to create the GUI frame and the other parts of the GUI which include the buttons, sliders and image display fields. The various classes that have been used for the GUI design are JPanel, JButton, JFrame, JTextField and Container. All these classes are inbuilt in Java. JPanel is used to create the panels for various parts of the GUI. JFrame is used to create the frame for the whole GUI. JTextField is used to code the textfields in the GUI. The JButton class is used to design the three push buttons: "Generate new reference image" button, "Match" button and "Help" Button. The "Generate new reference image" button is responsible for creating a new reference image every time the button is clicked. The "Match" button is responsible for creating a 2D DRR image from the values selected by the user from the sliders for the six degrees of freedom. The "Help" button is responsible for displaying a window which gives a brief overview of the GUI. The events responsible behind the buttons are explained in the next section. Sliders are programmed with a scrollbar and a textfield to the right of the scrollbar which shows the value of the scrollbar. When the scrollbar is either moved left or right, the value in the textfield changes accordingly. Similarly, when the value of the textfield is changed, the position of the scrollbar is changed accordingly.

- **Event Handling**

Event handling refers to all the actions which can be done through the GUI. There are various events involved behind the push buttons "Generate new reference image" and "Match". When the end user clicks the "Generate new reference image" button, new values of the six parameters which are x, y, z, azimuth, elevation and rotation are read from a data file. This data file consists of a number of sets, where each set has a specified value for each of the six variables. This data file is created so as to avoid any blank images, resulting from some values for the six variables. The file is named as original_image_values.txt in my project. If the need arises, a new file can also be added with the same name or a different name and this name has to be reflected in the code. After the file is created, a random function is run to select any one set from the text file and then that particular set of values are sent to the DRR function and ray tracing function which results in a 2D DRR image. The resulting 2D DRR image is displayed in the reference image display using the ImageIcon class. This new image now acts as the reference image for the user to attempt to guess the values of x, y, z, azimuth, elevation and rotation in order to obtain a 2D image which matches with the new reference image. When the user is trying to match the DRR images, he gives in new values for the six degrees of freedom using the sliders. When the "Match" button is clicked, the program takes in the values of the sliders which were changed by the user. With these values for the six degrees of freedom, the DRR and ray tracing methods are called to generate a new 2D DRR user image. With every "Match" button pressed, the alignment of the image in 3D also changes according to the values inputted. For the sliders, whatever values are selected by either the text field or the slider, the same values get entered in the DRR code and the new 2D image

is created with these values. The new 2D image created by the ray tracing method is shown in the image display. After the code behind the DRR and ray tracing methods is executed, the user is told if the images matched or not. If the user selected values for the six degrees of freedom match with the set of values selected randomly for generating the DRR reference image, a message is displayed that the images matched. Otherwise, the user is shown a message that the images did not match and a hint for guessing the actual values (if the user guessed values are greater or lesser than the actual values). Then the user can try again with new values for the six variables until he gets a perfect match. When a user clicks on "Help" button, a new window is displayed which gives a brief information on the functionality of the GUI so that the user can start off using the GUI. The new window which gets opened can be seen in Figure 5.3.

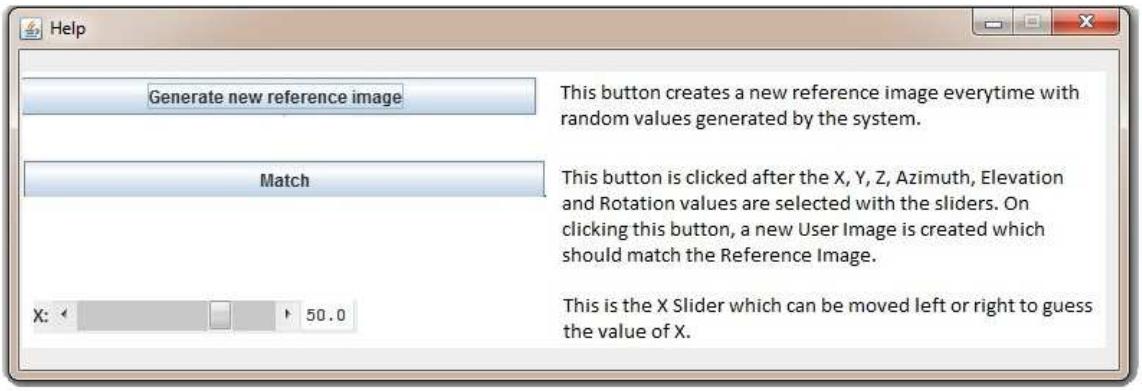

Figure 5.3: Help Window as displayed in the GUI

- **2D Image Display**

The 2D image is created with the help of the DRR and ray tracing methods. The DRR method takes in the values for the six parameters. The DRR method then calls the ray tracing method. In this method, the 3D data is reduced

to 2D data and all the values in the 2D array are scaled to the range of 0 - 255 so that the image can be displayed in gray scale. The 2D array formed is converted to a file of the image format with the help of the BufferedImage class. The WritableRaster class is then used to provide pixel writing capabilites. The resulting BufferedImage is then inputted to the ImageIcon class and the image is then displayed in the GUI.

5.2 3D Image Display

The 3D image shown in the GUI was developed using the Visualization Toolkit and Marching Cubes Algorithm as described in Chapter 3 and Chapter 4.

