Interactive Virtual Endoscopy of Upper GI Tract and Stomach

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Engineering

IN

COMPUTER SCIENCE AND ENGINEERING

by

Rahul Sharma

Computer Science and Automation

Indian Institute of Science

BANGALORE – 560 012

JULY 2013

TO

My Parents

Acknowledgements

I thank Prof. Vijay Natarajan for his guidance and inspiration for this work. I also thank the reader Dr. Uday Kumar Reddy B, for his thorough and careful review of this work and for valuable suggestions. Apart from them, I was able to get much insight into the problems by the discussions I had with Chayan Ghosh and Ankit Dixit. My sincere thanks to them. I thank my lab mates Rakesh Malviya, Talha Bin Masood and Dilip Thomas for their support.

I thank my father Rakesh, mother Geeta and sister Anjali for teaching me how to live life. I thank my family and friends for all their support.

Abstract

We have developed an interactive virtual endoscopy system for the training of doctors to hone their skills. In this document, we describe the extraction of the model of the upper GI tract and stomach, and the calculation of automated walk-through for navigating through the GI tract. We also present the functionality of mapping arbitrary texture to the 3D model as a viewing upgrade.

Contents

Acknowledgements **i**

Abstract **ii**

Keywords **vii**

Notation and Abbreviations **viii**

1 Introduction **1**
1.1 Virtual Endoscopy 1
1.2 Problems in Virtual Endoscopy 2

2 Related Work **3**
2.1 Virtual Endoscopy 3
2.2 Surface parameterization for texture mapping 4

3 BACKGROUND **5**
3.1 Visible Human Data 5
3.2 ITK-SNAP 6
3.3 B-Spline Curve 7
3.4 Texture Mapping 7

4 ALGORITHM **9**
4.1 Automated walk-through 9
4.2 Flattening 12
4.3 Texture mapping 13

5 IMPLEMENTATION **17**
5.1 Extraction of Model 17
5.2 Software Design and user Interface 17

6 EVALUATION AND RESULTS **19**

7 Conclusions **22**

A Framework **23**

B Headers **25**
B.1 cmath.h . . . 25
B.1.1 Class Vector2f . . . 25
B.1.2 Class Vector3f . . . 26
B.1.3 Class Vector4f . . . 27
B.2 curves.h . . . 28
B.2.1 Type Definitions . . . 28
B.2.2 Structures . . . 28
B.2.3 Class BSpline . . . 28
B.3 trackball.h . . . 30
B.3.1 Class TrackBall . . . 30
B.4 3deng.h . . . 31
B.4.1 Constants . . . 31
B.4.2 Structures . . . 31
B.4.3 Hierarchy of 3D Object Classes . . . 31
B.4.4 Class Camera3d . . . 32
B.4.5 Class Material3d . . . 32
B.4.6 Class Light3d . . . 33
B.4.7 Class Object3d . . . 34
B.4.8 Class Obj3ds . . . 35
B.4.9 Class ObjBIN . . . 36
B.4.10 Class Obj3dGraph . . . 37
B.4.11 Methods . . . 38

References **40**

List of Tables

6.1 Running time of $generateGraphFromMesh$ and $mappingTexture$. 20

B.1 Methods of Vector2f 25
B.2 Methods of Vector3f 26
B.3 Methods of Vector4f 27
B.4 Data members of BSpline 28
B.5 Methods of BSpline 29
B.6 Methods of TrackBall 30
B.7 Data members of Camera3d 32
B.8 Data members of Material3d 32
B.9 Data members of Light3d 33
B.10 Data members of Object3d 34
B.11 Methods of Object3d 35
B.12 Data members of Obj3ds 35
B.13 Methods of Obj3ds 36
B.14 ObjBIN File Format Specifications 36
B.15 Data members of ObjBIN 37
B.16 Methods of ObjBIN 37
B.17 Methods of Obj3dGraph 38

List of Figures

3.1 Visible Human Project - section through the abdomen of a human male . . . 5
3.2 Showing the process of calculating contour . . . 6
3.3 Showing the process of texture mapping . . . 8

4.1 Showing the process of calculating the path for automated walk-through . . . 11
4.2 Rotation of uncommon vertex of neighbourhood triangle along the shared edge by an amount of the angle between their normals. . . . 12
4.3 (a) An stochastic texture with random texture patches, (b) texture patches arranged without orientation and (c) unoriented texture patches without boundaries. 14
4.4 A triangle mesh (solid line) and its dual graph (dotted lines). . . . 15
4.5 Showing the difference between choosing random seeds and breadth-first seeds 16

5.1 Snapshot of the tool showing the extracted model . . . 18

6.1 Snapshot of the tool showing model view . . . 20
6.2 Snapshot of the tool showing interior of the extracted model . . . 21

A.1 Abstraction of the tool . . . 23
A.2 Internal working of 3D engine . . . 24

B.1 Derivative classes of Object3d . . . 31

Keywords

Virtual Endoscopy

Notation and Abbreviations

No notation is used in this document. No abbreviations have been used either.

