3D RECONSTRUCTION TOOL FOR CT (SIMILAR) SCANNED IMAGES

VAJIRA SUNJEEWAN KULATUNGA

M.Sc. 2012

3D RECONSTRUCTION TOOL FOR CT (SIMILAR) SCANNED IMAGES

A PROJECT REPORT PRESENTED BY

E.R.G.V.S. KULATUNGA

[PGIS/SC/M.Sc/CSC/08/1]

to the Board of Study in Statistics & Computer Science of the

POSTGRADUATE INSTITUTE OF SCIENCE

In partial fulfillment of the requirements
for the award of the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

of the

UNIVERSITY OF PERADENIYA

SRI LANKA

2012

DECLARATION

I do hereby declare that the work reported in this project report was exclusively carried out by me under the supervision of Dr. J. Wijayakulasooriya. It describes the results of my own independent research except where due reference has been made in the text. No part of this project report has been submitted earlier or concurrently for the same or any other degree.

Date: ……..……..….. ……...……
Signature of the Candidate

Certified by:

1. Supervisor (Name): Dr. J. Wijayakulasooriya Date: …….……..

(Signature): ……

2. Supervisor (Name): ……. Date: ……..

(Signature): ……

PGIS Stamp:

3D RECONSTRUCTION TOOL FOR CT (SIMILAR) SCANNED IMAGES

E R G V S Kulatunga
Postgraduate Institute of Science,
University of Peradeniya,
Sri Lanka.

This report presents the design and implementation of a prototype for volume visualization system which can isolate a certain part(s) of MRI (or similar) images and generate 3D representation of the selected part(s). The work involves developing a critical software component for image processing and 3D transformation. System mainly implemented base on SF type Contour connecting algorithm. It also relies on primitive image processing algorithms like Closing, Opening, Erode, and Dilate. The developed prototype works according to the expected specifications. It is therefore anticipated that the prototype could be enhanced and implemented as a fully functional volume visualization system.

Acknowledgments

It is a great pleasure to express my sincere thanks and deep gratitude to my project supervisor Dr Janaka Wijayakulasooriya, for his supervision, continuous guidance, encouragement and all the helpful provided through various stages of this research.

Also, I would like to express my thanks to Dr Athula Perera, Dr Amalka J. Pinidiyaarachchi and all other PGIS staff members for their valuable guidance and support.

Finally, I would like to express my sincere gratitude to my wife Lasanthi for her love and understanding over the past months, and also my other family members for their support, advices and encouragements.

Table of Contents

CHAPTER 1 1

1.1 Overview 1

1.2 Problem Statement 2

1.3 Proposed Solution 2

1.4 Report Outline 2

CHAPTER 2 3

2.1 Direct Volume Rendering (DVR) 3
2.1.1 Ray-casting 3
2.1.2 Splatting 5
2.1.3 Surface *Fitting (SF)* 6
2.1.4 Contour connecting 6
2.1.5 Marching cubes 6
2.1.6 Dividing cubes 7

2.2 Voreen 7

2.3 Adopted Technique 7

CHAPTER 3 8

3.1 MeshCreator 8

3.2 Anatomy of MeshCreator 9
3.2.1 Label A and B: Files containing image slides 9
3.2.2 Label C and E : Unprocessed image slides 10
3.2.3 Label D : Image processing 10
3.2.4 Label F : Selecting interested objects 11
3.2.5 Label G : Highlighted interested objects 13
3.2.6 Identifying and marking of object's boarder 14
3.2.7 Label H, I, J, K: Propagate interested objects 15
3.2.8 Propagation mechanism 15
3.2.9 Label L: Generate volumetric Dataset 15
3.2.10 Check for slide deviations 17
3.2.11 Label M : Final volumetric dataset exported to MRIC file 20

3.3 Structure of a MRIC file 20

CHAPTER 4 22

4.1 3DView 22

4.2 Anatomy of 3DView 22
4.2.1 Label *A : Initialize 3D view generation* 22
4.2.2 Label B : Main 3D rendering loop 24
4.2.3 3D rendering and triangles 25

4.2.4 Lighting setup for the system 26
4.2.5 Material setup for the system 27
4.2.6 Label C : 3D view 27
4.2.7 Label D : 3D view controller 29

CHAPTER 5 31

5.1 MeshCreator component 31
5.1.1 Loading slides 32
5.1.2 Processing slides for objects selection 33
5.1.3 Example slide processing 33

5.2 Manipulating 3D view 39
5.2.1 Using mouse only 39
5.2.2 Using view controller 39

CHAPTER 6 41

6.1 Conclusions 41

6.2 Recommendations 42

REFERENCES 43

APPENDIX - CODING 44

6.1 Threshold Flood Fill 44

6.2 Rendering volumetric dataset 46

6.3 Saving volumetric dataset as a MRIC file 47

List of Tables

TABLE 3-1 UNEVEN CENTROID SHIFT DATA....................19

List of Figures

FIGURE 2-1 VOLUME RAY CASTING 5
FIGURE 2-2 DOLPHIN SKULL CREATED USING SPLATTING 5
FIGURE 2-3 HUMAN HEAD CREATED USING SPLATTING 5
FIGURE 3-1 MESHCREATOR 8
FIGURE 3-2 MESHCREATOR'S INTERNAL PROCESSES 9
FIGURE 3-3 IMAGE PROCESS PIPELINE 11
FIGURE 3-4 4-CONNECTED PIXELS 12
FIGURE 3-5 8-CONNECTED PIXELS 12
FIGURE 3-6 PIXEL COMPARISONS IN THRESHOLD FLOOD FILL 13
FIGURE 3-7 MARKING OF OBJECT BOARDERS 14
FIGURE 3-8 OBJECT PROPAGATION PROCESSES 15
FIGURE 3-9 HOW SLIDE ARE INTERNALLY STORED 16
FIGURE 3-10 SELECTED OBJECT 16
FIGURE 3-11 SLIDE DEVIATION DIALOG BOX 18
FIGURE 3-12 STRUCTURE OF A MRIC FILE 21
FIGURE 4-1 3DVIEW INTERNAL PROCESSES 23
FIGURE 4-2 STRUCTURE OF A SLIDE IN VOLUMETRIC DATASET 24
FIGURE 4-3 TRIANGLE WINDING 26
FIGURE 4-4 SCREEN SHOTS OF 3DVIEW COMPONENT 28
FIGURE 4-5 SCREEN SHOTS OF 3DVIEW COMPONENT 28
FIGURE 4-6 SCREEN SHOTS OF 3DVIEW COMPONENT 29
FIGURE 4-7 3D VIEW CONTROLLER 29
FIGURE 5-1 MESHCREATOR'S MAIN WINDOW 31
FIGURE 5-2 LOAD SLIDES BUTTON 32
FIGURE 5-3 AFTER LOADING SLIDES (BOTTOM IS A MAGNIFIED VIEW) 32
FIGURE 5-4 OBJECT GOING TO SELECT 33
FIGURE 5-5 APPLY INVERT FILTER 34
FIGURE 5-6 APPLY OPENING 35
FIGURE 5-7 APPLY ERODE 35
FIGURE 5-8 TOOLBAR 36
FIGURE 5-9 SELECTING AN OBJECT 36
FIGURE 5-10 3DVIEW COMPONENT 38
FIGURE 5-11 3DVIEW AFTER LOADING A VOLUMETRIC DATASET 38
FIGURE 5-12 VIEW CONTROLLER 39

List of Abbreviations

DECOM	Digital Imaging and Communications in Medicine
MRI	Magnetic resonance imaging
MRIC	Magnetic resonance image cube (Abbreviation used by researcher for the file used to save volumetric dataset)
CT	Computed Tomography
SF	Surface Fitting
VOXEL	Volumetric Picture Element
DVR	Direct Volume Rendering

CHAPTER 1

Introduction

1.1 Overview

At present, volume visualization is considered to be a highly evolving field and it is essential for the medical and numerous fields for their operation and development. Volume visualization demands intensive processing power of both CPU and GPU. So the advancements in computer hardware sector (Specially CPU and Graphic card) greatly help to practically apply volume visualization technique in wide range fields. This in turn greatly helps to advancements in many other fields.

Research volume visualization technology tries to implement the basic functionality of a volume visualization system as a prototype. This should be an affordable and run on non-custom hardware. The research has the following objectives.

1. Investigate existing volume visualization technology.
2. Examine the hardware and software requirements of a volume visualization System.
3. Produce a suitable prototype of a volume visualization System.
4. Test the prototype.

An extensive literature review was done on existing volume visualization systems and various 3D rendering systems. To develop those systems mainly developers use Microsoft DirectX or OpenGL. It was found that DirectX requires both high end and latest hardware. It is also a proprietary product. OpenGL is open source and has richer community support. So a final decision was made to go with OpenGL for the system. To get basic image processing functionality system uses A Forge framework.

1.2 Problem Statement

In industry, most companies and even in government sector use proprietary volume visualization systems. They are very costly and typically require high performance custom made hardware to run. Sometime required special hardware also manufactured by the same company or their partners. Technologies behind those systems are unknown and there is limited customizability in those systems. The best solution for these problems is encouraging open source developments.

There are ongoing open source projects. Most of those open source projects were started as researches conducted by various universities. There should be more contribution to the developing of the volume visualization sector. This research tries to give a contribution to those open source attempts.

1.3 Proposed Solution

To avoid these problems and to support open source community, this volume visualization system is created from the scratch. This system does not exactly follow any existing volume rendering technique. Algorithm used in this system can be categorized under Surface Fitting (SF) volume rendering algorithms. External libraries used for primitive operations. 3D rendering was done using open source OpenGL platform. Microsoft .Net frame was mainly used for creating the user interface. System is implemented in way which is both light-weights, fully customizable and high in performance. So the system can be run even in low performance hardware.

1.4 Report Outline

This project report contains six chapters:

1. **Chapter 1** : This chapter introduces the motivation of the research, where it comes from, what it is about, and the structure of the report.
2. **Chapter 2** : Examines background material and previous work relating to volume visualization.
3. **Chapter 3** : Provided technical detail of the component used to generate volumetric dataset form image slides.
4. **Chapter 4** : Provided technical detail of the component used to generate 3D view from volumetric dataset.
5. **Chapter 5** : Conclusion of the report

CHAPTER 2

Literature Review

The planned work in this project is related to volume visualization as the work tries to visualize or render a volume dataset in 3D scale. There are a number of algorithms for rendering volume data sets. At this level analyzed algorithms can be categorized based on the portion of the volume raster set which they render. This categorization separates them into *direct volume rendering* and *surface fitting algorithms* [16].

2.1 Direct Volume Rendering (DVR)

DVR methods are characterized by mapping elements directly into screen. Those algorithms do not use geometric primitives like triangles as an intermediate representation. DVR methods are especially appropriate for creating images from datasets containing amorphous features like clouds, fluids, and gases. One disadvantage of using DVR methods is that the entire dataset must be traversed each time an image is rendered. This is both time consuming and processor intensive. The process of successively increasing the resolution and quality of a DVR image over time is called progressive refinement. Followings are examples for DVR algorithms.

2.1.1 Ray-casting

Ray casting algorithms focuses on casting a ray from each pixel into the volume, sample the volume at regular intervals along the ray, and then compose these samples to produce the final color seen at the pixel.

Initially developed volume ray-casting algorithm, that is called as the first algorithm [12] only detects the first intersection of the ray with the data set (a binary classification), and so can only show the data's outer surface. It uses a simple depth cueing illumination model. Within the same year developed another method [14], which gives a ray-tracing method for volume densities in the context of rendering clouds and other particle systems. Later introduced method which is the ***single-scattering shading model***, a commonly used volume-rendering illumination model. The volume is considered to be composed of many small luminescent particles. The illumination emitted by each particle may scatter off of a neighboring particle, but

only one level of this scattering is modeled due to the complex process of illumination [8]. There is an assumption that the volume raster is a set of samples of a trilinearly-varying scalar field [10]. Also introduce the idea of ***transfer functions***, where each visible data parameter (such as the red, blue, green, and opacity channels) is an arbitrary function of the scalar data field. By manipulating the transfer functions it is possible to generate many different images of the same data set, and thereby gain a deeper understanding of the underlying data. The Levoy ray-casting method combines many features of the above techniques [17]. In addition, it gives non-binary methods for displaying surfaces from the scalar field. The most often used volume visualization algorithm for the production of high-quality images is ray-casting and commonly the following stages are carried out in ray casting.

Step 1. Ray casting - For each pixel of the final image, a ray of sight is cast through the volume. At this stage it is useful to consider the volume being touched and enclosed within a bounding primitive, a simple geometric object usually a cuboids that is used to intersect the ray of sight and the volume.

Step 2. Sampling - Along the path of the ray of sight that lies within the volume, equidistant sampling points or samples are selected. In general, the volume is not aligned with the ray of sight, and sampling points will usually be located in between voxels. Because of that, it is necessary to interpolate the values of the samples from its surrounding voxels (commonly using trilinear interpolation).

Step 3. Shading - For each sampling point, a gradient of illumination values is computed. These represent the orientation of local surfaces within the volume. The samples are then shaded according to their surface orientation and the location of the light source in the scene.

Step 4. Compositing - After all sampling points have been shaded, they are composited along the ray of sight, resulting in the final color value for the pixel that is currently being processed. The composition is derived directly from the rendering equation. It works back-to-front, i.e. computation starts with the sample farthest from the viewer and ends with the one nearest to him. This work flow direction ensures that masked parts of the volume do not affect the resulting pixel.

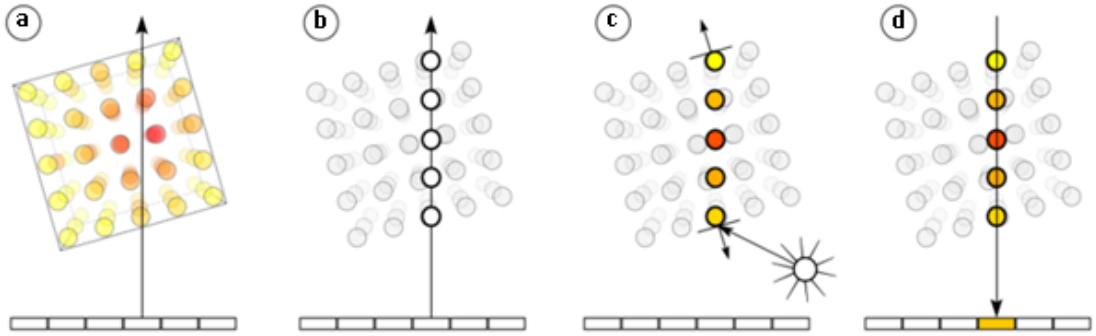

Figure 2-1 Volume ray casting

2.1.2 Splatting

A recently developed DVR algorithm is called splatting. Splatting performs a front-to-back object-order traversal of the voxels in the volumetric dataset. Each voxel's contribution to the image is calculated and composited using a series of table lookups. The procedure is called splatting because it is like throwing a snowball (voxel) at a glass plate. The snow contribution at the center of impact will be high and the contribution will drop off further away from the center of impact.

Figure 2-2 Dolphin Skull created using splatting

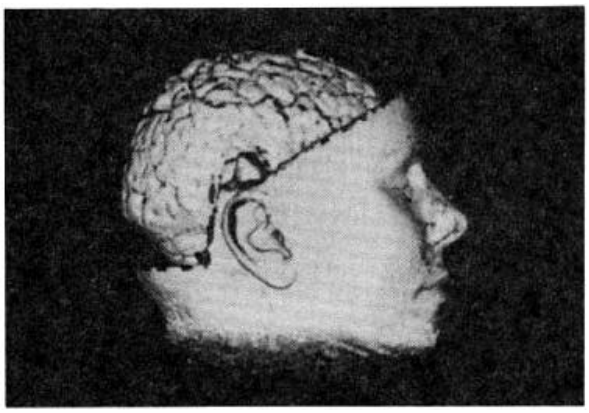

Figure 2-3 Human Head created using splatting

2.1.3 Surface *Fitting (SF)*

Algorithms of this category try to construct constant value contour surfaces in volumetric dataset using geometric primitives like triangles. The user begins by choosing a threshold value and then geometric primitives are automatically fit to the high contrast contours in the volume that match the threshold. Cells whose comer values are all above the chosen threshold (cell is inside) or all below the threshold (cell is outside) are discarded and have no effect on the final output. Showing just the cells falling on the threshold is sometimes useful, but can be a problem. Another thing to consider is the huge number of surface primitives generated for large volumetric datasets. Followings are examples for SF algorithms.