- **The Visualization Toolkit**

  VTK was first installed on a Windows™ machine. After installation, the vtk JAR file was added to the Java library path to enable VTK in Java. VTK requires the input data to be in the .vtk format. For this, the data file needs to be arranged in such a way that the rows are read first followed by columns and then the pages. The original input file for the three dimensional image which was in .txt format consisting of double values is then converted to unsigned character datatype and saved in .vtk format. After the file is converted to unsigned character, a vtk header is added to the beginning of the file. The vtk header attached to the new file is as follows:

  # vtk DataFile Version 1.0

  Datafile

  BINARY

  DATASET STRUCTURED_POINTS

DIMENSIONS 128 128 141

ASPECT_RATIO 1.0 1.0 1.0

ORIGIN 0.0 0.0 0.0

POINT_DATA 2310144

SCALARS scalars unsigned_char

LOOKUP_TABLE default

The header defined above specifies the details of the 3D image data.

1. The first line vtk DataFile Version 1.0 talks about the vtk version.

2. The second part is the header. The header consists of a character string terminated by end-of-line character \n. The header is 256 characters maximum. The header can be used to describe the data and include any other pertinent information.

3. The next part is the file format. The file format describes the type of file, either ASCII or binary. On this line the single word ASCII or BINARY appears. The data used is in BINARY format.

4. The fourth part is the dataset structure. The geometry part describes the geometry and topology of the dataset. This part begins with a line containing the keyword DATASET followed by a keyword describing the type of dataset. Then, depending upon the type of dataset, other keyword/ data combinations define the actual data. The 3D input image data to the program is of the form Structured Point Dataset. Structured point dataset is selected because it is a subclass of ImageData and is generally used to

represent 2D images and 3D volumes. The DIMENSIONS part in the header specifies the size of the 3D image data.

5. The final part describes the dataset attributes. This part begins with the keywords POINT_DATA or CELL_DATA, followed by an integer number specifying the number of points or cells, respectively. The SCALARS keyword defines the actual dataset attribute which is unsigned char.

The converted input file can be tested with the software Volview. Volview is an open-source, intuitive, interactive system for volume visualization that allows researchers to quickly explore and analyze complex 3D medical or scientific data on Windows, Mac and Linux computers. Users can easily load and interactively explore datasets using 2D and 3D display methods and tools. Volview is used in this project in order to make sure that the data looks right. Figure 5.4 shows the output when the converted 3D file is used as input. Figure 5.5 shows the first window when enlarged.

- **Marching Cubes Algorithm**

After the data testing is done and the 3D data is assured to be correct, the Marching Cubes algorithm is implemented on the image data. A panel is created to hold the 3D image of the bone joint and a vtkStructuredReader is implemented to read the datafile. Once the datafile is read, the opacity and color functions of the data are assigned to visualize the data with every detail. The camera angles are set up to view the 3D rendering of the bone. It provides methods to position and orient the view point and focal point. The 3D image is then bounded by a box using vtkCubeAxesActor2D class. The three axes of the bounding box are drawn to the image. The camera is made to control

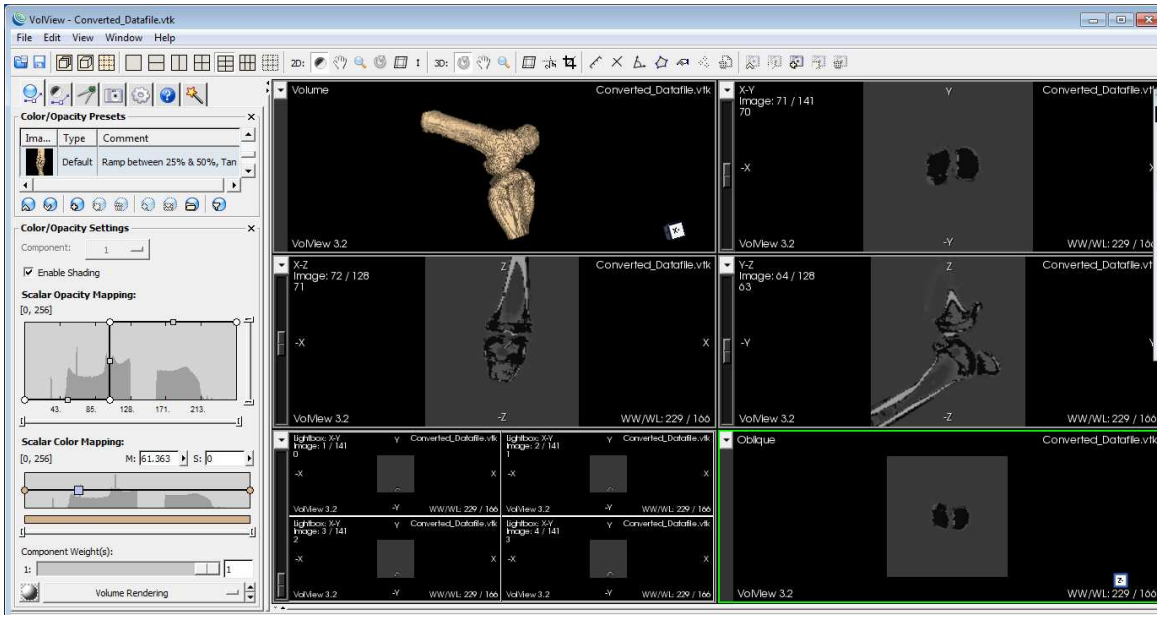

Figure 5.4: Three Dimensional Data as seen in Volview

the scaling and position of the image so that the image is always visible in the viewport. The output of the marching cubes shows us a bone structure in three dimensional view held within a bounding box showing the three axes.