Chapter 1

Introduction

1.1 Virtual Endoscopy

Endoscopy is used to view inside the body, basically interiors of the hollow organs. In the other medical imaging device doctors would not probe the device directly into the organ. Besides viewing, they can also control the position and angle of probe. In some situations, they also require an additional channel to inject or to take a sample. This process also includes a risk of infection and over-sedation [22]. Young doctors use a virtual endoscopy system to hone their skill by practicing. It has been shown that a virtual reality endoscopy simulator can distinguish between beginners and experts in endoscopy and assess whether training improves the performance of beginners [8].

Some work has been done in the area of virtual colonoscopy [12] to provide the training software for the virtual endoscopy. Virtual endoscopy can also be used to access special parts of human body such as blood vessels which are impossible to access with real endoscopy.

Jolesz [13] used MRI and CT scans to extract model of GI tract and other organs along with a viewer. CT scan and MRI scan are used to get the cross section of the inner organs but the resolution is only 512 by 512 pixels per slice which is not sufficient to extract good models of organs. Instead of CT scan and MRI scan we used RGB image version of Visible Human Data from National Library of Medicine [1], where each slice has a resolution of 2048 by 1216 pixels.

1.2 Problems in Virtual Endoscopy

We identified that exiting systems does not have a good quality 3D model of GI tract and stomach as they use MRI scan and CT scan image, which has their limitation. Our goal is to obtain a good quality 3D model of the GI tract and build an interactive tool that support good quality output of the 3D model. For interactivity we need a fly-through and an automated walkthrough mechanism to navigate inside the model.

The general problems we face while building our virtual endoscopy system are as follows: extraction of good quality model of upper GI tract and stomach, computing automated walk-through, optimizing the rendering process, mapping textures [25] that highlights the surface of the model.

Chapter 2

Related Work

2.1 Virtual Endoscopy

A lot of work has been done in the field of virtual endoscopy. Hong [11] designs an endoscopy system which was used for colonoscopy. It uses a modified region growing algorithm and marching cubes as segmentation techniques.

Yagel [28] describes a system for the training of surgeons in virtual sinus surgery and for teaching in virtual endoscopy. It uses volume rendering and 3D texture mapping [20].

Robb [21] presents two different endoscopy systems, one uses direct volume rendering and the other uses surface rendering. Darabi [4] proposed virtual endoscopy system that uses volume rendering with ray casting and planned navigation.

Ikuta [12] highlights the significance of virtual endoscopy with force sensation. Jolesz [13] developed a virtual endoscopy based on the CT and MRI data, with fly-through method to view and explore the data. The data used by Jolesz [13] has a resolution of .25 Megapixels per slice.

Hong [10] proposed the method for 3D virtual colonoscopy. Lakare [15] proposed a segmentation process for 3D virtual colonoscopy and You [30] proposed a method for interactive volumetric rendering of 3D virtual colonoscopy.

2.2 Surface parameterization for texture mapping

Initial work in this field has been carried out by Bennis [2]. They uses a series of user-defined patches and then optimises them. It proposed a new technique for interactive piecewise flattening of parametric 3D surfaces. It selects a chord curve on the surface and then unfold (flatten) the surface around this curve. However, this method compromise between discontinuities and distortions.

Pedersen [16, 17] proposed an interactive texture mapping of 3D surfaces. He used cut-and-paste method to map a surface. However his method require inputs form user to fully map the surface. Praun [19] uses similar method to parameterize the surface but it is fully automated. It describes the method for creating texture over an arbitrary surface mesh using an example 2D texture. We implemented this method because it does not requires user input. It repeatedly paste the texture patches on the surface patches until the whole mesh is completely covered. A surface patch is small area on the surface of the 3D model suitable to map a single texture patch. It select surface patches randomly. After covering the mesh it uses an energy minimising equation to optimise the alignment and local orientation of the mapped texture.

Chapter 3

BACKGROUND

A good model of upper GI tract and Stomach is required for a good visualization. In order to extract organs we used Visible Human Data. For the automated walkthrough we computed centreline of tract and fit a spline curve for smooth walkthrough.

3.1 Visible Human Data

Visible Human Data [1] is an effort to provide visualization applications a good and high resolution data-set of cross sectional images of the human body of a male and a female. To generate the visible male data, cadaver of the male was cut in the axial plane into slices at 1 millimetre interval. This results in 1,871 images, the size of each image is 2.375 megapixels.