2.1.4 Contour connecting

One of the first-invented methods for generating volume visualization images and descendents of the contour-connecting algorithm are still in use. The basic idea of tracing one closed contour in each slice of data and then connecting contours in adjacent slices of data and refined.

After the user has specified a threshold value, a closed-curve contour at this value is found for each data slice. Originally, the contour-tracing step was performed by hand. Image processing techniques now provide methods for automatically generating the contours

Once the slice contours are found, the problem becomes one of finding an optimal tessellation, usually of triangles, connecting the curves in each two adjacent slices. The last step is to select viewing, lighting, and rendering parameters, and to pass the strips of triangles to a surface renderer.

2.1.5 Marching cubes

Marching Cubes is an algorithm for rendering isosurfaces in volumetric data published in the 1987. The basic notion is that we can define a voxel(cube) by the pixel values at the eight corners of the cube. If one or more pixels of a cube have values less than the user-specified isovalue, and one or more have values greater than this value, we know the voxel must contribute some component of the isosurface. By

determining which edges of the cube are intersected by the isosurface, we can create triangular patches which divide the cube between regions within the isosurface and regions outside. By connecting the patches from all cubes on the isosurface boundary, we get a surface representation.

2.1.6 Dividing cubes

This is similar to the Marching Cubes algorithm except that when a voxel(cube) is determined to be intersected by the isosurface, it is projected to the image plane. If it projects to a single pixel or smaller, than it is just rendered as a point, else it is subdivided into surfaces as with the Marching Cubes algorithm.

2.2 Voreen

Voreen is a successful open source project aim to develop a high performance volume rendering system. The Voreen project has been initiated in 2005 at the University of Münster as subproject of the collaborative research center SFB 656, and is now developed cooperatively between the University of Münster and Linköping University[7].

Voreen project intention is not to create a light weight volume visualization system. Its aim is to produce high performance volume rendering system. This is a problem when this system is try to execute is a low end system. To avoid this limitation, the prototype implemented in this research mainly focused on low end systems.

2.3 Adopted Technique

When analysing these methods, the literature reveals that SF methods are typically faster than DVR methods since SF methods only traverse the volume once to extract surfaces. After extracting the surfaces, rendering hardware and well-known rendering methods can be used to quickly render the surface primitives each time the user changes a viewing or lighting parameter. Surface fitting algorithms seems fulfilling the objective set in this project work so more focus was on suface fitting algorithms.

Contour connecting algorithm, that is a type of suface fitting algorithms which is well known for its simplicity and the plethora of known surface-rendering methods. The surface-fitting portion of this algorithm is parallelizable since no two adjacent slices are dependent on the result of another pair [16].

CHAPTER 3

Generate Volumetric Dataset

The system has two major components that were built as separate projects. This chapter discuses architecture and the internal technical details of the first component called MeshCreator. Here mainly focusing on essential components of the MeshCreator. This component is used to construct volumetric dataset form image slides

3.1 MeshCreator

This component is used to construct volumetric dataset for volume rendering from the image slides. Figure 3-1 is a screen shot of MeshCreator graphical user interface.

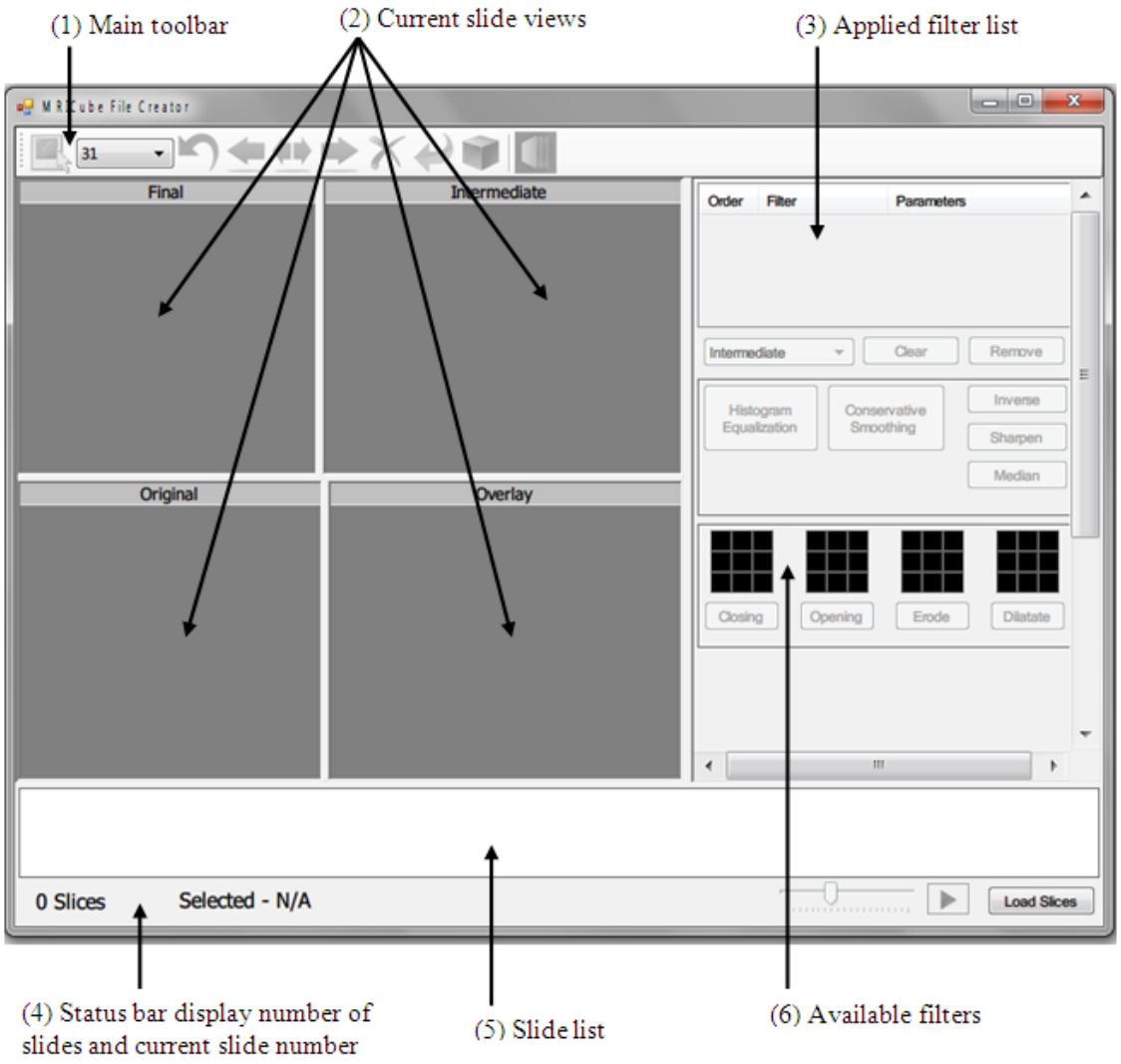

Figure 3-1 MeshCreator

3.2 Anatomy of MeshCreator

Figure 3-2 diagram illustrate detail view of the MeshCreator component. Diagram has labels for each step. Those labels will be used to describe MeshCreator.

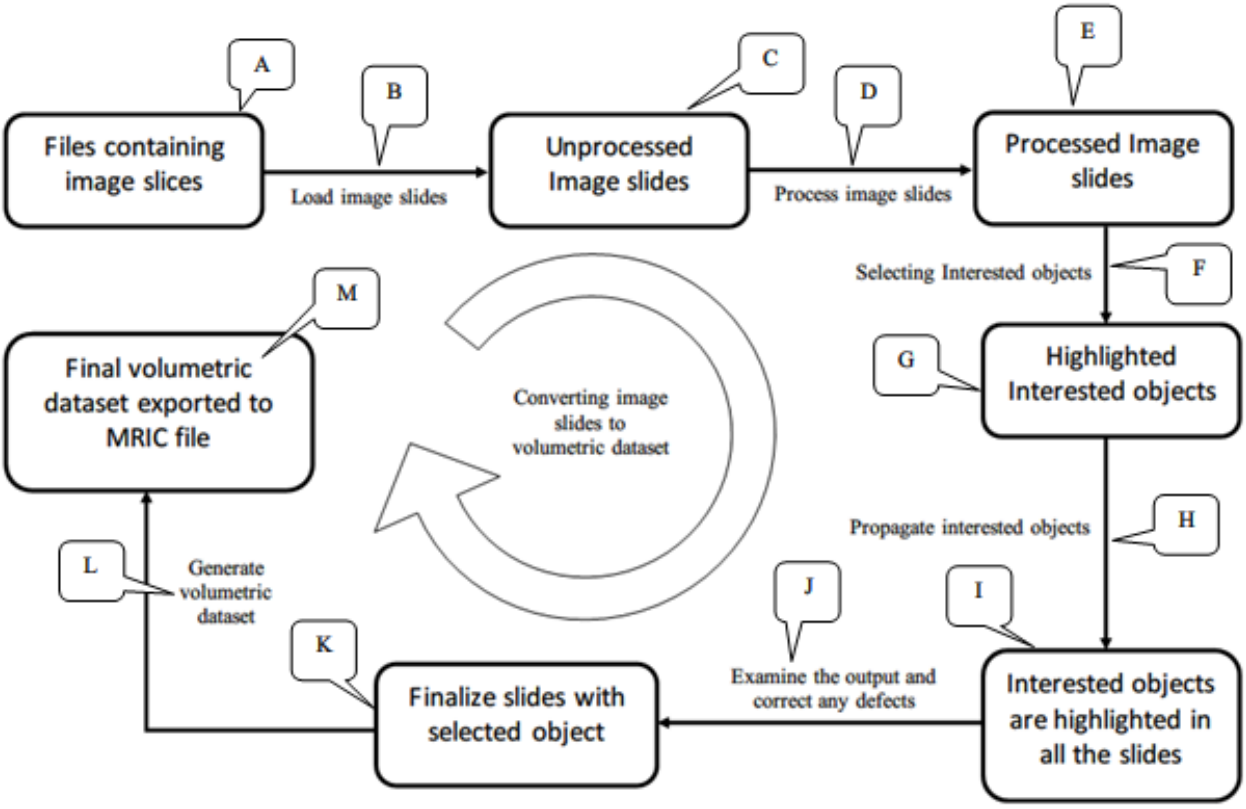

Figure 3-2 MeshCreator's internal processes

Following section describe each step in detail.

3.2.1 Label A and B: Files containing image slides

The main images used for testing of this prototype system were obtained by a CT scan of patient from Kandy general hospital.

They were in DICOM format. So for easy access they need to be converted in to a common image format. MATLAB was a great tool for getting the details and converting DICOM formats. As there were 300 image sides it was necessary to use a third party scripting tool for automating the conversion format. Script was written using AutoIt scripting tool

Output of this script was a series of BMP image. When fetching those images to the system, they must be in the same order as the order of the DICOM slides. So BMP images were named as 0, 1,2,3,4... 299. So just by using file name system can load then in correct order. System loads all the files at once.

3.2.2 Label C and E : Unprocessed image slides

Each slide can be access by using navigation bar at the bottom. Each slide has four views associated with it (Figure 3 1 MeshCreator).

a. Original view	:	Display slide as it is without any alteration.
b. Overlay view	:	Display cursor location as a red circle and any selected object in deferent colors.
c. Intermediate view	:	Display the result after first step image processing.
d. Final view	:	Display the result after second step image processing.

All four views will be the same for unprocessed slides.

3.2.3 Label D : Image processing

Following basic image processing technique can be applied to slides[4].

a. Closing
b. Opening
c. Erode
d. Dilate
e. Histogram equalization
f. Conservative smoothing
g. Inverse
h. Sharpen
i. Median

Image processing can be applied in two steps. This gives a better control over the whole processes. Ultimately allows for better isolation of objects.

When applying a filter, targeted step can be chosen from a dropdown list. That can be ether Intermediate or Final. Following diagram shows the process.

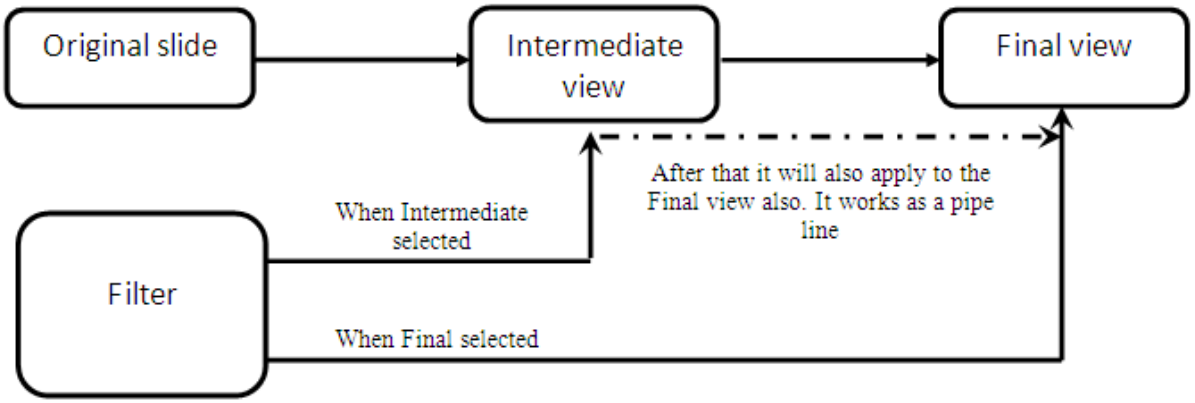

Figure 3-3 Image process pipeline

Custom kernel can be visually setup for the filters witch needs a kernel.

3.2.4 Label F : Selecting interested objects

Once satisfied with the processed final view, click on an interested object. System will try to select the interested object base on threshold controlled flood fill method. Here the threshold can be manually set. If the output is not satisfying selection can be undone and retry with a deferent settings. Following section will explain the flood fill method of the system.

3.2.4.(a) Flood fill method

Basically flood fill algorithms can be divided into two categories.

a. 4-Way method.
b. 8-Way method.

(a) 4-Way method.

Following is the pseudo code for 4-Way method.

```
Flood-fill (node, target-color, replacement-color):
 1. If the color of node is not equal to target-color, return.
 2. Set the color of node to replacement-color.
 3. Flood-fill (one step to the west of node, target-color, replacement-color).
    Flood-fill (one step to the east of node, target-color, replacement-color).
    Flood-fill (one step to the north of node, target-color, replacement-color).
    Flood-fill (one step to the south of node, target-color, replacement-color).
 4. Return.
```

Here four neighbors of each pixel checked for filling [1].

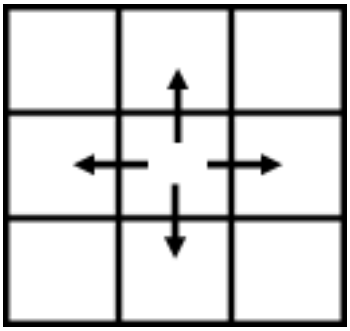

Figure 3-4 4-Connected pixels

(b) 8-Way method.

8-Way algorithm is very similar to the previous one, except it doesn't test 4 neighbors, but 8. This means, that this version of the flood fill algorithm will leak through sloped edges of 1 pixel thick. [1].