5.3 Conversion of 3D images to 2D images

The conversion from three dimensional volume data to a two dimensional image representing a fluoroscopic image is done with the help of the Ray Tracing process. Before it goes through the Ray Tracing process, the DRR (Digitally Reconstructed Radiograph) method handles the 3D volume data and the six variables provided by the user to determine the two end points of every ray that will need to be traced. This method allows the camera to take a picture of the 3D image at any position by making use of the six position variables discussed in Chapter 3. The three camera angles are

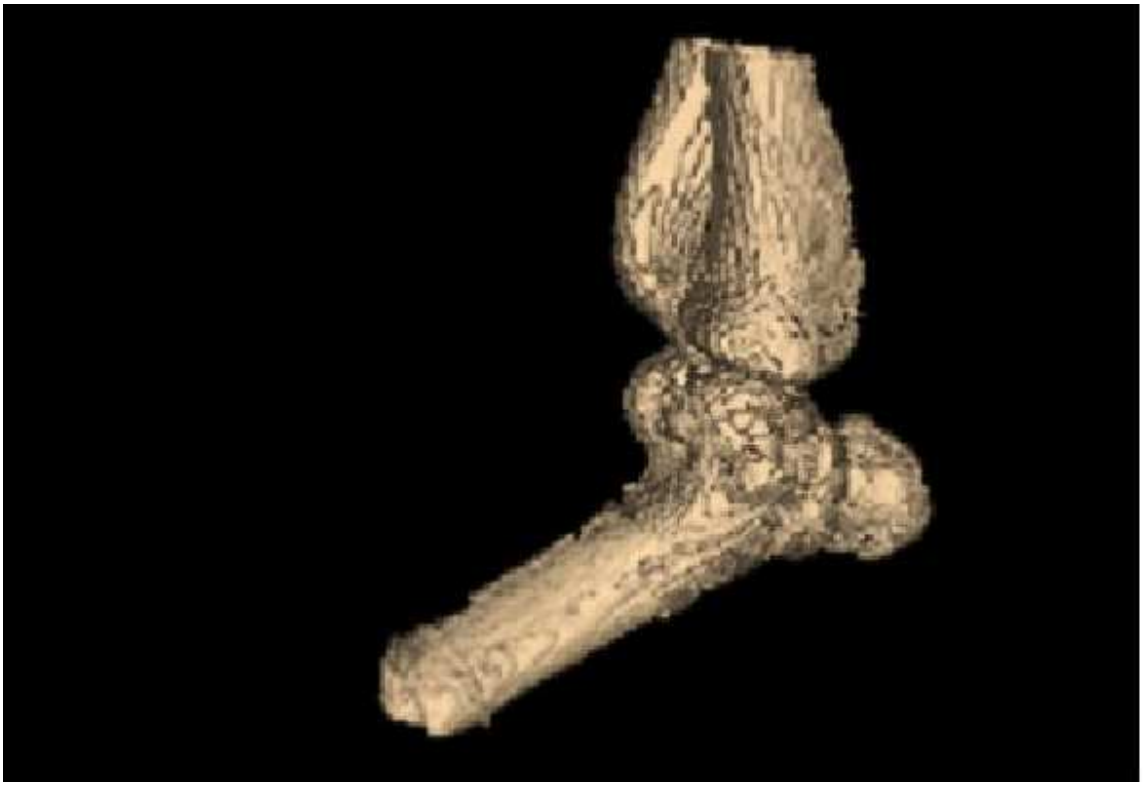

Figure 5.5: Three Dimensional Joint as seen in Volview

azimuth, elevation and rotation. Azimuth varies from $-180°$ to $180°$. Elevation and rotation vary from $-90°$ to $90°$. The other three degrees of freedom are x-offset, y-offset and z-offset. The x-offset, y-offset and z-offset are used to shift the 3D image in the x-direction, y-direction and z-direction respectively. All these values are used to configure the variables to the call to the ray-tracing method. The image in two dimensions represents the densities of the image at each position. The ray-tracing method then calculates these densities using Siddon's Algorithm [3]. The part of the project involving the conversion of a 3D image to a 2D image was available at the start of this project coded in MATLAB and C. All the methods were converted into Java and additional features of the GUI are introduced in this project to make the

interface friendly.

- **Digitally Reconstructed Radiograph**

  The DRR method was converted from MATLAB to Java. Part of the problem was to understand the MATLAB programming language and sketch out the possibilities of implementing the same in Java. MATLAB makes use of arrays to represent data of any number of dimensions and referencing the coordinates can be done by indexing the arrays. In Java, it had to be implemented differently using the Point3d class which is included in the javax.vecmath.Point3d package. There are a few other snippets of code which involve the conversion of degrees to radians and vice versa, converting spherical coordinates to polar co-ordinates. For this purpose, the Jsci.Math.CoordinateMath Package was included in the project.

- **Ray Tracing Method**

  The ray tracing function is used to calculate the densities based on the Siddons Algorithm [3]. It makes use of a CT image and constructs a DRR by calculating the path of the ray through a three dimensional CT image array. It involves the position of the camera, position of the cube through which the rays will be traced, the position of the points on the sheet the image is to be projected onto and the three dimensional CT array. The ray tracing method was converted from C to Java. In the previous project, there was a class which handled the computation between the MATLAB DRR code and the C ray tracing code. The C code involved many pointers so all the pointers had to be removed and an equivalent Java code is now written and the DRR code and ray tracing code are made to communicate with each other.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Graphical User Interface for Fluoroscopic Analysis

In this project, a model is designed which consisted of a CT cube placed between a camera and an X-ray film. The goal of the project is to find the position of the CT cube and azimuth and elevation angles of the camera and the rotation of the film array, given a three dimensional CT scan image and a two dimensional video fluoroscopy image.