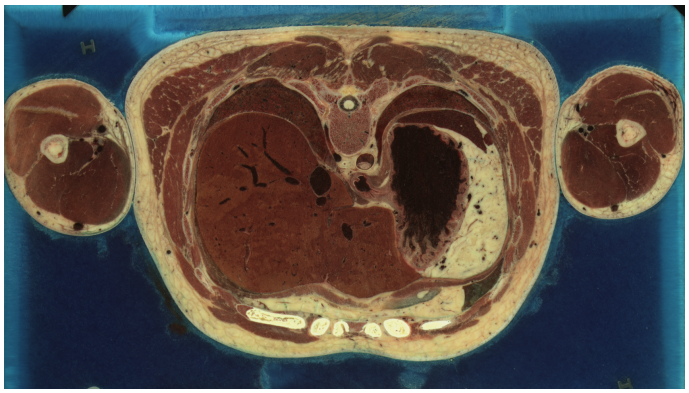
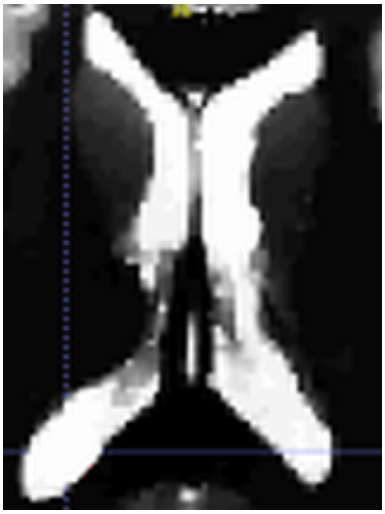


Figure 3.1: Visible Human Project - section through the abdomen of a human male

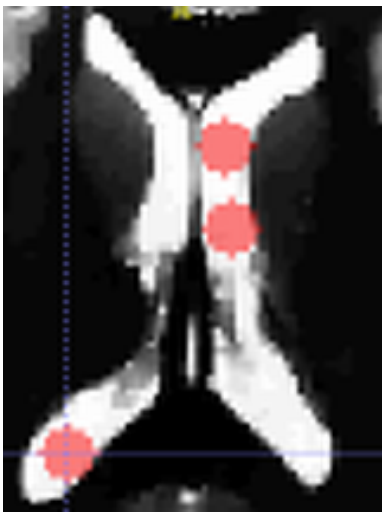
3.2 ITK-SNAP

ITK-SNAP [31] is an open-source tool used for level set active contour segmentation. First it computes the contour of the 3D data using 3D geodesic active contour method (fig-3.2a and fig-3.2b). After that user has to mark the initial region of the snake also called snake tale (fig-3.2c). Then it uses a region growing snake method (fig-3.2d) to construct the 3D triangle mesh from the input data.

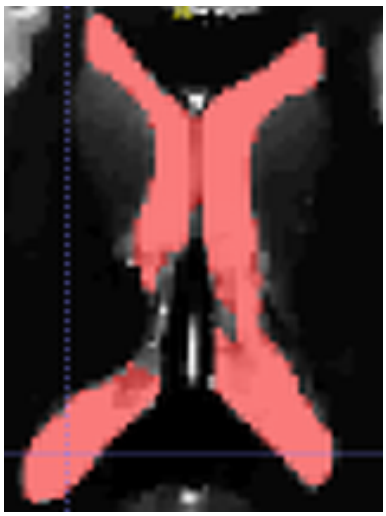


(a) Original image

(b) Contour of fig-3.2a



(c) Initial snake tale



(d) Final snake region

Figure 3.2: Showing the process of calculating contour

3.3 B-Spline Curve

A spline curve is a single continuous curve which is formed by connecting sequence of curve segments. A B-Spline [29] is a generalisation of Bézier curve [29]. We used B-Spline curve to compute the path for automated walkthrough for the GI tract. We used B-Spline curve because most shapes are too complicated to use a single Bézier curve as it can avoid the Runge phenomenon [5] without increasing the degree of B-spline [7].