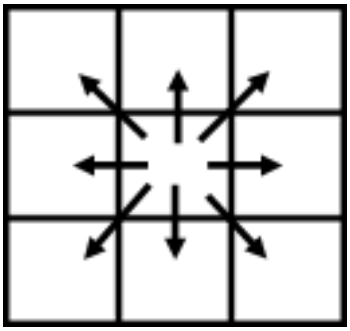

Figure 3-5 8-Connected pixels

By testing with the sample images it was found out that 8-Way method works better than 4-Way method for the system. So this system uses 8-Way method flood fill method.

3.2.4.(b) Threshold flood fill

Normally flood fill search for exact match with the target color. But in this system instead of target color it work with target color threshold. If the current pixel value is within that range, it will consider as belong to the object. Threshold value can be selected by the user.

Flood fill function is a recursive one. So there is a chance of stack overflow exception. To prevent this system uses a queue (data structure) base method. Objects selected using threshold flood fill will be highlighted in overlay view. Following is the flowchart for pixel comparison in threshold flood fill. (*6.1 Threshold Flood Fill*)

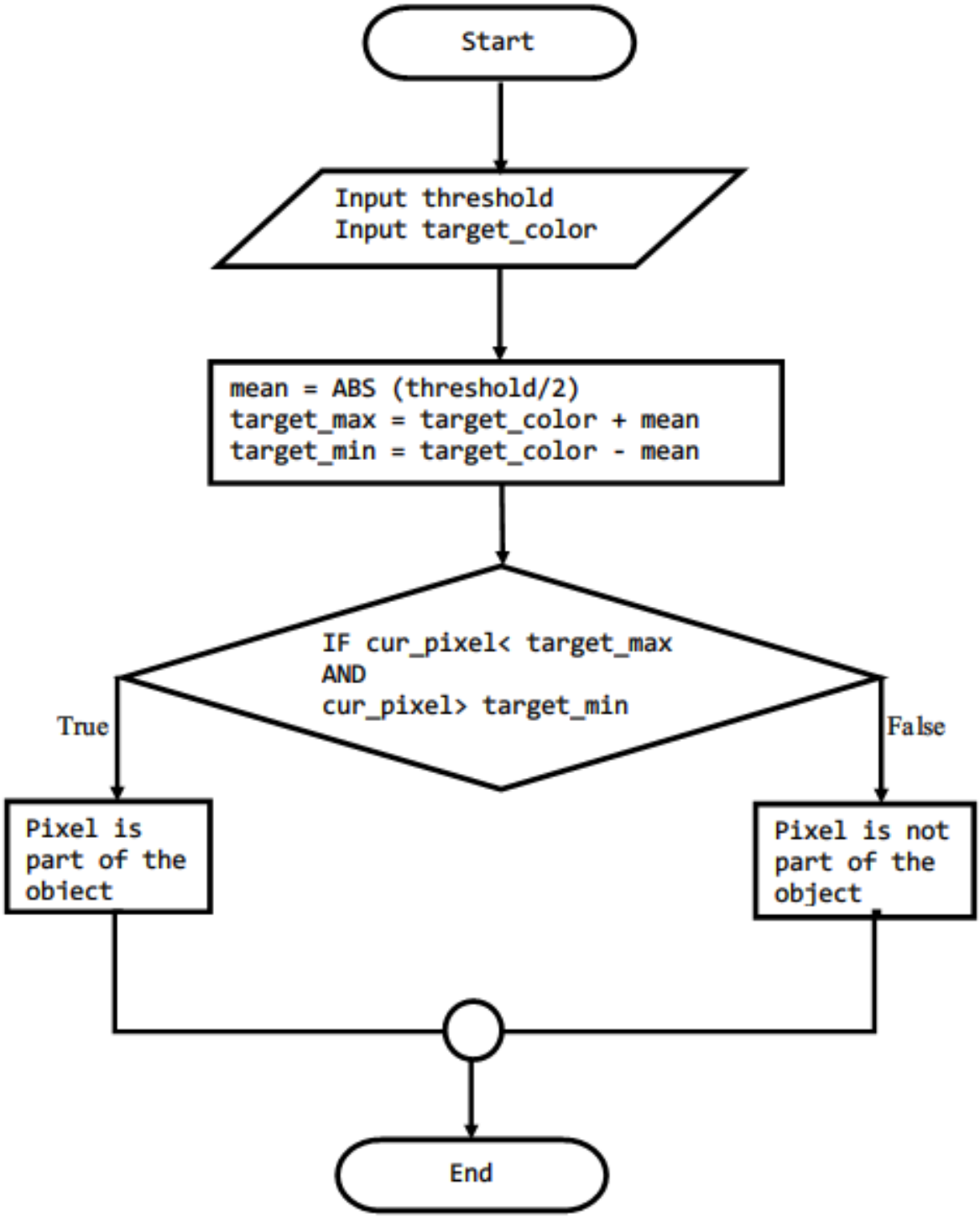

Figure 3-6 Pixel comparisons in threshold flood fill

3.2.5 Label G : Highlighted interested objects

Highlighted objects are internally marked in a newly created separate layer (object mask) of the side. This allows easy undoing and easy management of the object. Up to 10 objects can be selected in one set of slides. Each object will be highlighted with a unique color.

3.2.6 Identifying and marking of object's boarder

Once the object is selected, object's boarder identification process starts. Images are horizontally and vertically scan to identify object boarders. Now objects can be easily identified using object mask. When a boundary pixel is detected it will be marked in the object mask. Both horizontal and vertical scan require for detect all boarders in some objects. Correct boarder detection is essential for generate volumetric dataset.

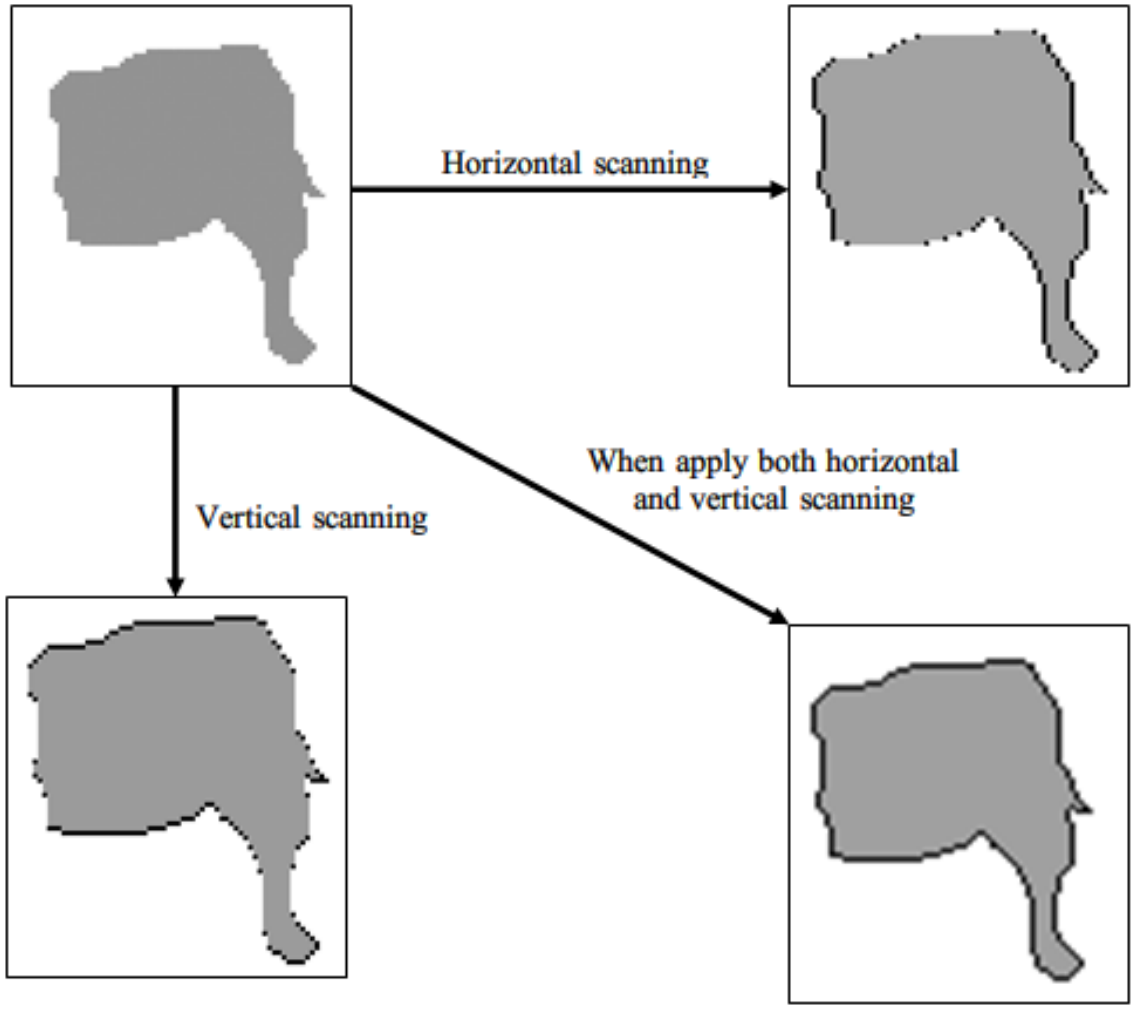

Figure 3-7 Marking of object boarders

3.2.7 Label H, I, J, K: Propagate interested objects

Object selection can be done with any selected image slide. To generate volumetric dataset, selected object must be propagated though all the slides.

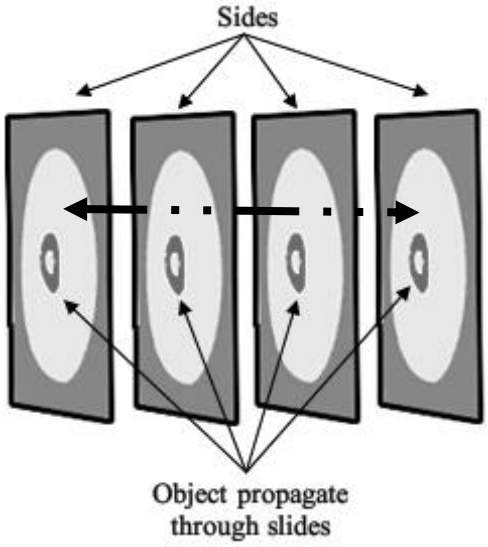

Figure 3-8 Object propagation processes

3.2.8 Propagation mechanism

At first image processing applied to the current slide only. Once an object is selected and start to propagate, same image processing methods will be automatically applied to all other slides. Propagation starts from the current slide to specified direction.

Here the first slide's object used as a mask for the next slide. Flood fill method applies to the second slide for the area covered by the first (previous) slide's object. And the process continues till the object ends. Each object should be propagated separately. Evan after propagating, object can be discarded by undoing.

3.2.9 Label L: Generate volumetric Dataset

By now all the objects are selected and object boarders marked. Entire slide series is internally stored in a special data structure called ByteCube. ByteCube is consisting of ByteImages which represent individual slides. As mentioned earlier (Label G : Highlighted interested objects) all objects are marked in a separated layer (mask) associated with each ByteImage.

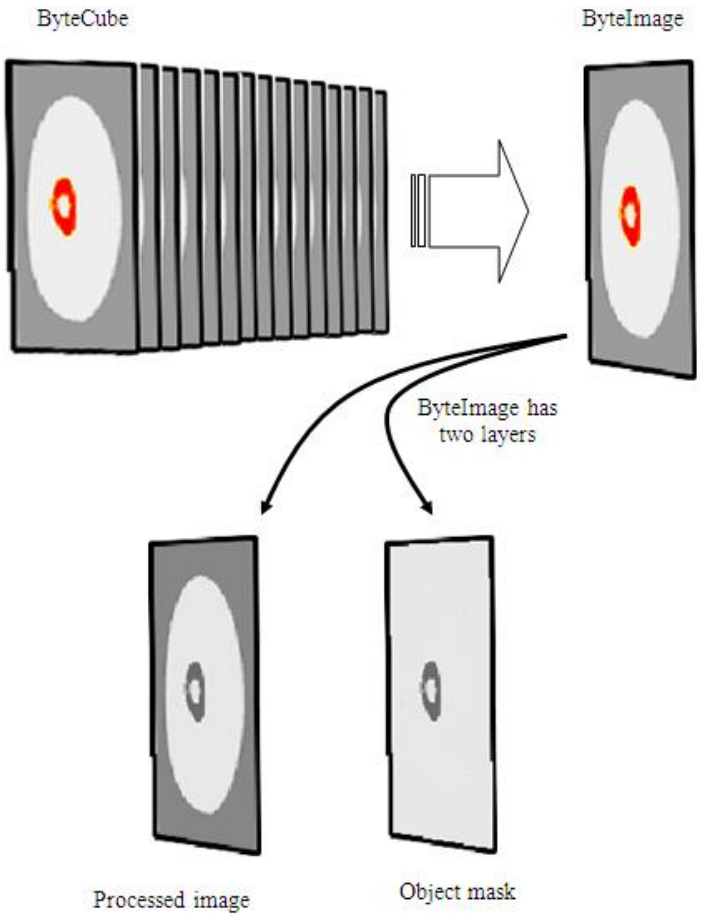

Figure 3-9 How slide are internally stored

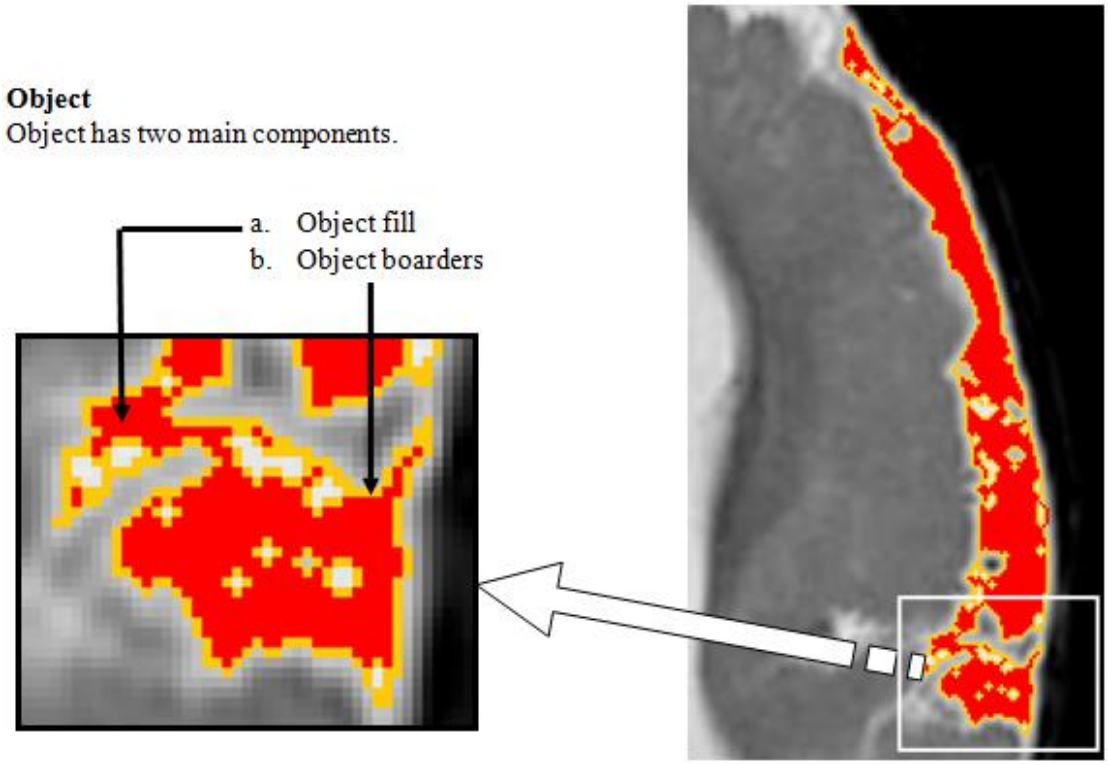

Figure 3-10 Selected object

a. **Object fill** - When an object is selected, object fill is found using threshold flood fill.
b. **Object boarders** - As illustrated in Figure 8, certain complex objects can have many boarders. When saving objects as a volumetric dataset, boarders must be saved in a certain order. For example scan pixels in counter clock vise and store them as vertex. Then only the system can produce volumetric view of the object.