A Graphical User Interface is developed in this project for finding the position and orientation of the bone. The GUI shows a three dimensional CT scan image and a two dimensional fluoroscopy image with known values for the position and orientation of the bone. The sliders are developed which helps the user in selecting any random values for the six variables (X, Y, Z, Azimuth, Elevation and Rotation). When the user selects some random variables, the orientation of the 3D image also changes which further helps the user in understanding the position and orientation of the bone.

The Graphical User Interface developed helps any non-researcher or a student to understand the research done before and how the automated search for the six degrees of freedom was done in the previous research. The software developed is also an improvement over the previous research in terms of the speed with which the

computation is done.

6.2 Future Scope

The project can be further developed and new features can be added. For instance, an option can be given to the user to look at only the femur part of the bone or only the tibia part of the bone or the complete bone. The project can also be extended to include Magnetic Resonance Imaging (MRI) images. Here the ray tracing method would be replaced with a 3D planar slice of the 3D MRI, and the fluoroscopic images would be 2D MR Images.

REFERENCES

[1] Charles Scott and Elisa H. Barney Smith. *An Unsupervised Fluoroscopic Analysis of Knee Joint Kinematics.* 19th IEEE International Symposium on Computer-Based Medical Systems (2006), Salt Lake City, Utah, June, pp. 377 - 377.

[2] Renu Ramanatha, *A Parallel Computing Test Bed for Performing an Unsupervised Fluoroscopic Analysis of Knee Joint Kinematics.* Master Thesis, Boise State University, October 2009.

[3] Siddon, R. L. *Fast calculation of the exact radiological path for a three dimensional CT array.* Medical Physics 12(2), Mar/Apr 1985, pp. 252-255.

[4] William E. Lorensen and Harvey E. Cline. *Marching Cubes: A High Resolution 3D Surface Construction Algorithm.* SIGGRAPH '87, Anaheim, July 27 - 31, 1987. pp. 163 - 169.

[5] Wikipedia - Marching Cubes Algorithm. *World Wide Web.* `http://en.wikipedia.org/wiki/Marching\_cubes`

[6] Computed Tomography. *World Wide Web.* `http://www.fda.gov/Radiation-EmittingProducts/RadiationEmittingProductsandProcedures/MedicalImaging/MedicalX-Rays/ucm115318.htm`

[7] *CT Scan - Mayoclinic.com* `http://www.mayoclinic.com/health/ct-scan/MY00309`

[8] George W. Sherouse, Kevin Novins and Edward L. Chaney. *Computation of digitally reconstructed radiographs for use in radiotherapy treatment design.* International Journal of Radiation Oncology Biology Physics, Volume 18, pp. 651 - 658.

[9] *Digitally Reconstructed Radiograph* `http://plastimatch.org/drr.html`

[10] Daniel B. Russakoa, Torsten Rohlnga, Daniel Rueckertb, Ramin Shahidia, Daniel Kima and Calvin R. Maurer *Fast calculation of digitally reconstructed radiograph using light fields.* IEEE Transactions, Medical Imaging 2003: Image Processing, Volume 5032, pp. 684 - 695, 2003.

[11] Wikipedia - Ray Tracing Algorithm. *World Wide Web.* http://en.wikipedia.org/wiki/Ray_tracing_\%28graphics\%29

[12] Kitware, Inc. *The VTK User's Guide Updated for Version 5.* Kitware, Columbia, 2006.

[13] Will Schoeder, Ken Martin, Bill Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics.* Kitware, United States of America, 2002.

[14] Visualization Toolkit. *World Wide Web.* http://www.vtk.org

[15] William J. Schroeder, Kenneth M. Martin, William E. Lorensen. *The Design and Implementation Of An Object-Oriented Toolkit For 3D Graphics And Visualization.* Proceedings of the 7th conference on Visualization, 1996, IEEE Computer Society Press Los Alamitos, CA, USA, 1996, pp. 93 - 100.

[16] Volview. *World Wide Web.* http://www.kitware.com/products/volview.html

APPENDIX A

README FILE

This project creates a GUI to perform fluoroscopic analysis of knee - joint kinematics. This appendix chapter discusses the steps involved in running the project in a system.

1. Description
2. How to Compile and Run the Project

1. Description

The following are the steps involved:

1. Creating a GUI
2. Reading a CT scan image
3. Generating DRR image with known values of x, y, z, azimuth, elevation and rotation
4. Generating another DRR image with user defined values of x, y, z, azimuth, elevation and rotation
5. Matching the two DRR images and displaying the result of matching

2. How to Compile and Run the Project

The main program is structured in such a way that there is a folder, which contains

the project in .jar format and the data files required by the project. The jar file is created with the required data files and the image filess needed by the program. The input data for the jar file consists of CT image file, a text file having a list of values for the six degrees of freedom (x, y, z, azimuth, elevation and rotation) and CT image file in .vtk format. The "Release" folder contains all the dll files which are required to run VTK on any Windows™ system. The "Release" folder gets created after VTK is installed on a computer. The VTK installation creates all the dll files and stores them in the "Release" folder. The "Release" folder can then be used on any system to run VTK programs even if VTK is not installed on that new system. The project can be run on any computer by following the steps outlined below. The operations involved include

1. Copy the project folder onto the system in which the project needs to be run. The files can be placed anywhere on the system. For instance, they can be placed on the Desktop.