Given $h+1$ control points $P_0, P_1, ..., P_h$ and a knot vector $U = u_0, u_1, ..., u_m$, B-spline curve [23] of degree p and knot vector U is defined as

$$C(u) = \sum_{i=0}^{h} N_{i,p}(u) P_i$$

,where $N_{i,p}(u)$ is the basis function of degree p defined as:

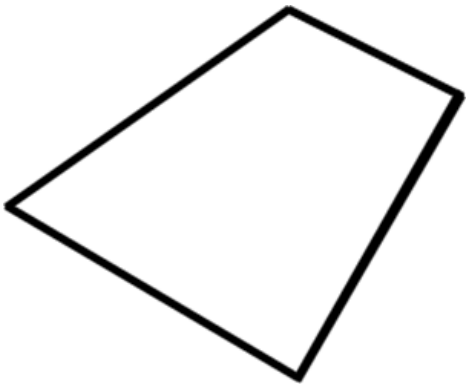
$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u \leq u_{i+1} \\ 0 & \text{if } u < u_i \text{ or } u_{i+1} < u \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

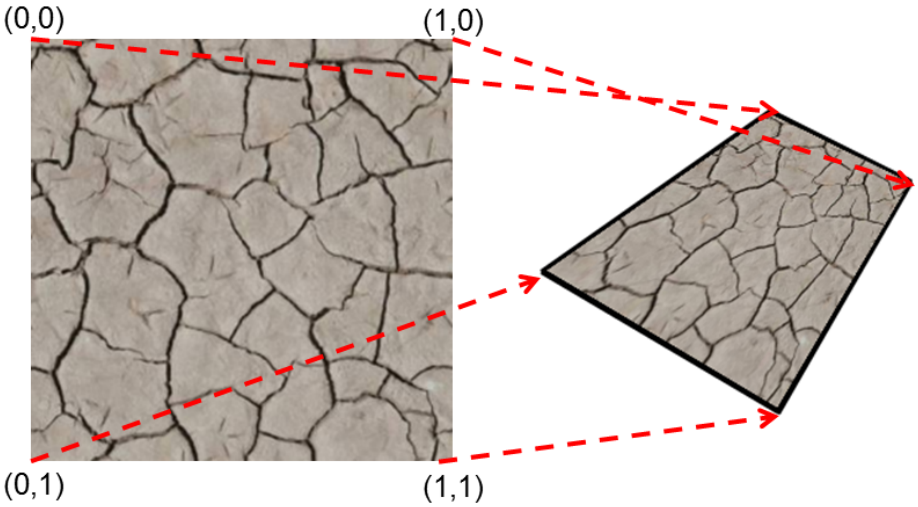
3.4 Texture Mapping

In Computer Graphics, texture mapping is used to enhance the richness of computer generated images [3, 9]. A 2D texture is a 2D image which is mapped to the surface of a 3D object. In OpenGL [26], texture mapping is done by assigning a location within the given texture to each vertex of the 3D object. Fig-3.3 shows the process of texture mapping.

(a) A 2D texture



(b) A quadrilateral



(c) Mapped quadrilateral showing mapped parameters

Figure 3.3: Showing the process of texture mapping

Chapter 4

ALGORITHM

4.1 Automated walk-through

We also provide automated walk-through of the interior of upper GI tract from the extracted model. This path is in the form of B-spline curve. The path extraction process contains three steps:

In the first step, we intersect planes $L_1, L_2, ..., L_r$ with the triangle representation of the mesh by aligning the mesh normal to the planes. For each plane L_i we calculate the set points $L_i = s_1, s_2, ..., s_t$ of intersection of the plane with respect to the mesh (shown in fig-4.1b).

In the second step, we calculate the centroid of each group of points i.e. L_i. This is done by taking the average of all the points in the set L_i (shown in fig-4.1c).

In the final step, we fit a B-spline curve using centroids as the data points for the curve (shown in fig-4.1d).

Let $D_0, D_1, ..., D_h$ be $n + 1$ data points given and we wish to approximately fit a B-spline curve of $h + 1$ control points and degree p, as

$$C(u) = \sum_{i=0}^{h} N_{i,p}(u) P_i$$

We want the curve to pass through the first and the last data points i.e. $D_0 = C(0)$ and $D_n = C(n)$. Now the equation becomes

$$C(u) = N_{0,p}(u) D_0 + \sum_{i=1}^{h-1} N_{i,p}(u) P_i + N_{h,p}(u) D_n$$

Let parameter t_i correspond to data point D_i in the approximation of the curve. For a least square approximation, the error function $f()$ has to be minimized, where

$$f(P_0, P_1, ..., P_h) = \sum_{i=0}^{n-1} |D_i - C(t_i)|$$

This can be solved as a system of linear equations [23].