3.2.10 Check for slide deviations

MRI and slides obtained by any other acquisition methods some times have an uneven slide deviation (shift) from one to another. This may be due to the movements of live objects like living body part when scanning. That can cause artificial ripples on the final 3D view. MeshCreator toolbar has a special button to check for any slide deviations (3.2.10 Check for slide deviations). System uses object's centroid of each slide to plot the deviation graph. Each object can be plotted separately. Following function is used to find each object's centroid.

```
public MyPoint FindCentroid()
{
    int xtot = 0;
    int ytot = 0;
    MySize s = m_bi.Size;
    byte[,] msk = m_bi.GetObjectMask();

    for (int x = 0; x < s.Width; x++)
    {
        for (int y = 0; y < s.Height; y++)
        {
            if ((msk[x, y] == m_fill_color)||(msk[x, y] == m_bor_color))
            {
                xtot += x;
                ytot += y;
            }
        }
    }
    m_centroid = new MyPoint(xtot/m_length,ytot/m_length);
    return m_centroid;
}
```

DICOM slides use for the testing of this system has uneven slide deviations in them. Following figure shows the slide deviation check dialog of the system.

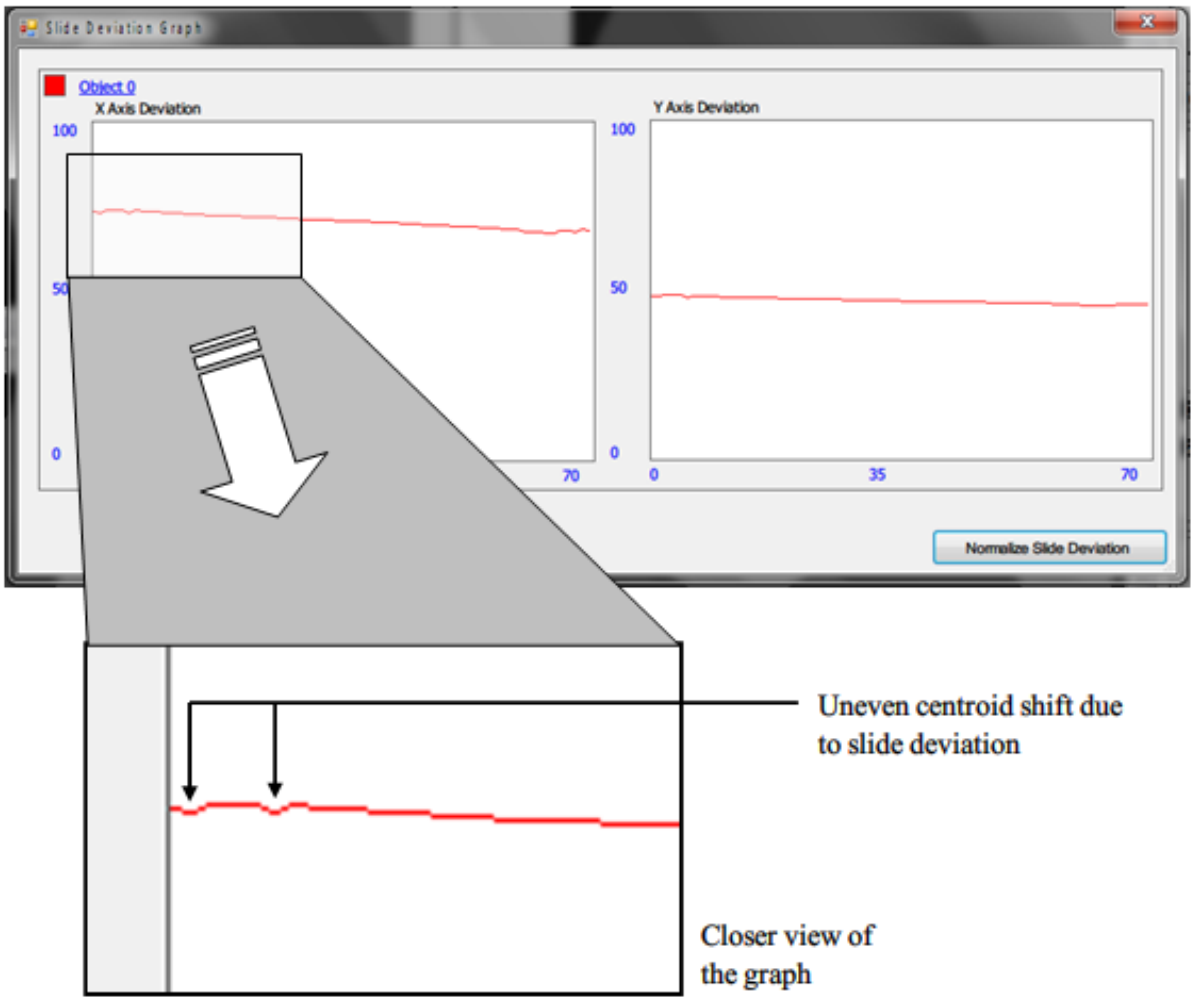

Figure 3-11 Slide deviation dialog box

Table 3-1 Uneven centroid shift data

Slide Number	X Coordinates	Y Coordinates
1	178	308
2	179	308
3	174	306
4	174	306
5	175	306
6	179	308
7	175	307
8	176	307
9	177	307
10	177	307
11	180	308
12	180	309
13	181	309
14	181	309
15	182	310
16	183	310
17	183	310
18	184	311
19	185	311
20	185	311
21	186	312
22	187	312
23	187	312
24	188	312
25	188	313
26	189	313
27	190	313
28	190	314
29	191	314
30	192	314
31	192	314
32	193	315
33	193	315
34	194	315
35	195	315
36	195	316
37	196	316
38	197	316
39	197	316
40	198	317
41	199	317
42	199	317
43	200	317
44	201	318
45	201	318

46	202	318
47	203	318
48	203	318
49	204	319
50	205	319
51	206	319
52	206	319
53	207	320
54	208	320
55	208	320
56	209	320
57	210	320
58	211	321
59	211	321
60	212	321
61	217	322
62	218	322
63	218	323
64	220	323
65	220	323
66	215	322
67	215	322
68	216	322
69	213	321
70	214	321

3.2.11 Label M : Final volumetric dataset exported to MRIC file

When creating volumetric dataset, first system scans mask of each slide until an object boarder is located. Then the system scans along the object's boarder clock vice till the boarder finishes. Each pixel will be stored in a clock vice order as a vertex in volumetric dataset. Finally, volumetric dataset is saved in a special file format called MRIC.

3.3 Structure of a MRIC file

The system uses its own file format to store volumetric dataset. As this is a prototype its helps to avoid the overhead of integrating other widely used 3D file format. It also maximizes the customizability. Later if needed this none standard format can be converted to any other standers format.(*6.3 Saving volumetric dataset as a MRIC file*)

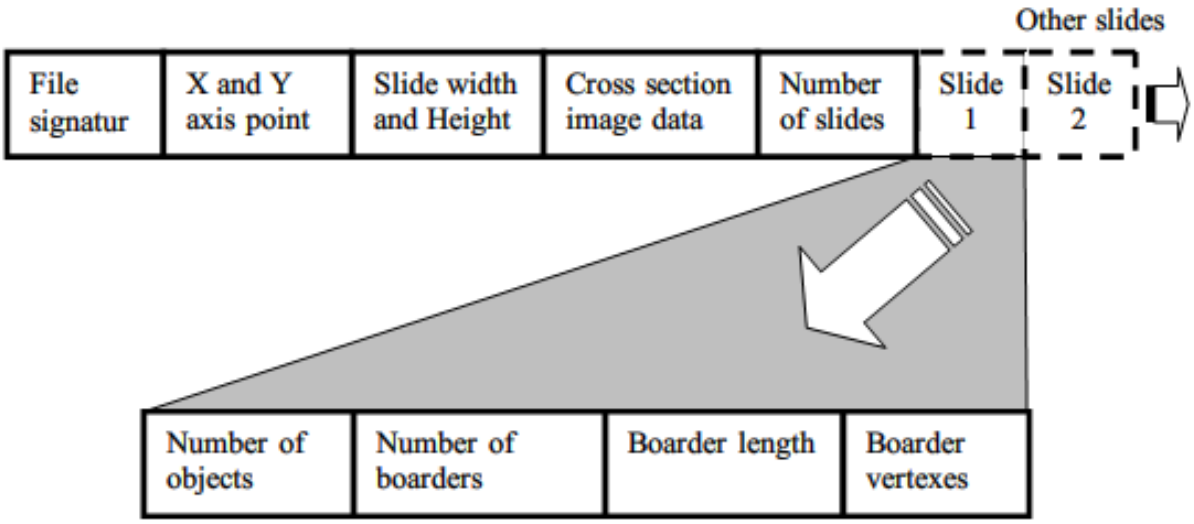

Figure 3-12 Structure of a MRIC file

Description of each field of the MRIC file

1. **File signature** – Special signature to uniquely identify MRIC file.
2. **X and Y axis point** – Axis coordinates. This can be manually set.
3. **Slide width and height** – All the slides must be the same size.
4. **Cross section image data** – Used by 3DView component of the system.
5. **Numbers of slides** – Number of slides stored in volumetric dataset. From there, each slide is separately stored one after another. Each slide consists of four fields.
6. **Number of objects** – Number of objects in the current slide.
7. **Number of boarders** – Number of boarders in the current object.
8. **Length of boarder** – Length of the current boarder.
9. **Boarder vertexes** – Boarder vertexes of the current boarder. After the next boarder length and vertexes. Then after finishing all the boarders next slide data.

CHAPTER 4

Generate 3D View

4.1 3DView

This chapter discussed architecture and the internal technical details of the second major component called 3DView. Here mainly focusing on essential sub components of the 3DView. This component use to construct 3D view from volumetric dataset (MRIC file) and manipulate it three dimensionally.

4.2 Anatomy of 3DView

Figure 4-1 illustrate detail view of the 3DView component. Diagram has labels for each step. Those labels will be used to describe each step.

4.2.1 Label *A : Initialize 3D view generation*

When loading a MRIC file, system checks for the valid signature. If the signature is a valid one system continues to load the file other vice it stops there.

Then the system loads the following basic information from the file.

1. Axis X and Y coordinates.
2. Width and height of the slides.
3. Cross section image data.
4. Number of slides

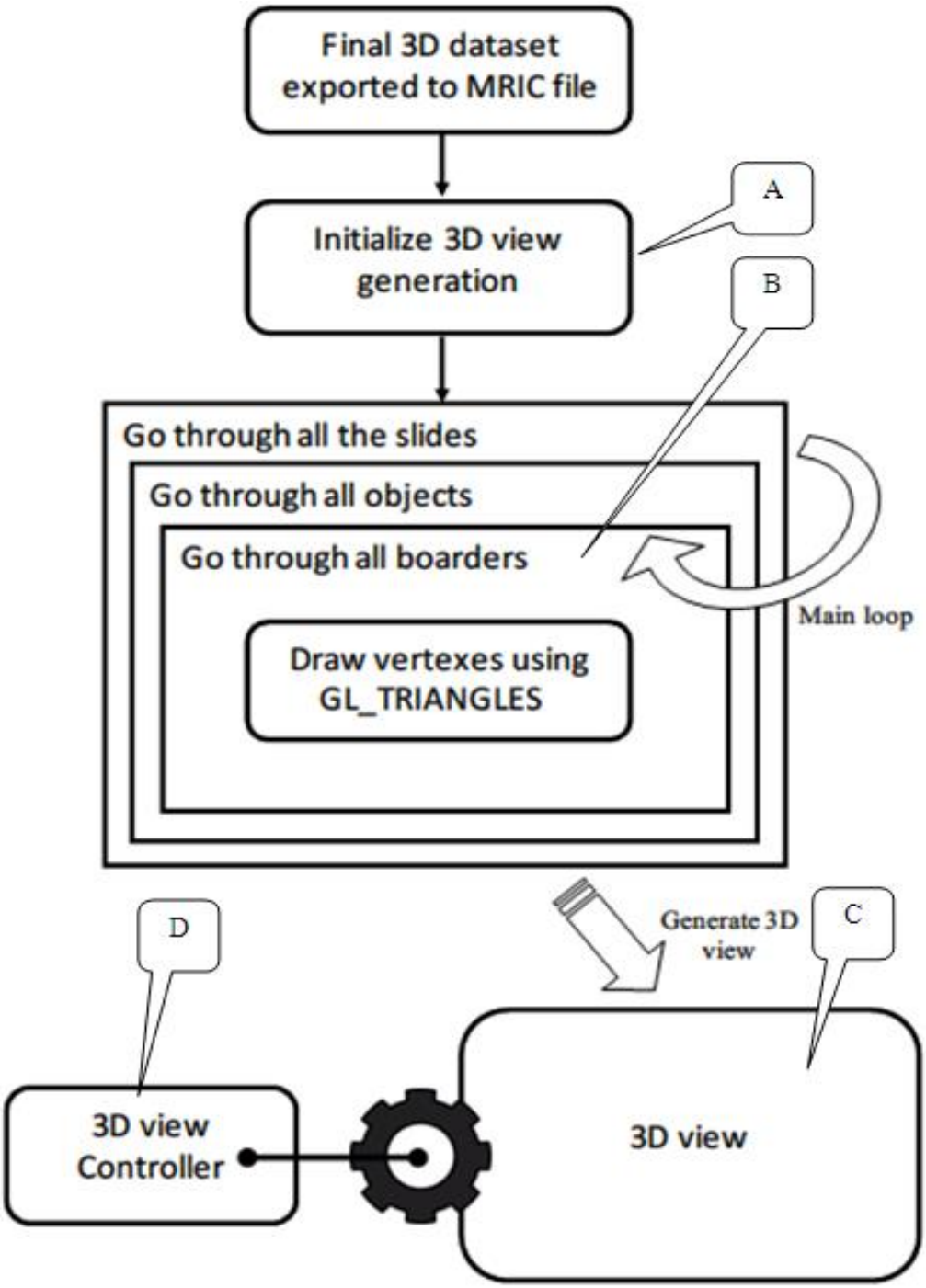

Figure 4-1 3DView internal processes

4.2.2 Label B : Main 3D rendering loop

Main rendering loop is set to run through all the slides saved in the volumetric dataset. Each slide contains object collection and each object contains boarder collection. System goes through all the slides, objects and each object's boarder collection.