2. The code needs some dll files to run on the system and these dll files are located in the "Release" folder. In order that the system finds the required dll files, we need to set the environment variable to locate the "Release" folder.

3. Setting environment variables:

   (a) Click Start on your computer, right click on My Computer. Then click on Properties.

   (b) Click on Advanced tab and then click on Environment Variables.

   (c) Go to the System variables and scroll down until you find "Path" variable. Double click on Path and you will get a new dialog box.

(d) In the Variable value, go to the end of the line and type a semicolon (;). After typing the semicolon(;), enter the address of the "Release" folder. For example, if the "Release" folder is located on the Desktop, enter the full address of the folder which might be C:\Documents and Settings\username\Desktop\Release

(e) After entering the full address to the Release folder, click on OK to all the windows opened.

(f) After setting the environment variable, open the BiomedicalProject folder and double click on Biomedical_gui. Wait for a minute or so and the project should run.

APPENDIX B

INSTALLING THE PROGRAM

The following section describes the steps involved if you need to edit the code in the project. The project is run in Eclipse software and it also requires VTK Installation. The following are the steps involved in installing the program freshly on a different computer.

1. Download the Eclipse classic software on the system.
2. Create a new Java Project in eclipse.
3. Import the project into the workspace by hitting on Import button and loading the program in the current project.
4. Create two sub directories in the src directory with the names “data” and “images”.
5. Load the data files (CT image data, text file consisting of list of values for the six degrees of freedom and CT image data in vtk format) in the data folder in the src directory.
6. Load the image files (blank image for showing it on the user image display initially and a “Images do not match” image shown on the user image display when the user image does not match with the reference image) in the images folder in the src directory.

7. In Eclipse, click on project properties and in the java build path, add external jars (vtk.jar and vecmath.jar).

8. In Eclipse, click on project properties an click Run and Debug Settings. Edit the launch configuration of the class already present in the Lauch configuration. Set the environment variable to the location of the dll files in the system.

9. Click on Run to run the program.

APPENDIX C

RUNNING THE PROGRAM WITH DIFFERENT DATA

The project is not confined to one particular data. Any CT image data can be read as input to the project and the three dimensional image can be shown. However, the data needs to be modified in order to be compatible with the VTK program. The following steps illustrate the steps involved in modifying any data:

1. Check the arrangement of the array elements in the given CT image array. Make sure that the array is arranged in such a way that the rows are read first followed by columns and then the pages.

2. If the CT image array arrangement is not the same as mentioned above, the alignment needs to be changed in MATLAB.

3. Convert the data type of CT image array elements to unsigned character data type. This can be done with the "dataCon" method mentioned in Appendix D. The "dataCon" method is written in Java.

4. A vtk header is added to the converted CT image array. The vtk header to be appended to the beginning of the file is:
   # vtk DataFile Version 1.0
   Datafile
   BINARY

DATASET STRUCTURED_POINTS

DIMENSIONS 128 128 141

ASPECT_RATIO 1.0 1.0 1.0

ORIGIN 0.0 0.0 0.0

POINT_DATA 2310144

SCALARS scalars unsigned_char

LOOKUP_TABLE default

More details on the VTK header are described in Chapter 4.

5. Save the file as "name of the file".vtk.

After the data is converted to vtk format, it can be read as input to the vtkMarchingCubes class and the program can be re-run to show the new image.

APPENDIX D

CODE DOCUMENTATION

This appendix chapter consists of the source code documentation for the essential components of the code.

D.1 Classes Documentation

D.2 Slider Class Reference

D.2.1 Detailed Description

This class is used to design the sliders which contain a horizontal scrollbar and a textfield to the right of the scrollbar. The textfield shows the value of the slider. The scrollbar can be moved to the left and right to increase or decrease the value and the resulting value is displayed in the textfield. The textfield is set to editable so as to change the value of the slider by editing the value in the textfield. On changing the value of the textfield, the scrollbar also changes accordingly.

Class Variables

- private Scrollbar scrollbar
- private TextField textfield
- private ScrollbarPanel scrollbarPanel

Constructors

- public Slider (double min, double max, double initialValue)

Functions

- public boolean handleScrollbarEvent (Event event)
- public boolean handleTextfieldEvent (Event event, Object object)
- public double getValue()
- public void setValue (double oldValue)
- public String getText()
- public void setText (String text)
- private void setTextFieldValue()

D.2.2 Constructor Documentation

D.2.2.1 public Slider (double min, double max, double initialValue)

Constructs a slider with the specified minimum, maximum and initial values.

Parameters:

min Minimum value of the slider to be set

max Maximum value of the slider to be set

initialValue Initial value of the slider to be set

D.2.3 Function Documentation

D.2.3.1 public boolean handleScrollbarEvent (Event event)

Detects the event in the program when the scrollbar changes.

Parameters:

event Event defining the change in scrollbar

Returns:

Returns true when there is a change in the scrollbar else returns false

D.2.3.2 public boolean handleTextfieldEvent (Event event, Object object)

Detects the event in the program when the textfield changes.