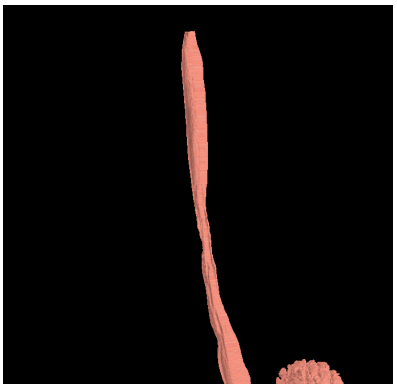
$$(N^T N)P = Q$$

Where, $P = [P_0\ P_1\ ...\ P_h]^T$ and

$$Q = \begin{pmatrix} \sum_{i=1}^{n-1} N_{1,p}(t_i)Q_i \\ \sum_{i=1}^{n-1} N_{2,p}(t_i)Q_i \\ \vdots \\ \sum_{i=1}^{n-1} N_{h-1,p}(t_i)Q_i \end{pmatrix}$$

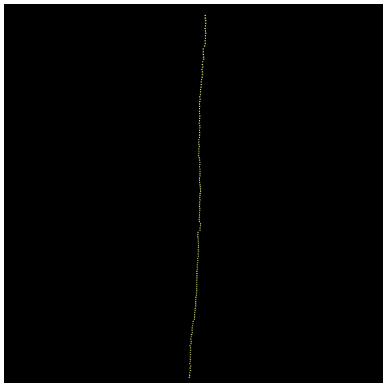
$Q_i = D_i - N_{0,p}(t_i)D_0 - N_{h,p}(t_i)D_n$ and

$$N_{n-1,h-1} = \begin{pmatrix} N_{1,p}(t_1) & N_{2,p}(t_1) & \cdots & N_{h-1,p}(t_1) \\ N_{1,p}(t_2) & N_{2,p}(t_2) & \cdots & N_{h-1,p}(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ N_{1,p}(t_{n-1}) & N_{2,p}(t_{n-1}) & \cdots & N_{h-1,p}(t_{n-1}) \end{pmatrix}$$



(a) Snapshot of the tool showing extracted model

(b) Lines showing the intersection of the planes with the extracted model



(c) Points showing the centroid of the points in the intersection of the planes with the extracted model

(d) Approximated curve along with the extracted model of upper GI tract

Figure 4.1: Showing the process of calculating the path for automated walk-through

4.2 Flattening

In the process of texture mapping we need to assign a 2D texture coordinate to each of the vertices. It is an easy task if all the vertices of the triangle mesh lies in the same plane. But in general, a 3D triangle mesh can have vetices in different planes. For that purpose we took a small neighbourhood of triangles around a selected triangle called seed. After that, we apply tranformation to all vertices in the neighbourhood of triangles, except for the vertices of seed triangle, such that they all lies in the same plane. This process is called flattening. Figure-4.2 shows the flattening of a single triangle. We can easily repeat this process in bredth-first order to flatten a region containig many triagles.

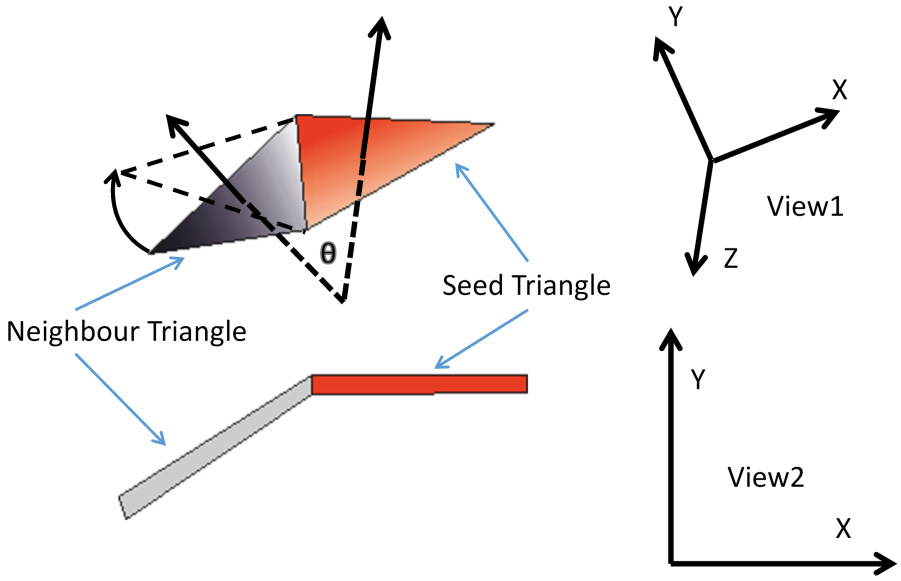


Figure 4.2: Rotation of uncommon vertex of neighbourhood triangle along the shared edge by an amount of the angle between their normals.