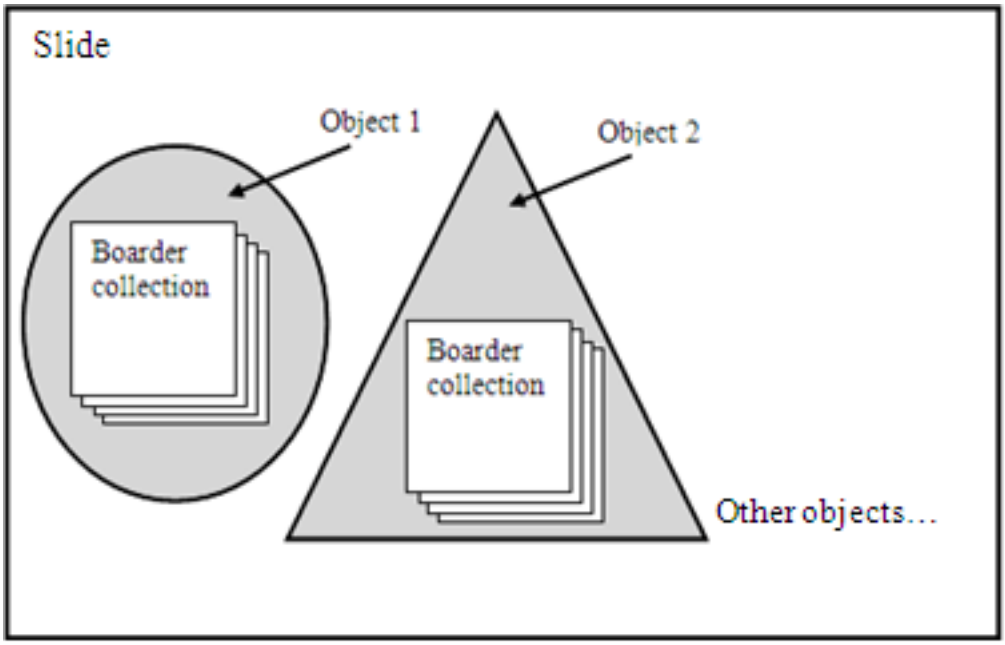

Figure 4-2 Structure of a slide in volumetric dataset

4.2.2.(a) Pseudo code of main rendering loop

Following is a greatly simplified main rendering loops pseudo code.(*6.2 Rendering volumetric dataset*)

```
FOR a = 1 TO nslides
    nobjects = GetNumberOfObjects()
    FOR b = 1 TO nobjects
        nbranches = GetNumberOfBoarders()
        FOR c = 1 TO nbranches
            nVertexes = GetNumberOfVertexes()
            FOR d = 1 TO nVertexes
                DrawGLTriangle(vertext1,vertext2,vertext3)
            NEXT
        NEXT
    NEXT
NEXT
```

4.2.3 3D rendering and triangles

Among other primitives like points, lines, quads, triangles are the main primitive for 3D rendering. There are a few reasons for that.

1. Modern hardwires handle triangle very efficiently.
2. OpenGL nicely calculate lightings.
3. It is a versatile primitive to create any kind of shape.

Initially this system is tested with point primitives. But the performance was unbearably poor. So the next prototype used triangles instead of points.

4.2.3.(a) Winding

OpenGL detect the front and back side of surface base on its winding direction. There are two types of windings [2].

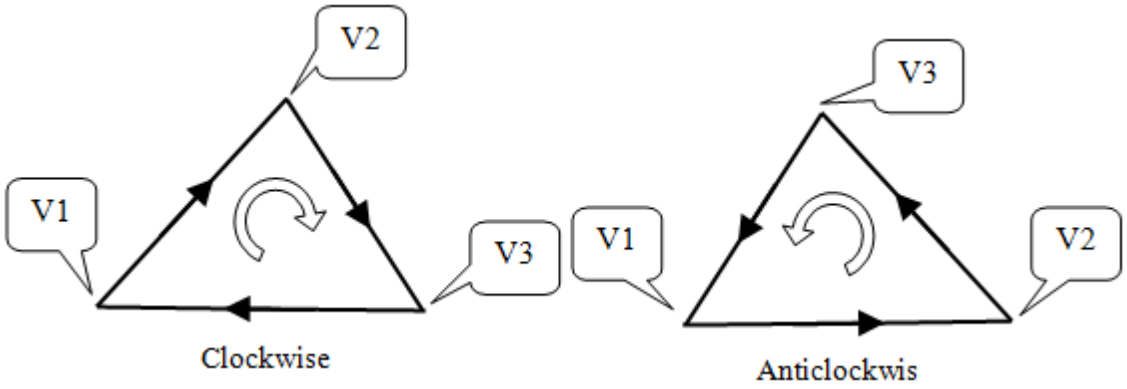

Figure 4-3 Triangle winding

OpenGL can be setup to ether clockwise front or anticlockwise front. This is very important when apply lighting. This system uses clockwise front mode. So the objects triangles must be draw in a way that all front faces get clockwise winding.

4.2.4 Lighting setup for the system

To produce a realistic look on final 3D view it is vital to enable lighting effects on OpenGL. There are three main kinds of light.

a. **Ambient light** - Ambient light does not come from any particular direction. It is a directionless light. Objects illuminated by ambient light are evenly lit on all surfaces in all directions.[2]

b. **Diffuse Light** - It is a directional light that appears to come from a particular direction. This type of light reflected off a surface with an intensity proportional to the angle at which the light rays strike the surface. So the object surface is brighter if the light is pointed directly at the surface than if the light grazes the surface from a greater angle.[2]

c. **Specular Light** - Specular light is a highly directional than diffuse light. But it interacts more sharply with the surface and in a particular direction. Specular light tends to cause a bright spot on the surface it shines on. Because of its highly directional nature, it is even possible that depending on a viewer's position.[2]

The system uses all three kinds of light to light up 3D objects. And there is a spot light source which produces specular light. Spot light has 360 degree Cut off angle. Direction and position of the spot light can be change by using 3D view controller

explain bellow (*Label D : 3D view controller*). This helps to observe 3D objects in a better way.

4.2.5 Material setup for the system

Lighting in OpenGL works close to gather with material properties of primitives. This volume visualization system uses color materials witch has high shininess. Following coding set the material properties.

```
GL.Enable(EnableCap.ColorMaterial);
// Set Material properties to follow glColor values
GL.ColorMaterial(MaterialFace.Front, ColorMaterialParameter.
AmbientAndDiffuse);
// All materials hereafter have full specular reflectivity with
a high shine
GL.Material(MaterialFace.Front, MaterialParameter.Specular, new
float[] { 1.0f, 1.0f, 1.0f, 1.0f });
GL.Material(MaterialFace.Front, MaterialParameter.Shininess, 128);
```

4.2.6 Label C : 3D view

Final output of the system is presented in the main window of 3DView component. There are two ways to manipulate 3D view. Ether using mouse movements and mouse buttons or by using view controller explains in the next section (*5.2 Manipulating 3D view)*. Figure 4-5, Figure 4-6 and Figure 4-7 are several screen shots of 3DView component

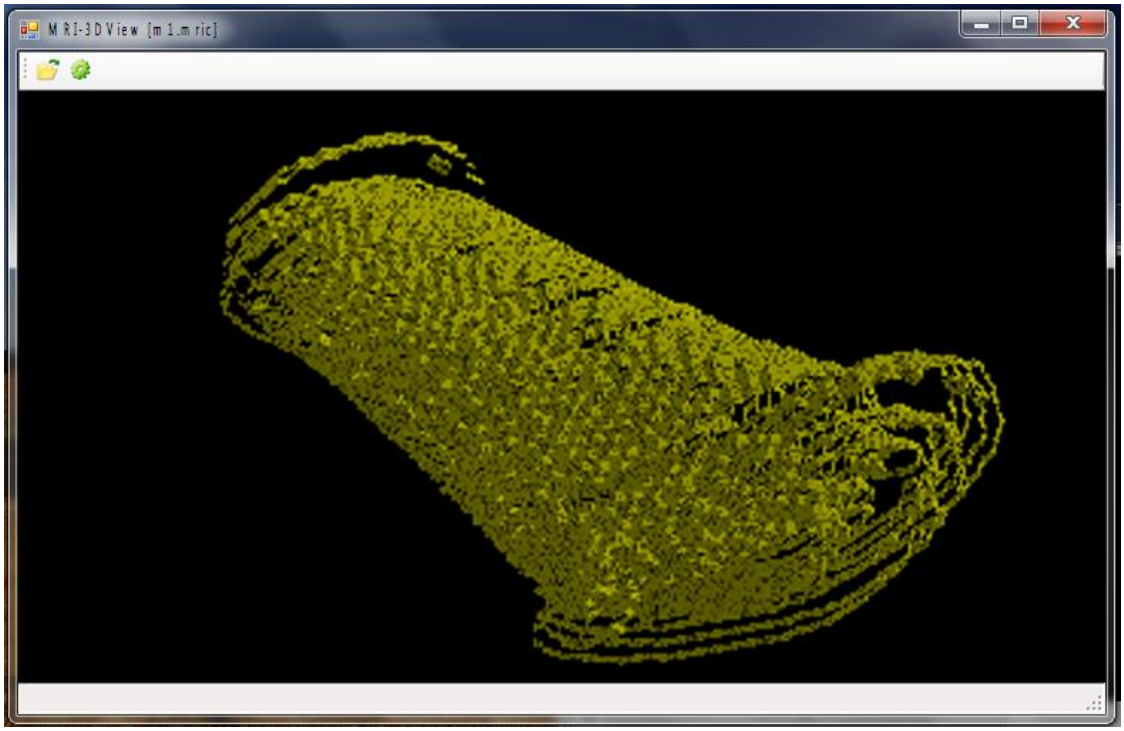

Figure 4-4 Screen shots of 3DView component

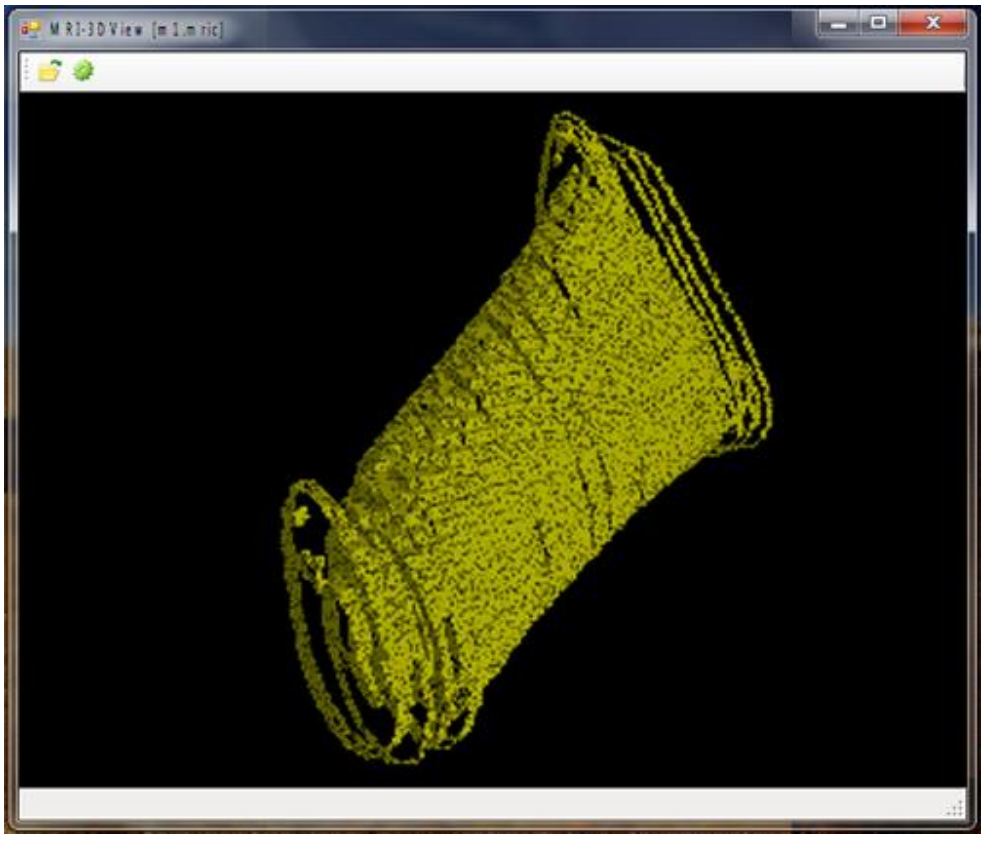

Figure 4-5 Screen shots of 3DView component

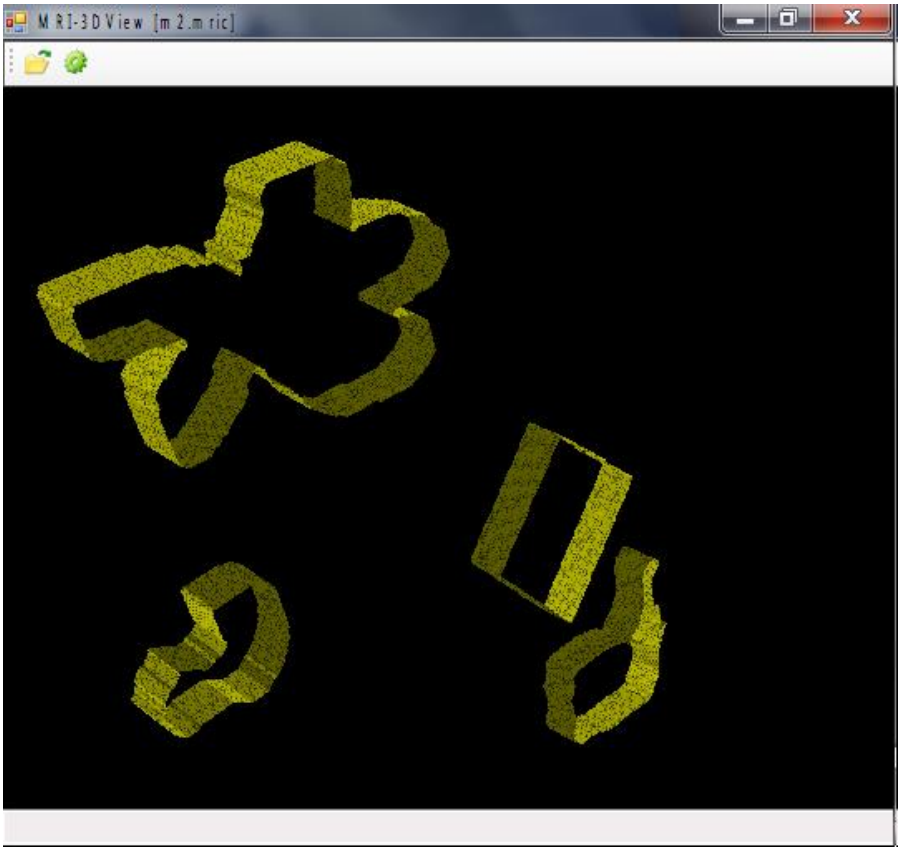

Figure 4-6 Screen shots of 3DView component

4.2.7 Label D : 3D view controller

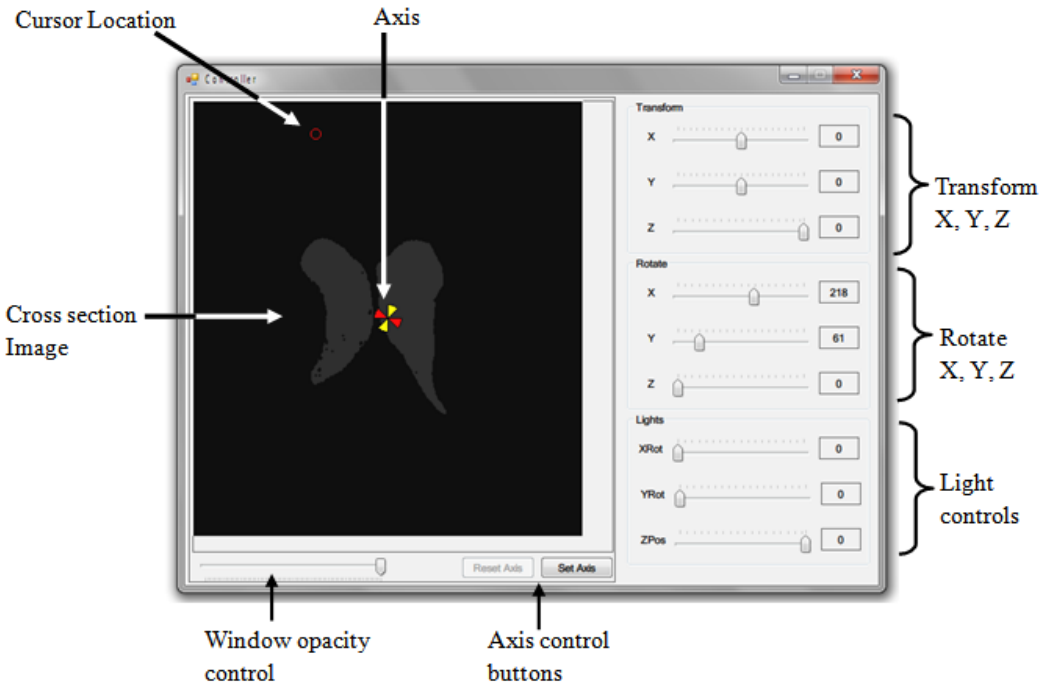

Figure 4-7 3D View Controller

This is a helper component that can be used to control the 3D view. Following diagram and labels will be used explain functionalities of the components.