Parameters:

event Change in the textfield

object Indicating the textfield

Returns:

Returns true if there is a change in the textfield else returns false

D.2.3.3 public double getValue()

Returns the current value of the scrollbar.

Returns:

Value of the scrollbar

D.2.3.4 public void setValue (double oldValue)

Assigns value to the scrollbar.

Parameters:

oldValue The value which is set to the scrollbar

Returns:

Null

D.2.3.5 public String getText()

Gets the current value of the textfield.

Returns:

Current value of the textfield

D.2.3.6 public void setText (String text)

Sets a new value to the textfield.

Parameters:

text The new value of the textfield

Returns:

Null

D.2.3.7 private void setTextFieldValue()

Gets value from the scrollbar and sets the textfield to that value.

Returns:

Null

D.2.4 Variable Documentation

D.2.4.1 private Scrollbar scrollbar

D.2.4.2 private TextField textfield

D.2.4.3 private ScrollbarPanel scrollbarPanel

D.3 ScrollbarPanel Class Reference

D.3.1 Detailed Documentation

This class is used to design a panel which holds the scrollbar with adjustable margins.

Class Variables

- private Insets insets

Constructors

- public ScrollbarPanel (int margins)

Functions

- public int getMargins()
- public void setMargins (int margins)

D.3.2 Constructor Documentation

D.3.2.1 public ScrollbarPanel (int margins)

Constructs a scrollbar with the given margins.

Parameters:

margins The margins with which the scrollbar needs to be designed

D.3.3 Function Documentation

D.3.3.1 public int getMargins()

Gets the margins of the scrollbar.

Returns:

The top insets value

D.3.3.2 public void setMargins (int margins)

Sets the margins of the scrollbar to the new value.

Parameters:

margins The value of the margins to be set

Returns:

Null

D.3.4 Variable Documentation

D.3.4.1 private Insets insets

D.4 GUI Class Reference

D.4.1 Detailed Documentation

This class is responsible for developing the GUI. Every part of the GUI other than the display of 3D image is done in this class.

Class Variables

- JLabel OrigImgLabel

- JLabel OrigImgNameLabel
- JLabel UserImgLabel
- JLabel UserImgNameLabel
- ImageIcon UserImgIcon
- ImageIcon OrigImgIcon
- JLabel resultLabel
- double [][][] CT_image
- Container pane
- TextField xText
- TextField yText
- TextField zText
- TextField azimuthText
- TextField elevationText
- TextField rotationText
- JPanel xPanel
- JPanel yPanel
- JPanel zPanel
- JPanel azimuthPanel

- JPanel elevationPanel
- JPanel rotationPanel
- JPanel ImgPanel
- JPanel LeftImgPanel
- JPanel RightImgPanel
- double xValue = 0.0
- double yValue = 0.0
- double zValue = 0.0
- double azimuthValue = 0.0
- double elevationValue = 0.0
- double rotationValue = 0.0
- double x_generate
- double y_generate
- double z_generate
- double azimuth_generate
- double elevation_generate
- double rotation_generate
- double x_match

- double y_match
- double z_match
- double azimuth_match
- double elevation_match
- double rotation_match
- double tempX
- double tempY
- double tempZ
- double tempAzimuth
- double tempElevation
- double tempRotation
- double x_lower = -60
- double y_lower = -10
- double z_lower = -60
- double azimuth_lower = -180
- double elevation_lower = -90
- double rotation_lower = -90

Constructors

- public GUI()

Functions

- protected Component getImagePanel()
- private Component getRightImagePanel()
- private Component getLeftImagePanel()
- private JComponent getNorthPanel()
- protected JComponent getCenterPanel()

Action Listeners

- public class generateButtonListener implements ActionListener
- private class matchButtonListener implements ActionListener

D.4.2 Constructor Documentation

D.4.2.1 public GUI()

Constructs a GUI consisting of a "Help" button, "Generate new reference image" button, "Match" button, six sliders and two image displays for showing two dimensional DRR images.

D.4.3 Function Documentation

D.4.3.1 protected Component getImagePanel()

Designs a panel to hold two image panels, the left image panel and the right image panel.

Returns:

Panel consisting of two other panels

D.4.3.2 private Component getRightImagePanel()

Designs a panel to hold the right image display. The right image display holds the user image.

Returns:

Panel consisting of right image display which shows the user image

D.4.3.3 private Component getLeftImagePanel()

Designs a panel to hold the left image display. The left image display holds the reference image.

Returns:

Panel consisting of left image display which shows the reference image

D.4.3.4 protected Component getNorthPanel()

Designs a panel consisting of the buttons, "Generate new reference image" and "Match".

Returns:

Panel consisting of "Generate new reference image" button and "Match" button

D.4.3.5 protected Component getCenterPanel()

Designs a panel consisting of the six sliders, xSlider, ySlider, zSlider, azimuthSlider, elevationSlider, rotationSlider. Each slider consists of a label which gives a unique name to the slider, scrollbar to move left and right to change the value and a textfield

which shows the value of the scrollbar. The textfield is also editable to enable changing the value of the scrollbar.

Returns:

Panel consisting of six sliders

D.4.4 Action Listeners Documentation

D.4.4.1 public class generateButtonListener implements ActionListener

Action listener to handle the events when the "Generate new reference image" button is clicked.

Functions

- public void actionPerformed (ActionEvent arg0)

Function Documentation

Performs the actions when "Generate new reference image" button is clicked. When the button is clicked, the value for the six degrees of freedom are read from a data file. The values are inputted to the DRR function to generate a 2D DRR image representing the reference image. The reference image is displayed on the left image display panel.