The rotation matrix, for rotation by an angle θ about an axis in the direction of a 3D unit vector $u = (u_x, u_y, u_z)$ is given as

$$R = \begin{pmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) - u_y\sin\theta \\ u_y u_x(1-\cos\theta) - u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) - u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{pmatrix}$$

4.3 Texture mapping

We implemented Praun's [19] method for texture mapping. This method works on repeatedly pasting texture patches on to the surface until the whole mesh is covered. The algorithm $mappingTexture$ [Algorithm 2] is used to assign texture co-ordinates to a given 3D triangle mesh. It uses random seeds in each iteration. After the mapping it uses an energy minimization equation to align the texture and to find the local orientation of texture, and decide the boundaries of patches for good output. Since we are using a stochastic texture [6, 18] for the model we don't need to align the texture. It is clear from (fig-4.3) that orientation does not matter when we use stochastic texture. But if we do not use energy minimization function then the output is not aligned. Figure 4.5b shows the texture we used in mapping of the dummy model (fig-4.5a) and the output of random mapping is shown in fig-4.5c. So, we used breadth-first search around the initial seed to find the nearby patches and do the process of mapping in a non-random fashion (fig-4.5d). To use breadth-first search we need to calculate the dual of the mesh (fig-4.4) which is given as $generateGraphFromMesh$ [Algorithm 1].

Algorithm 1 generateGraphFromMesh

Input 3D triangle Mesh M

Output Graph G

1: For each edge (u, v) that belong to triangle t in M make triplet $< u, v, t >$

2: Sort triplets in increasing order of u, v and t

3: Make an empty graph G in which number of vertices is equal to number of triangles in M

4: For each pair of triplet $< u, v, t_1 >$ and $< u, v, t_2 >$ make an edge between t_1 and t_2 in G