This window can be used to control 3D view. By using this window user can do transform and rotate on X, Y, Z axis. Rotations are done base on axis. If needed, user can change axis using *Set axis* button. There is also a slider to change the opacity of the controller window. By default controller window opacity set to 80% so the user can see through controller window.

4.2.7.(a) Cross section image

This image is load from the *cross section image data* section of MRIC file (*3.3 Structure of a MRIC file*). This gives a cross section view of the objects using the center slide. This can be used to set common axis of the objects and can be useful when rotating objects in 3D view.

4.2.7.(b) Controllers for light source

There is a one specular light source in addition to the ambient and diffuse light. This light source can be controls using those controls.

CHAPTER 5

Practical usage of the system

This chapter is arranged in to two sections. First section describes the steps for creating 3D dataset and the second section describes how to visualize created volumetric dataset (MRIC file).

5.1 MeshCreator component

Following image illustrate GUI arrangement of the MeshCreator main window.

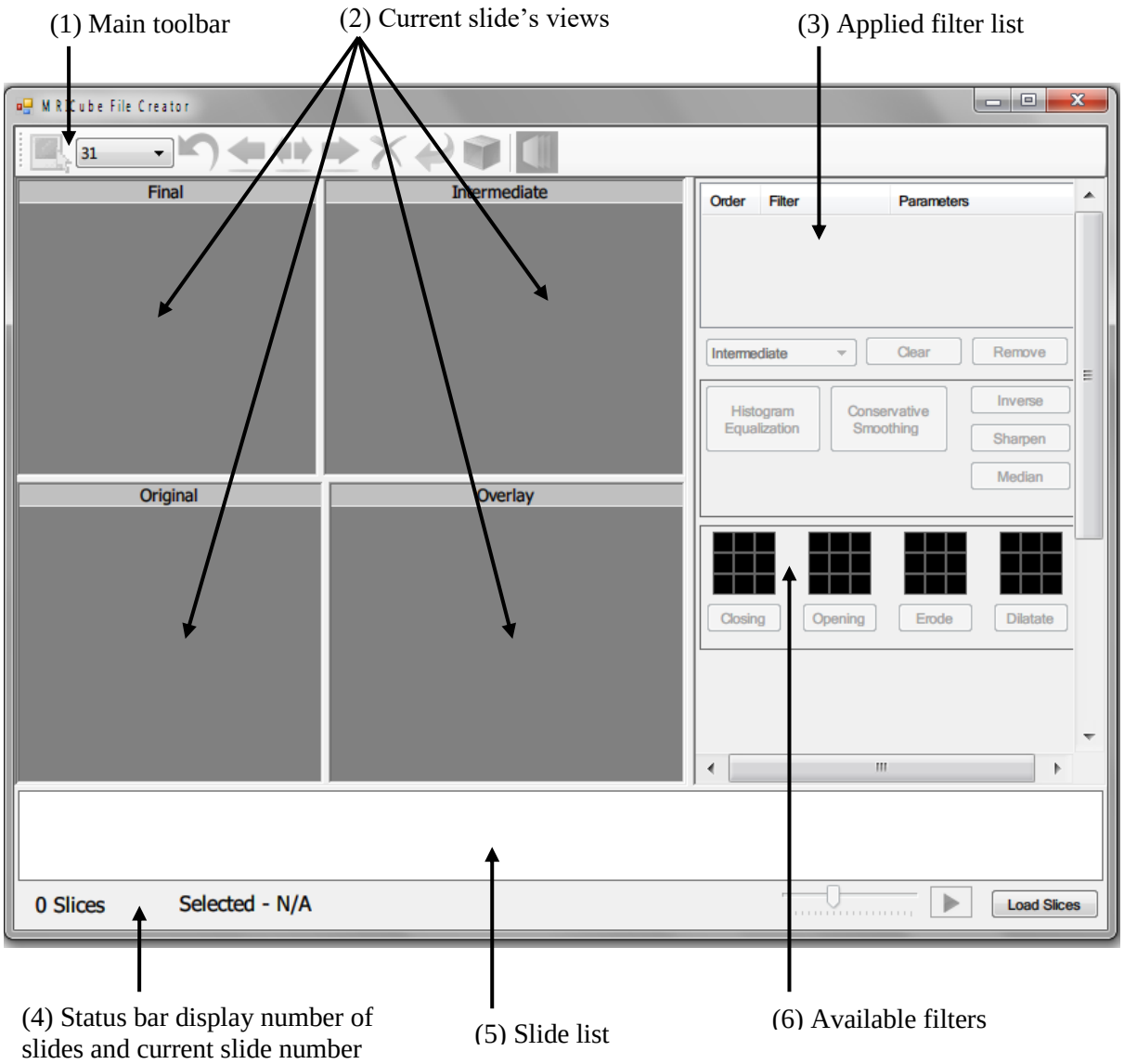

Figure 5-1 MeshCreator's main window

5.1.1 Loading slides

Click load slides button and select the folder containing slides. System will load all the slides in the folder in sequence.

Figure 5-2 Load slides button

Following image shows system after loading some 300 slides. Here current slide is slide-11 and five slides are bookmarked for easy identification.

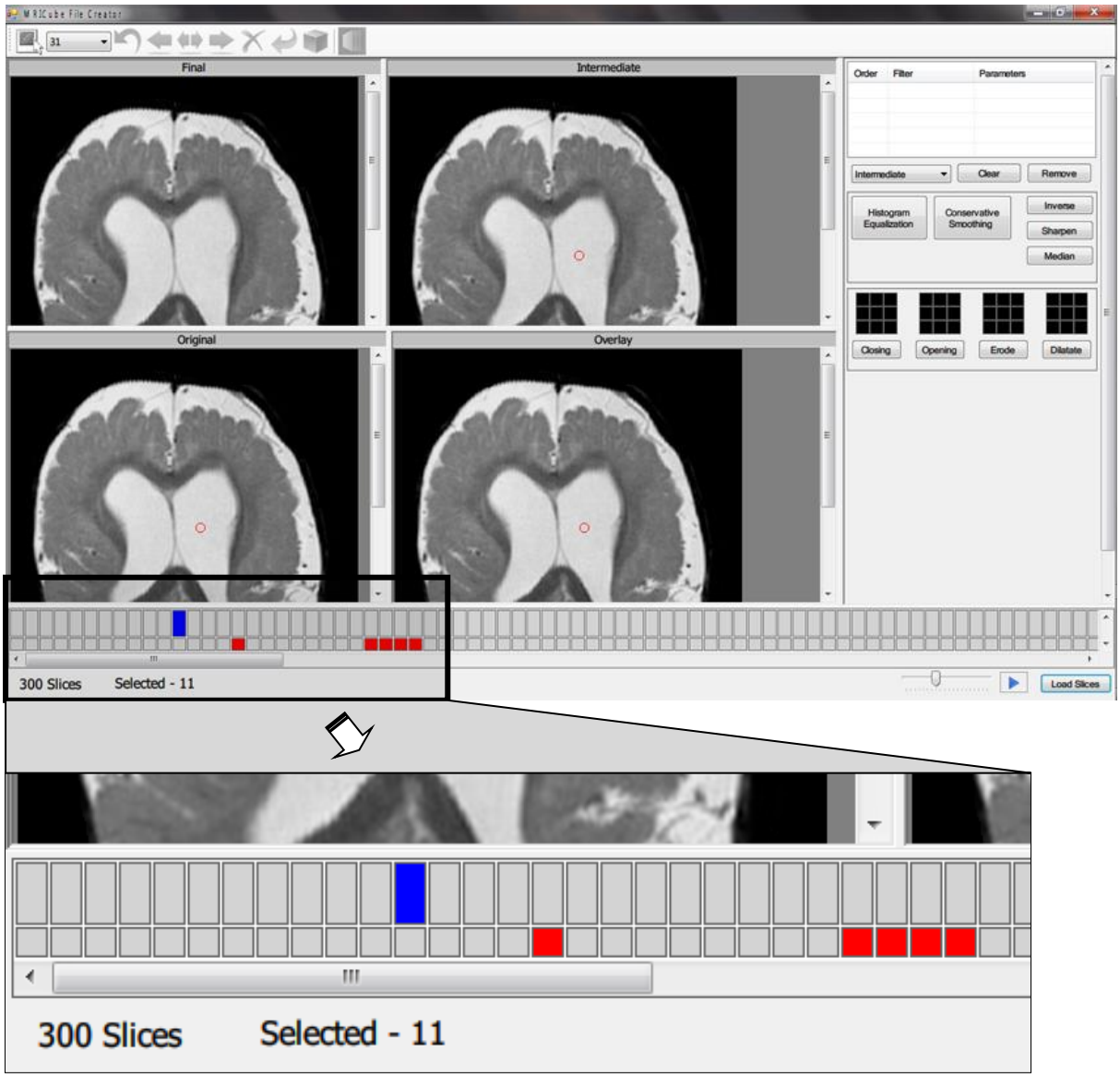

Figure 5-3 After loading slides (bottom is a magnified view)

5.1.2 Processing slides for objects selection

Nine basic image processing method can be applied for slides. Followings are the available nine.

1. Histogram equalization
2. Conservative smoothing

} Built in filter default AForge

3. Invert
4. Sharpen – Fixed value sharpen
5. Median
6. Open
7. Close
8. Dilate
9. Erode

} Can use custom kernel

5.1.3 Example slide processing

Following steps shows how to apply various filters for object extraction.

Step 1

Here some slides are loaded in to the system.

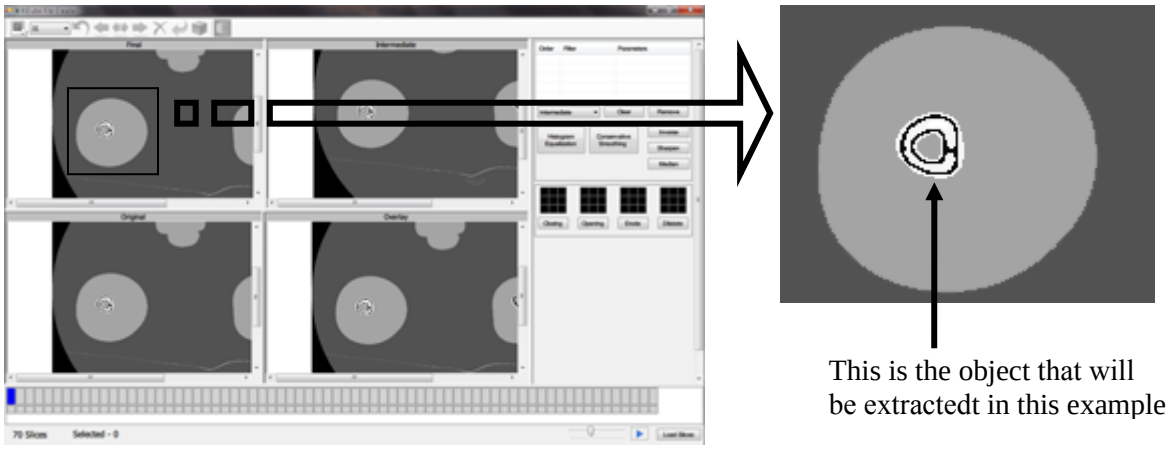

Figure 5-4 Object going to select

Step 2

To apply invert filter, Select intermediate from the (3) applied filter list (Figure 3) area. Then click the Invert button (6) Available filters (Figure 3). When a filter is applied it will be listed on the applied filter list. Following figure will help to understand the process.

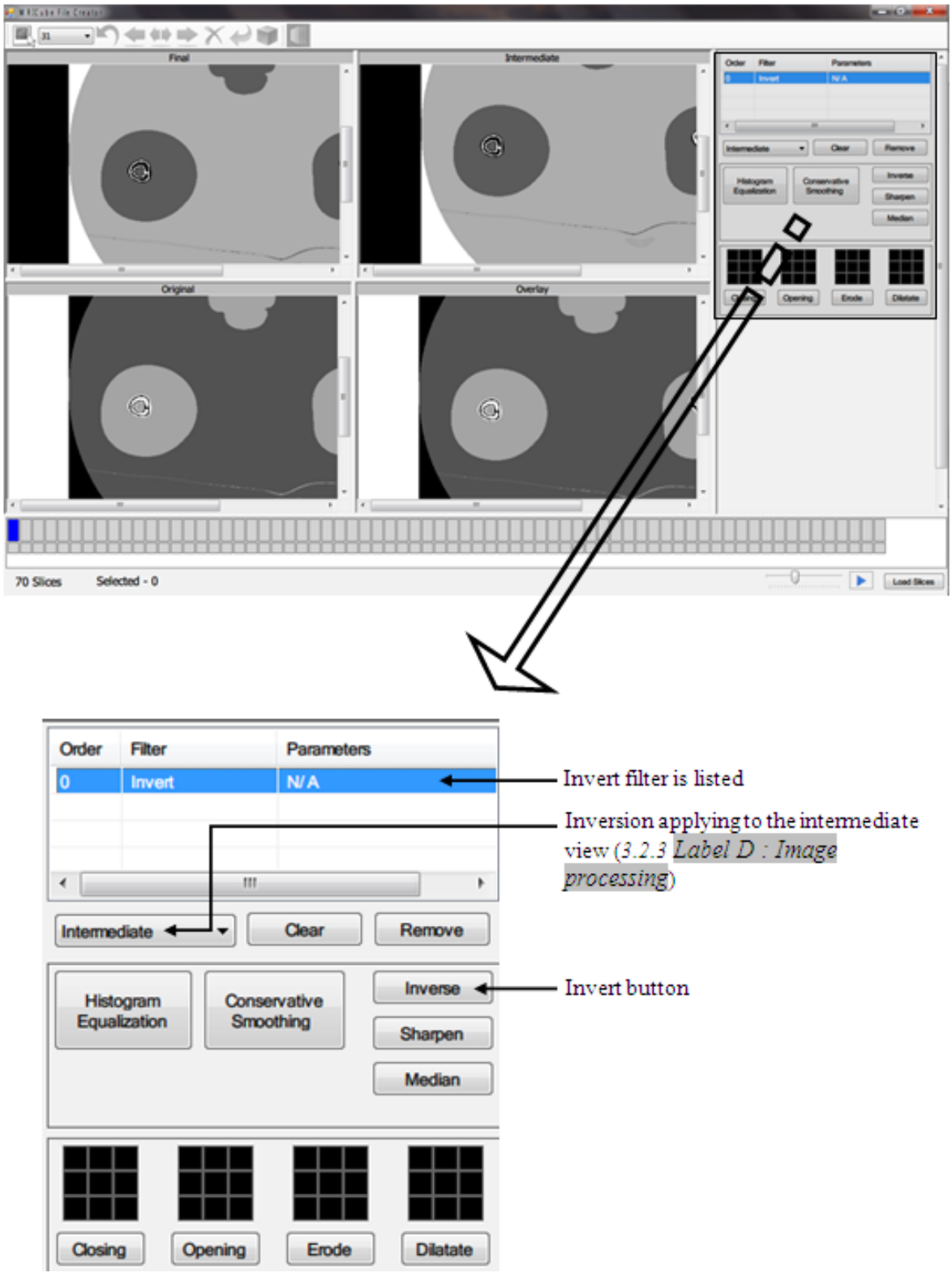

Figure 5-5 Apply invert filter

Step 3

Next performs opening on intermediate view. First create a kernel for the opening operation. Then as intermediate view is already selected from the drop down list, press Open button.