Parameters:

ActionEvent arg0 Action when the "Generate new reference image" button is clicked

Returns:

Null

D.4.4.2 public class matchButtonListener implements ActionListener

Action listener to handle the events when the "Match" button is clicked.

Functions

- public void actionPerformed (ActionEvent arg0)

Function Documentation

Performs the actions when "Match" button is clicked. When the button is clicked, the value for the six degrees of freedom are read from the sliders. The values are inputted to the DRR function to generate a 2D DRR image representing the user image. The user image is displayed on the right image display panel.

Parameters:

ActionEvent arg0 Action when the "Match" button is clicked

Returns:

Null

D.4.5 Variable Documentation

D.4.5.1 JLabel OrigImgLabel

D.4.5.2 JLabel OrigImgNameLabel

D.4.5.3 JLabel UserImgLabel

D.4.5.4 JLabel UserImgNameLabel

D.4.5.5 ImageIcon UserImgIcon

D.4.5.6 ImageIcon OrigImgIcon

D.4.5.7 JLabel resultLabel

D.4.5.8 double [][][] CT_image

D.4.5.9 Container pane

D.4.5.10 TextField xText

D.4.5.11 TextField yText

D.4.5.12 TextField zText

D.4.5.13 TextField azimuthText

D.4.5.14 TextField elevationText

D.4.5.15 TextField rotationText

D.4.5.16 JPanel xPanel

D.4.5.17 JPanel yPanel

D.4.5.18 JPanel zPanel

D.4.5.19 JPanel azimuthPanel

D.4.5.20 JPanel elevationPanel

D.4.5.21 JPanel rotationPanel

D.4.5.22 JPanel ImgPanel

D.4.5.23 JPanel LeftImgPanel

D.4.5.24 JPanel RightImgPanel

D.4.5.25 double xValue = 0.0

D.4.5.26 double yValue = 0.0

D.4.5.27 double zValue = 0.0

D.4.5.28 double azimuthValue = 0.0

D.4.5.29 double elevationValue = 0.0

D.4.5.30 double rotationValue = 0.0

D.4.5.31 double x_generate

D.4.5.32 double y_generate

D.4.5.33 double z_generate

D.4.5.34 double azimuth_generate

D.4.5.35 double elevation_generate

D.4.5.36 double rotation_generate

D.4.5.37 double x_match

D.4.5.38 double y_match

D.4.5.39 double z_match

D.4.5.40 double azimuth_match

D.4.5.41 double elevation_match

D.4.5.42 double rotation_match

D.4.5.43 double tempX

D.4.5.44 double tempY

D.4.5.45 double tempZ

D.4.5.46 double tempAzimuth

D.4.5.47 double tempElevation

D.4.5.48 double tempRotation

D.4.5.49 double x_lower = -60

D.4.5.50 double y_lower = -10

D.4.5.51 double z_lower = -60

D.4.5.52 double azimuth_lower = -180

D.4.5.53 double elevation_lower = -90

D.4.5.54 double rotation_lower = -90

D.5 DRR Class Reference

D.5.1 Detailed Documentation

This class contains DRR function to creat a DRR model which includes setting up the point on a film aray and uses ray tracing to creat a DRR image for the given orientation and offset values. The DRR function helps in creating Digitally Reconstructed Radiograph images

Class Variables

- double x_adj
- double y_adj

- double z_adj
- double azimuth
- double elevation
- double rotation
- static int film_sizePIX = 256
- static double [][][] film_array
- static int step = 1
- static double vol_width = 22.0160
- static double vol_length = 22.0160
- static double vol_height = 28.1000
- double[][] DRR_img = new double[256][256]
- static double[] p_camera = {0, 0, 0}

Functions

- public static void DRR_function (double x_adj, double y_adj, double z_adj, double azimuth, double elevation, double rotation) throws IOException

D.5.2 Function Documentation

public static void DRR_function (double x_adj, double y_adj, double z_adj, double azimuth, double elevation, double rotation) throws IOException

Calculates the end points of the ray passing through the CT cube.

Parameters:

x_adj The X co-ordinate of the position of the CT cube

y_adj The Y co-ordinate of the position of the CT cube

z_adj The Z co-ordinate of the position of the CT cube

azimuth The azimuth angle of the camera

elevation The elevation angle of the camera

rotation The rotation angle of the film array

Returns:

Null

D.5.3 Variable Documentation

D.5.3.1 double x_adj

D.5.3.2 double y_adj

D.5.3.3 double z_adj

D.5.3.4 double azimuth

D.5.3.5 double elevation

D.5.3.6 double rotation

D.5.3.7 static int film_sizePIX = 256

D.5.3.8 static double [][][] film_array

D.5.3.9 static int step = 1

D.5.3.10 static double vol_width = 22.0160

D.5.3.11 static double vol_length = 22.0160

D.5.3.12 static double vol_height = 28.1000

D.5.3.13 double[][] DRR_img = new double[256][256]

D.5.3.14 static double[] p_camera = {0, 0, 0}

D.6 RayTracing Class Reference

D.6.1 Detailed Documentation

Implements Siddon's ray tracing algorithm. Ray tracing involves constructing a two dimensional DRR image from a CT image by calculating the radiological path

through a three dimensional CT image data.