5: **return** G

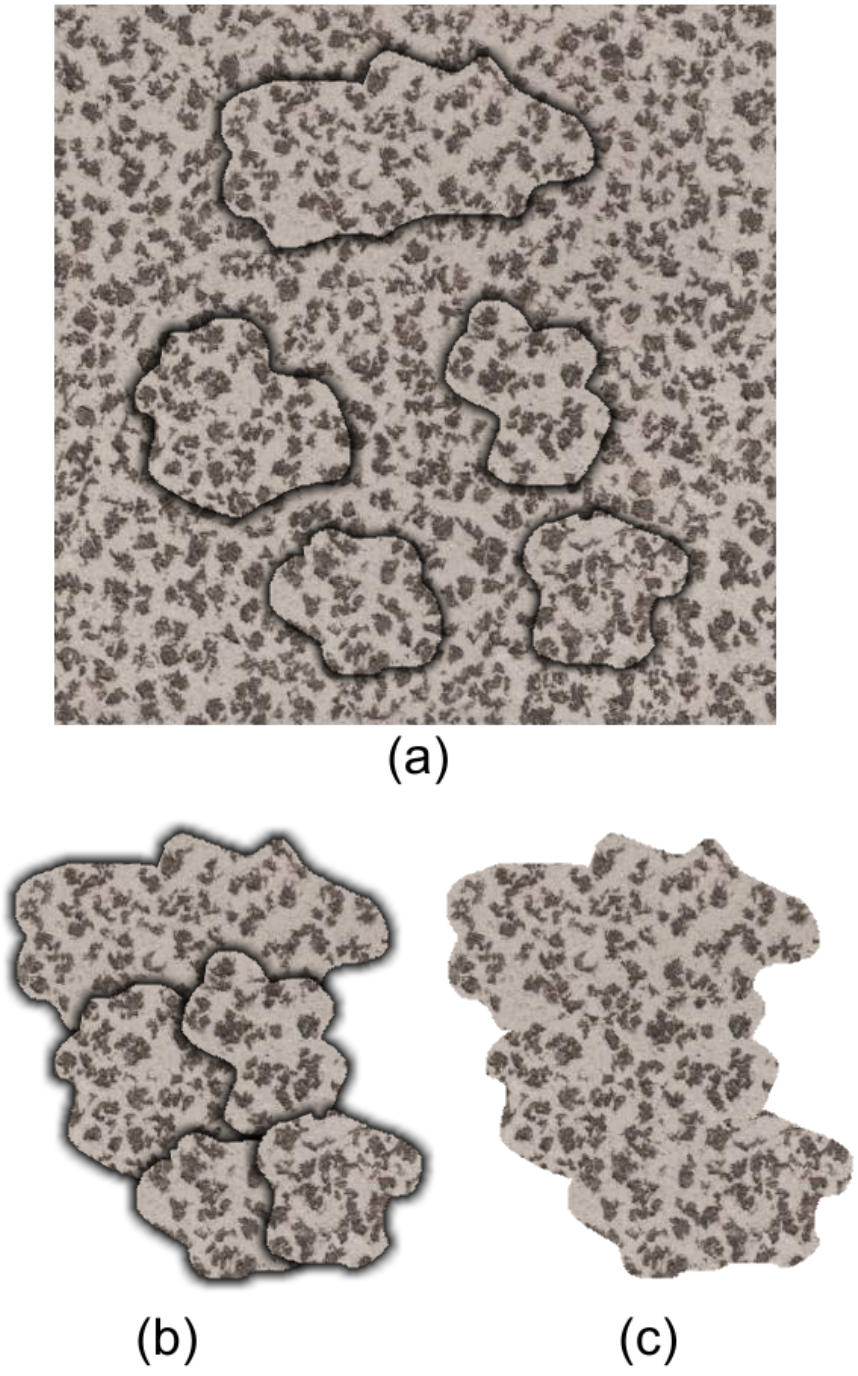


Figure 4.3: (a) An stochastic texture with random texture patches, (b) texture patches arranged without orientation and (c) unoriented texture patches without boundaries.

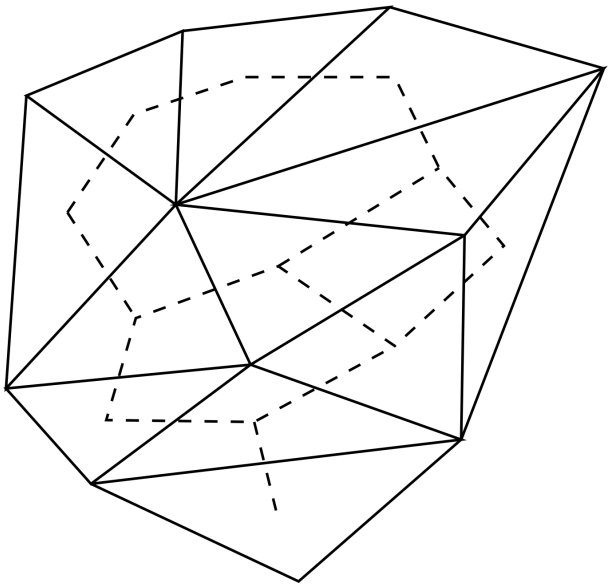


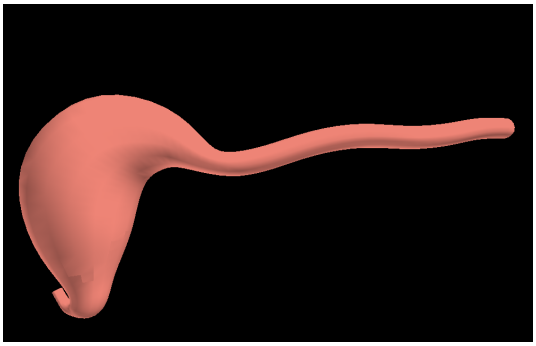
Figure 4.4: A triangle mesh (solid line) and its dual graph (dotted lines).

Algorithm 2 mappingTexture

Input 3D triangle Mesh M, texture T

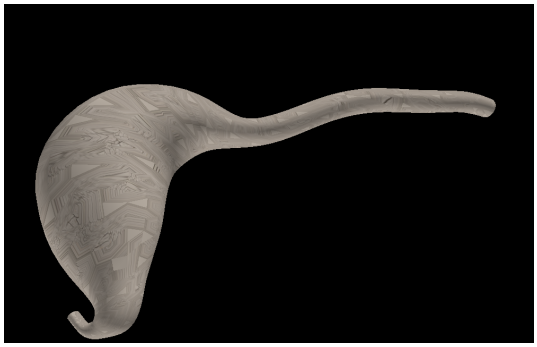
Output 3D triangle mesh with mapping co-ordinates

1: Graph G = generateGraphFromMesh(3D Triangle Mesh)
2: Pick a random triangle t as seed
3: **repeat**
4: Use graph G to select a patch S around seed
5: Flatten S over T
6: Assign mapping parameter in the triangle mesh
7: Get new seed around the previous seed
8: **until** 3D Mesh is covered

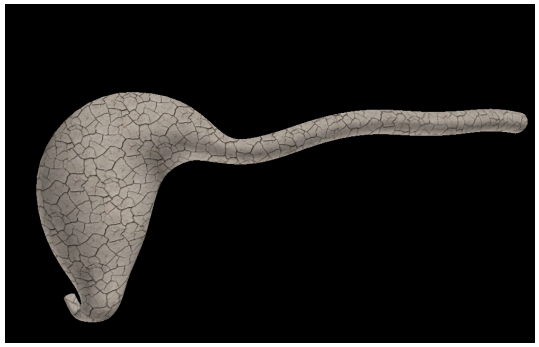


(a) Original 3D triangle mesh before mapping

(b) Texture used to map the 3D triangle mesh



(c) Mapping of texture with random seeds



(d) Mapping of texture with breadth-first seeds

Figure 4.5: Showing the difference between choosing random seeds and breadth-first seeds

Chapter 5

IMPLEMENTATION

5.1 Extraction of Model

We used ITK-SNAP [31] tool to extract the surface of upper GI tract and stomach. It uses a region growing snake method. The parameters are: balloon force 1.0, curvature force 0.20, advection force 5.0. Extracted surface is retrieved as triangle mesh (shown in fig-5.1).