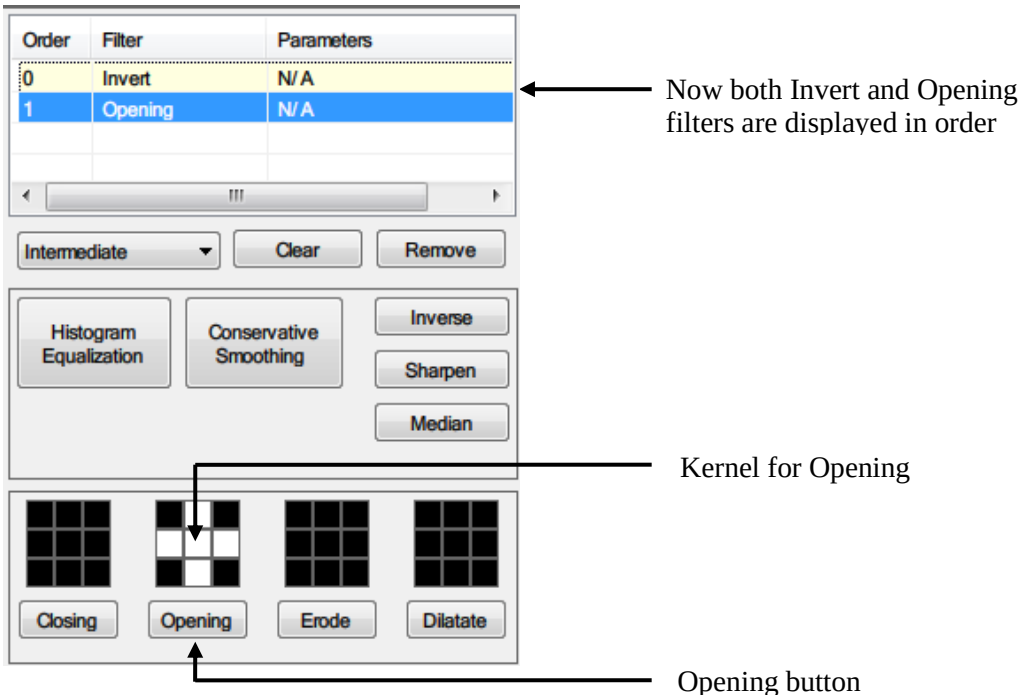

Figure 5-6 Apply Opening

Step 3

Next will perform erode on final view. First create a kernel for the erode operation. Then select final from the drop down list, press Erode button.

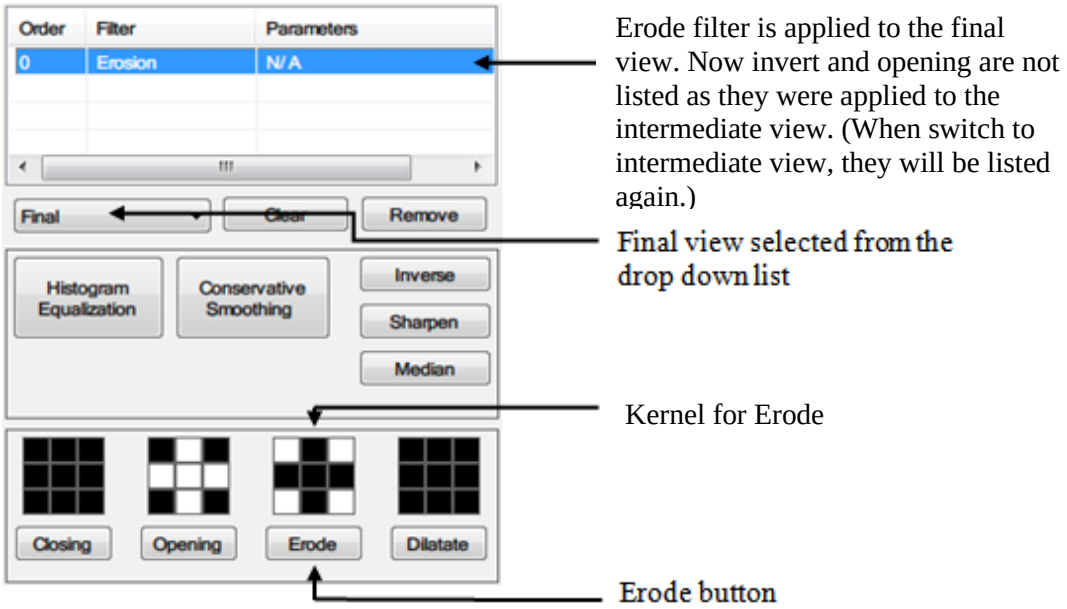

Figure 5-7 Apply Erode

Toolbar buttons description

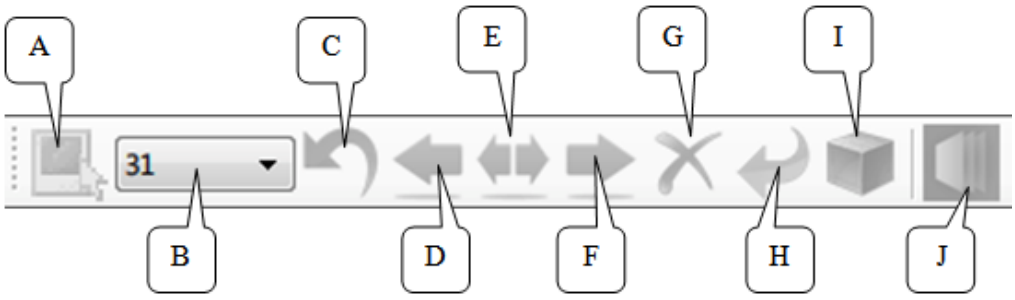

Figure 5-8 Toolbar

A. Clicked before object selection.
B. Threshold (tolerance) value for flood fill, when selecting objects.
C. Undo last object selection.
D. Propagate object backward.
E. Propagate object both ways.
F. Propagate object forward.
G. Cancel last object propagation.
H. Accept last object propagation (After that, can select another object).
I. Save volumetric dataset in a MRIC file.
J. Check slide deviations.

Above labels will be used when mentioning toolbar buttons.

1.3 Selecting an object

To select an object first click button - A and click on the object of final view. If the object is selected successfully, it will be highlighted on Overlay view.

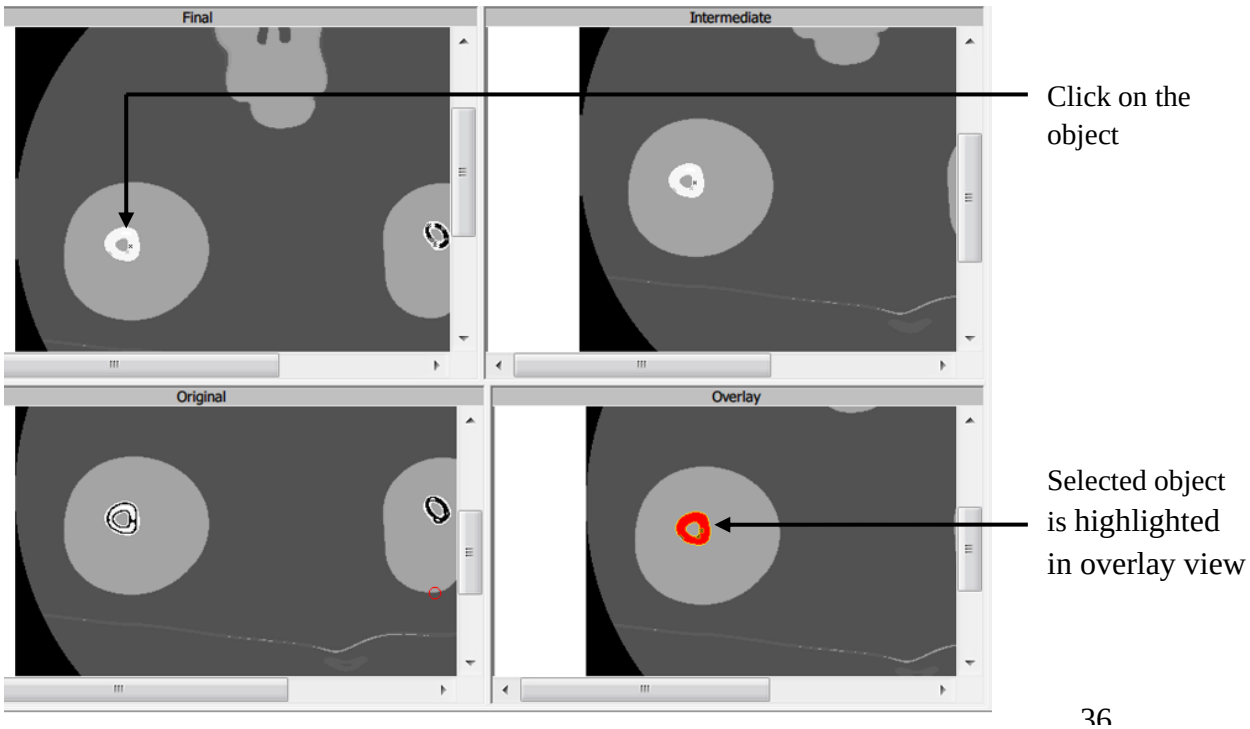

Figure 5-9 Selecting an object

Threshold value for flood fill method can be adjusted by using button-B of the toolbar when selecting objects. If you are not satisfied with the selection undo selection with the button-C.

Propagating a selected object

There are three ways to propagate an object forward, backward or both ways. If the current slide is the first slide, then you can propagate forward only. But if you are in an intermediate slide you can use any method depending on the requirement. Press D, E or F button to propagate the object.

If you made a mistake or the output is not satisfactory, you can undo whole process by clicking button G. Once the object propagated, click button-H to accept it. Then you can either select another object following the same procedure or move on to create volumetric dataset form selected object.

Check for slide deviations

By pressing J button slide deviation dialog box can be open. This dialog box can be used to check individual objects for any uneven slide deviation.

Creating volumetric dataset

Once all the objects are selected and propagated press button-I for create volumetric dataset. When you press this button a dialog box will appear allowing to select a location to save the generated volumetric dataset as a MRIC file.

Open and view MRIC file.

3DView component is used to view volumetric dataset saved in a MRIC file. Following figure will used for explain this component.

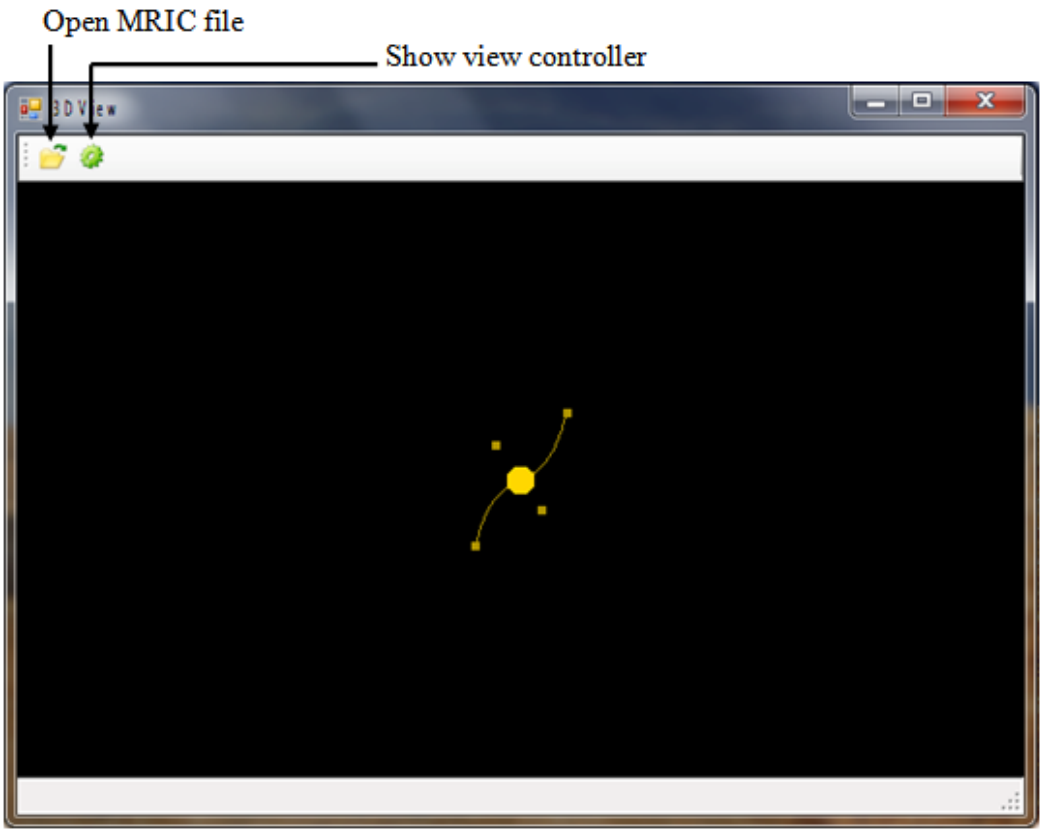

Figure 5-10 3DView component

Clicked *open MRIC file* button and select the MRIC file that you want to view. Then the system will generate and display 3D view.

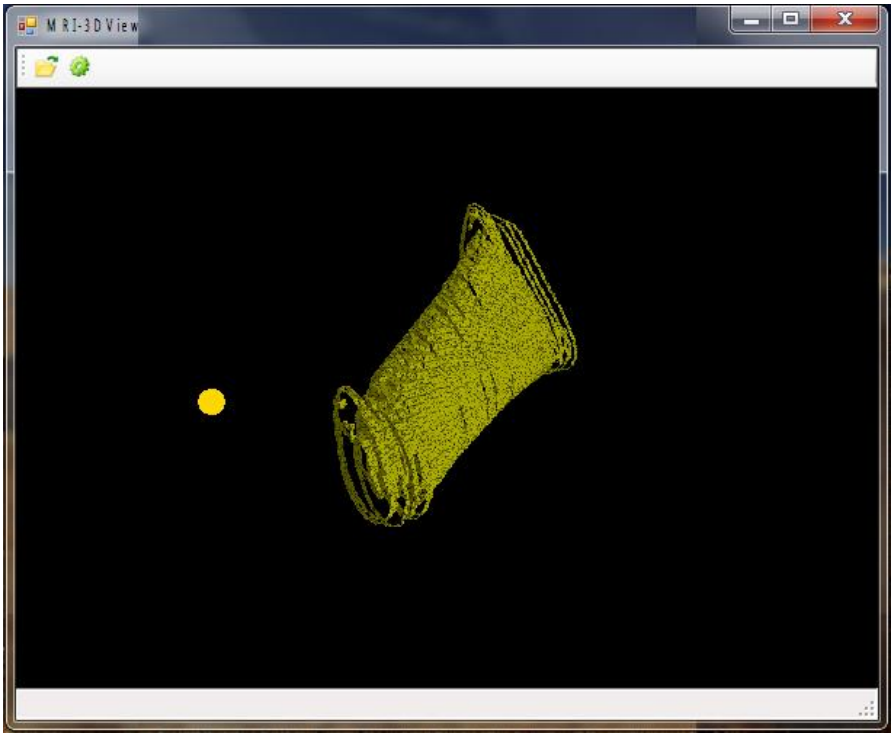

Figure 5-11 3DView after loading a volumetric dataset

5.2 Manipulating 3D view

There are two ways to manipulate 3D view.

1. Using mouse only
2. Using view controller

5.2.1 Using mouse only

1. Drag left mouse button – Rotate X, Y axis.
2. Drag right mouse button – Transform X, Y axis.
3. Rotate mouse wheel – Transform Z axis.

5.2.2 Using view controller

Click *show view controller* button to launch view controller.