Class Variables

- public static int NR_END = 1
- public static double[] CT_orig = {0, 0, 0}
- public static double[][] DRR_out_im = new double [256[256]
- public static double[] d_vox = new double[3]
- static BufferedImage bi
- static double[] ct_farcnr = {0, 0, 0}
- static int count1 = 0
- static int blank

Functions

- public static void ray_function (double [][][] CT_image, double [] cam_pt, double [][][] film_array) throws IOException

D.6.2 Function Documentation

D.6.2.1 public static void ray_function (double [][][] CT_image, double [] cam_pt, double [][][] film_array) throws IOException

Constructs a two dimensional DRR image from the given CT image using Siddon's ray tracing algorithm.

Parameters:

CT_image 3D CT volume data

cam_pt Position of camera in terms of x, y and z co-ordinates

film_array 3D array containing points on a film

Returns:

Null

D.6.3 Variable Documentation

D.6.3.1 public static int NR_END = 1

D.6.3.2 public static double[] CT_orig = {0, 0, 0}

D.6.3.3 public static double[][] DRR_out_im = new double [256][256]

D.6.3.4 public static double[] d_vox = new double[3]

D.6.3.5 static BufferedImage bi

D.6.3.6 static double[] ct_farcnr = {0, 0, 0}

D.6.3.7 int count1 = 0

D.6.3.8 static int blank

D.7 InsertionSort Class Reference

D.7.1 Detailed Documentation

This class is used for implementing insertion sort.Insertion sorting is used in ray tracing function to sort the values to write the film array.

Functions

- public void insertion (int p, int n_alpha, double [] alpha_range)

D.7.2 Function Documentation

D.7.2.1 public void insertion (int p, int n_alpha, double [] alpha_range

Parameters:

p Starting value of the array

n_alpha Last value of the array

alpha_range Array to be sorted

Returns:

Null

D.8 vtkMarchingCubes Class Reference

D.8.1 Detailed Documentation

This class is used to create a three dimensional image of the CT data. It uses VTK (Visualization Toolkit) functions to read in the datafile. The datafile is in .vtk format with the values converted to unsigned character datatype. VTK header is added to the datafile to describe the type of the data representing the CT joint.

Class Variables

- vtkPanel pan
- vtkStructuredPointsReader reader
- vtkPieceWiseFunction opacityTransferFunction
- vtkColorTransferFunction colorTransferFunction

- vtkVolumeProperty volumeProperty
- vtkVolumeRayCastCompositeFunction compositeFunction
- vtkVolumeRayCastMapper volumeMapper
- vtkVolume volume
- vtkContourFilter skinExtractor
- vtkPolyDataNormals skinNormals
- vtkPolyDataMapper skinMapper
- vtkActor outline
- vtkActor skin
- vtkOutlineFilter outlineData
- vtkPolyDataMapper mapOutline
- vtkTextProperty tprop

Constructors

- public vtkMarchingCubes()

Functions

- public void repaintImage (double x_generate, double y_generate, double z_generate, double azimuth_generate, double elevation_generate)

D.8.2 Constructor Documentation

D.8.2.1 public vtkMarchingCubes()

Creats a panel to display the three dimensional CT image. The 3D image is displayed in a new window with the bounding box around the image and the X, Y and Z co-ordinate axes.

D.8.3 Function Documentation

D.8.3.1 public void repaintImage (double x_generate, double y_generate, double z_generate, double azimuth_generate, double elevation_generate)

Repaints the panel to display a new 3D image in the window. This method is called when the user changes the value of the sliders and clicks on "Match" button. With the new values for the six degrees of freedom, the CT bone is also aligned with respect to the values selected. The x, y and z determine the position of the CT bone. Azimuth and elevation determine the angles of the camera.

Parameters:

x_generate X co-ordinate of the position of the CT cube

y_generate Y co-ordinate of the position of the CT cube

z_generate Z co-ordinate of the position of the CT cube

azimuth_generate Azimuth angle of the camera

elevation_generate Elevation angle of the camera

Returns:

Null

D.8.4 Variable Documentation

D.8.4.1 vtkPanel pan

D.8.4.2 vtkStructuredPointsReader reader

D.8.4.3 vtkPieceWiseFunction opacityTransferFunction

D.8.4.4 vtkColorTransferFunction colorTransferFunction

D.8.4.5 vtkVolumeProperty volumeProperty

D.8.4.6 vtkVolumeRayCastCompositeFunction compositeFunction

D.8.4.7 vtkVolumeRayCastMapper volumeMapper

D.8.4.8 vtkVolume volume

D.8.4.9 vtkContourFilter skinExtractor

D.8.4.10 vtkPolyDataNormals skinNormals

D.8.4.11 vtkPolyDataMapper skinMapper

D.8.4.12 vtkActor outline

D.8.4.13 vtkActor skin

D.8.4.14 vtkOutlineFilter outlineData

D.8.4.15 vtkPolyDataMapper mapOutline

D.8.4.16 vtkTextProperty tprop

D.9 BiomedicalMatching Class Reference

D.9.1 Detailed Description

This is the main class for the program. It creats objects for vtkMarchingCubes class and GUI class and designs a frame to hold the GUI.

Class Variables

- static VtkMarchingCubes iron
- static Gui slice

Functions

- public static void main (String[] args)

D.9.2 Function Description

D.9.2.1 public static void main (String[] args)

The main function of the program creates a frame for the GUI and initializes the objects for vtkMarchingCubes class and GUI class.

D.9.3 Variable Documentation

D.9.3.1 static VtkMarchingCubes iron

D.9.3.2 static GUI slice