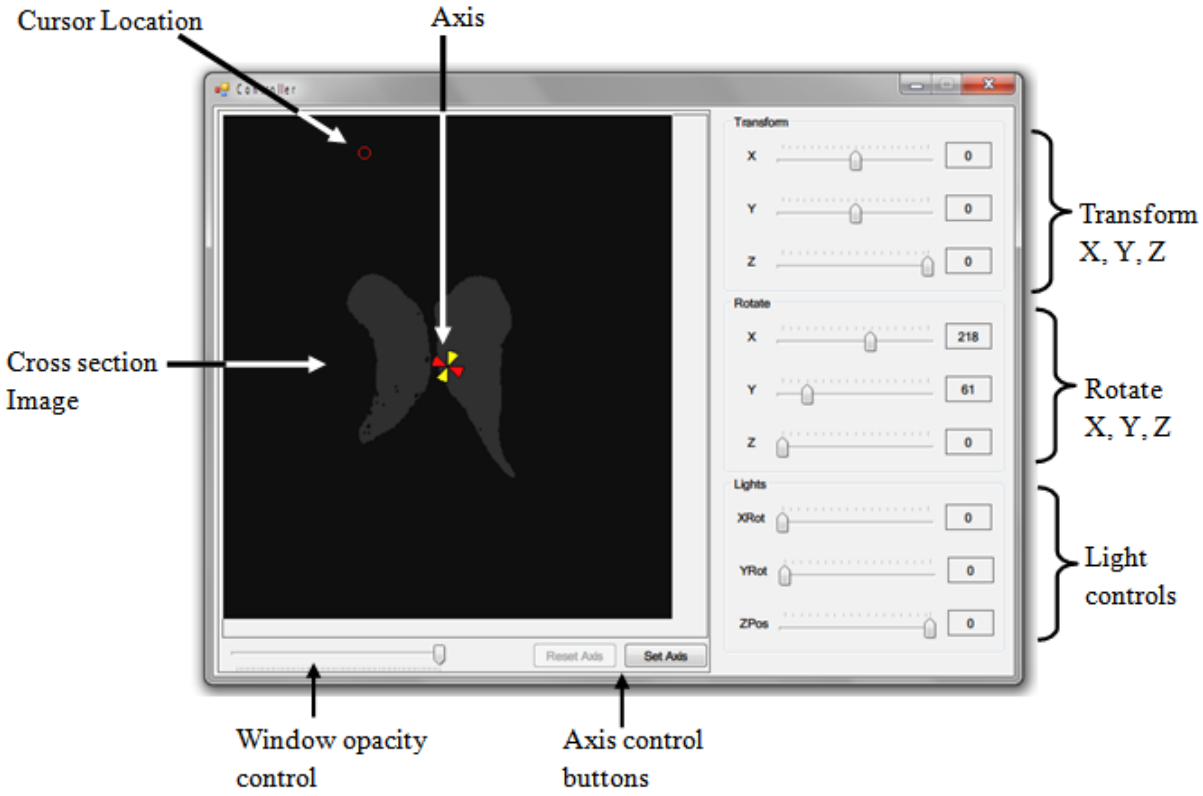

Figure 5-12 View controller

Transform, view rotation and rotating specular light source can be done using controls on the right side of the window.

Window opacity control

Sometimes controller window can cover 3D view window. Reduction opacity of the controller window may help to adjust the 3D view easily. By dragging *window opacity control* slider, you can adjust the opacity.

Cross section image

When creating MRIC file, a middle slide in the slide list will be save as cross section image. View controller loads and displays this cross section image (*Figure 5-12 View controller*) as an aid for manipulating 3D view.

Axis control

When creating a volumetric dataset, user can specify X, Y axis of the volumetric dataset. This is saved in the MRIC file (*3.3 Structure of a MRIC file*). This axis will be used to rotate 3D view. In view controller axis will be displayed in red-yellow color target. By clicking on a deferent location of the cross section image and pressing *Set Axis* button the user can manually change the axis location. This is a temporally change. By pressing Reset axis, axis can be restore to original location saved in MRIC file.

CHAPTER 6

Conclusions and Recommendations

6.1 Conclusions

Although the study area of rendering 2D images to 3D images is vast and full of complex thereories and algorithms, this study only focused on a small area which is visualising MRI or CT similar images in 3D format and all the processing were vastly relating features of Contour connecting algorithm which is a surface fitting algorithm.

Main objective of this study and research was to find out different technologies and techniqes available for volume visualisation so that it will enable to visualise CT , MRI and other medical images to be rendered in 3D format which is eargerly sought by many medical practitioners. Especially in Sri Lankan hospitals they prefer to have such systems to give better services and they cannot afford the systems which are being used in other developed coutnries due to its complexity and high system requirements. So, particularly the study in this research focused on how to make this system with simple techniques, which can be run on low performance generally used hardware without using any special proprietary software . Considering all this facts, the prototype was developed, which satisfies the basic requirements sought.

Although many algorithms and techniques have been presented in this context, each technique has its own merits and demerits but combining the strenghts of those techniques obviously will give more room for development.

The developed prototype only requires basic configuration such as 1.8MHz processing speed, 2GB RAM, and Windows 7 Operating system to fully fuction. The prototype was tested and outputs were analysed to see whether it meets the required objective. As it positivelely responded, it gave more hope to further develop the system to add more functionalities.

6.2 Recommendations

As this study gave indepth knowledge of various techniques in the field, the sytem can be furhter tuned to function by combining the strengths of various algorithems. Mainly due to the time constrains object identification process is optimized up to a basic usable level. By adding more filters and fine tuning existing filters, object identification process can be further improved.

Further, the system does not use any interpolation technique to identify intermediate voxels as it involves rigorous mathematical expressions. By adding a better interpolation mechanism, smoothness of 3D views can be greatly increased.

References

1. Article (Graphic): Lode Vandevenne, http://lodev.org/cgtutor/floodfill.html. (17 December 2011)
2. Book: Richard S. Wright, Jr. Benjamin Lipchak Nicholas Haemel, OpenGL Superbible Fourth Edition, 2007
3. Journal with Survey of Algorithms for Volume Visualization: T. Todd Elvins, http://csee.umbc.edu/~nicholas/601/elvins.pdf. (04 January 2012)
4. Article (Image Processing): Andrew Kirillov, http://www.codeproject.com/ Articles/9727/Image-Processing-Lab-in-C, (14 Mar 2007)
5. Article: Survey of Algorithms for Volume Visualization", T. Todd Elvins, Computer Graphics 26:3, pp. 194-201 (August, 1992)
6. Article: F. Permadi, http://www.permadi.com/tutorial/raycast/index.html (December, 2011)
7. Site: Timo Ropinski, http://www.voreen.org (December, 2011)
8. Sabella, P., “A Rendering Algorithm for Visualizing 3D Scalar Fields”, Computer
9. Graphics (proceedings of SIGGRAPH), 22(4), August 1988, pp. 51–58.
10. Upson, C., and Keeler, M., “V-BUFFER: Visible Volume Rendering”, Computer
11. Graphics (proceedings of SIGGRAPH), 22(4), August 1988, pp. 154–159.
12. Tuy, H. K., and Tuy, L. T., “Direct 2-D Display of 3-D Objects”, IEEE Computer
13. Graphics and Applications, 4(10), November 1984, pp. 29–33.
14. Kajiya, J. T. and Von Herzen, B. P., “Ray Tracing Volume Densities”, Computer
15. Graphics (proceedings of SIGGRAPH), 18(3), July 1984, pp. 165–174.
16. T. Todd Elvins, “A Survey of Algorithms for Volume Visualization”, Advanced Scientific Visualization Laboratory, San Diego Supercomputer Center,1992.
17. Levoy, M., “Display of Surfaces from Volume Data”, IEEE Computer Graphics &
18. Applications, 8(3), May 1988, pp. 29–37.

Appendix - Coding

6.1 Threshold Flood Fill

```
public static int FloodFill(FloodFillInfo ffi)
{
	ByteImage bi = ffi.m_bi;
	MySize s = bi.Size;
	MyPoint p = ffi.m_start_point;
	byte[,] pixels = bi.GetPixels();

	if (!IsInRange(p, ffi, pixels))
		return 0;//Can't fill

	byte[,] objmask = bi.GetObjectMask();
	Queue<MyPoint> q = new Queue<MyPoint>(200);
	MyPoint top, right, bottom, left;
	int len = 0;
	s.Width--;

	q.Enqueue(p);

	while (q.Count != 0)
	{
		p = q.Dequeue();
		//if (bi.GetPix(p.x, p.y) != tar_col)
		if (!IsInRange(p, ffi, pixels))
			continue;

		right = p;
		//scann right
		do
		{
			//bi.SetPix(right.x, right.y, fill_col);
			pixels[right.x, right.y] = ffi.m_fillcol;
			objmask[right.x, right.y] = ffi.m_obj_fill;
			len++;

			//add top
			top = right; top.y--;
			if (top.y >= 0)
			{
				//if (bi.GetPix(top.x, top.y) == tar_col)
				if (IsInRange(top, ffi, pixels))
					q.Enqueue(top);
			}

			//add bottom
			bottom = right; bottom.y++;
			if (bottom.y < s.Height)
			{
				//if (bi.GetPix(bottom.x, bottom.y) == tar_col)
				if (IsInRange(bottom, ffi, pixels))
					q.Enqueue(bottom);
			}

			if (right.x == s.Width)
				break;
			right.x++;
			//} while (bi.GetPix(right.x, right.y) == tar_col);
		} while (IsInRange(right, ffi, pixels));
```

```
                //scann left
                left = p; left.x--;
                if (left.x >= 0)
                {
                        //while (bi.GetPix(left.x, left.y) == tar_col)
                        while (IsInRange(left, ffi, pixels))
                        {
                                //bi.SetPix(left.x, left.y, fill_col);
                                pixels[left.x, left.y] = ffi.m_fillcol;
                                objmask[left.x, left.y] = ffi.m_obj_fill;
                                len++;

                                //add top
                                top = left; top.y--;
                                if (top.y >= 0)
                                {
                                        //if (bi.GetPix(top.x, top.y) == tar_col)
                                        if (IsInRange(top, ffi, pixels))
                                                q.Enqueue(top);
                                }

                                //add bottom
                                bottom = left; bottom.y++;
                                if (bottom.y < s.Height)
                                {
                                        //if (bi.GetPix(bottom.x, bottom.y) ==
tar_col)
                                        if (IsInRange(bottom, ffi, pixels))
                                                q.Enqueue(bottom);
                                }

                                left.x--;
                                if (left.x < 0)
                                        break;
                        }
                }
        }

        if (len > 5)
        {
                MRIObject obj = bi.GetObjectByFill(ffi.m_obj_fill);
                if (obj==null)
                {
                        //No object with this color
                        bi.AddObject(len, ffi.m_obj_fill);
                } else{
                        //There is a object with this color so increase area of
existing object
                        obj.IncreaseArea(len);
                }

                MarkBoarders(objmask, ffi);
                return len;
        }

        return 0;
}
```

6.2 Rendering volumetric dataset

```
public int RenderDataCube(Point ap,int scale)
{
	DataSlide sl;
	MRIObject obj;
	Branch bra;
	MyVertex v1 = null;
	MyVertex v2 = null;
	MyVertex v3 = null;

	int nbra, nobj;
	float fscale = 100.0f / scale;
	int len, ns = m_Slides.Count;

	//Z scale
	float zstep = 1.0f / fscale;
	float z = (ns / 2) * zstep * (-1);
	int dls = GL.GenLists(1);
	GL.NewList(dls, ListMode.Compile);
	GL.Color3(Color.Gray);

	MyTexture tx = MyTexture.LoadTexture(@"Images/t1.png");
	tx.MakeCurrent();

	for (int i = 0; i < ns; i++)
	{//************* Go through all the slides **********
		sl = m_Slides[i];
		nobj = sl.ObjectCount;

		for (int j = 0; j < nobj; j++)
		{//Go through objects ************
			obj = sl.GetObject(j);
			nbra = obj.BranchCount;
			Console.WriteLine(nbra);
			for (int k = 0; k < nbra; k++)
			{//Go through branches ************
				bra = obj.GetBranch(k);
				len = bra.VertexCount;
				//Console.WriteLine(len);
				GL.Begin(BeginMode.Triangles);
				for (int a = 0; a < len-1; a++)
				{//Go through vertexes ************
					v1 = bra.GetVertex( a, ap, fscale,z);
					v2 = new MyVertex(v1.X, v1.Y,  z + zstep);
					v3 = bra.GetVertex(a+1, ap, fscale,z);

					MyVertex.DrawTriangle(v1, v2, v3,true);
					v1 = new MyVertex(v3.X, v3.Y, v3.Z + zstep);
					MyVertex.DrawTriangle(v3, v2, v1, true);
				}
				GL.End();
			}
		}
		z += zstep;
	}//************* Go through all the slides **********

	GL.EndList();
	return dls;
}
```

6.3 Saving volumetric dataset as a MRIC file

```
public void Save(string savepath, int from, int to)
{
	int p;
	MemoryStream ms = new MemoryStream();
	BinaryWriter mbw = new BinaryWriter(ms);
	FileStream fs = new FileStream(savepath, FileMode.Create);
	BinaryWriter fbw = new BinaryWriter(fs);

	//Write signature VAJIRA-MRICUBE-V6
	byte[] sng = {86,65,74,73,82,65,45,77,82,73,67,85,66,69,45,86,56};
	mbw.Write(sng);
	//Write slides dimension
	mbw.Write(m_size.Width);
	mbw.Write(m_size.Height);
	//Write no of slides
	to++;
	int i = to - from;
	mbw.Write(i);
	//Write axis point
	Point ap = m_r.m_pix_con.PicOver.AxisPoint;
	mbw.Write(ap.X); mbw.Write(ap.Y);
	//Write midle crossection image ********************
	i = i / 2;
	ByteImage bi = m_image_list[0];
	Byte[,] arr = bi.GetObjectMask();

	for (i = 0; i < m_size.Height; i++)
	{
		for (int j = 0; j < m_size.Width; j++)
		{
			mbw.Write(arr[j, i]);
		}
	}
	arr = null;
	//Write midle crossection image ********************
	int c = 0;
	int blen;
	MyPoint po;
	Stack<MyPoint> st = new Stack<MyPoint>(100);//Give stack for ByteImage
(Efficient than one for each ByteImage)
	//CreateBoaders and Write objects for individual slide
	for (i = from; i < to; i++)
	{
		bi = m_image_list[i];
		bi.CreateBoaders(st);
		mbw.Write(bi.ObjectCount);//Write no of Objects(Byte)
		for (int k = 0; k < bi.ObjectCount; k++)
		{
			MRIObject obj = bi.GetObjectByIndex(k);
			mbw.Write(obj.BranchCount);//Write no of Branches
			for (int j = 0; j < obj.BranchCount; j++)
			{
				Branch bra = obj.GetBranch(j);
				blen = bra.PointCount;
				mbw.Write(blen);
				//bra.Revers();
				for (int m = 0; m < blen; m++)
				{
					po = bra.PopPoint(m);
					mbw.Write((short)po.x);
```

```
                                mbw.Write((short)po.y);
                                mbw.Write((byte)255);
                        }
                    }
                }

                if (c++==10)
                {//Write 10 slides at a time
                    p = (int)ms.Position;//must get before closing ms
                    mbw.Close();
                    fbw.Write(ms.GetBuffer(), 0, p);// ms.GetBuffer().Length
!= p
                    ms.Close();
                    ms = null;
                    c = 0;
                    if (i < to)
                    {//Still there are remaining slides so need ms
                        ms = new MemoryStream();
                        mbw = new BinaryWriter(ms);
                    }
                }
            }

            if (ms!=null)
            {//ms contains data so write them to HD
                p = (int)ms.Position;//must get before closing ms
                mbw.Close();
                fbw.Write(ms.GetBuffer(), 0, p);
                ms.Close();
                ms = null;
            }
            fbw.Close();
            fs.Close();
            fs = null;
            IP.GCCollect();
}
```