5.2 Software Design and user Interface

We develop the tool in C++ using OpenGL [26] for graphical visualization of the mesh. We are using the QT [27] framework for the GUI.

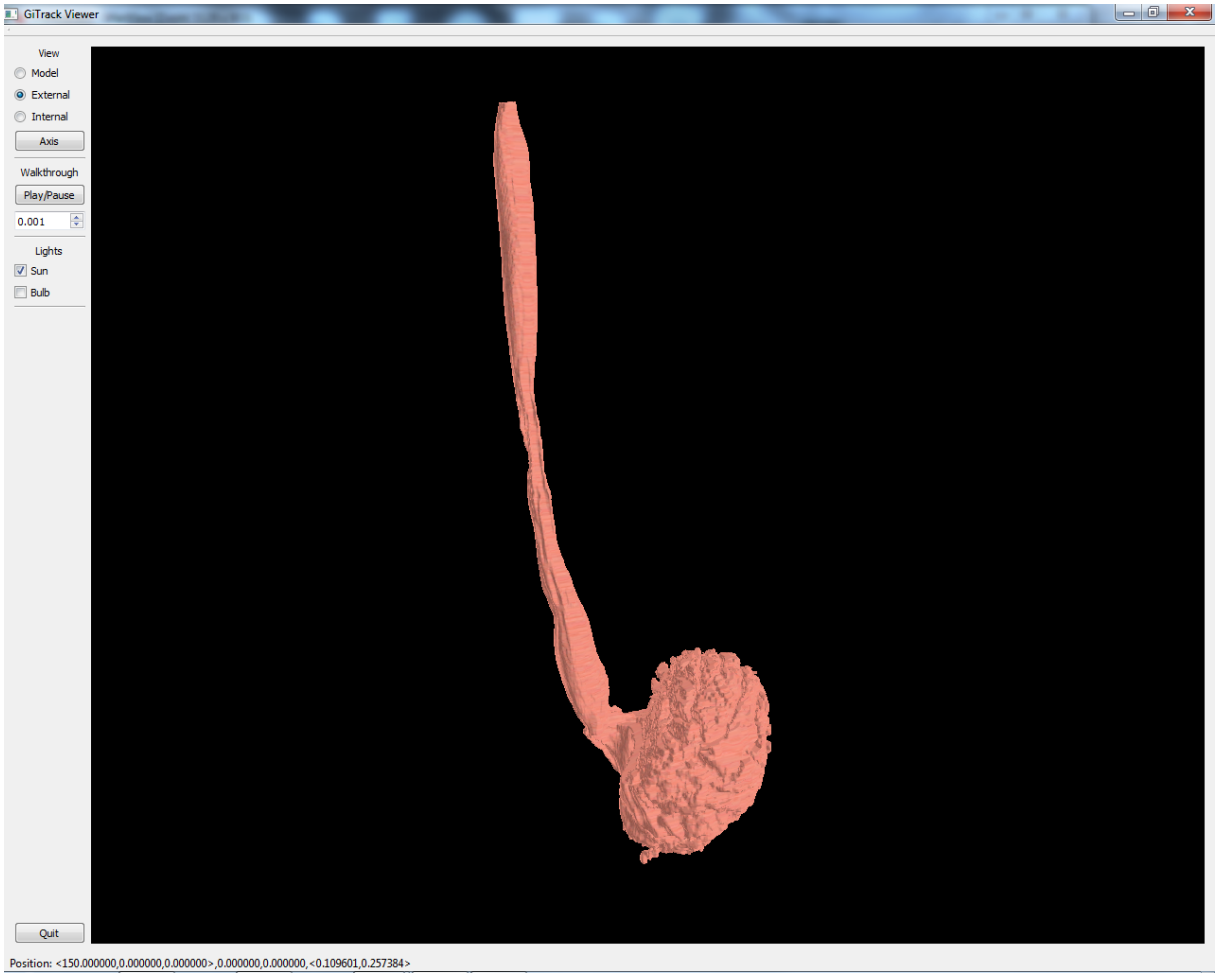


Figure 5.1: Snapshot of the tool showing the extracted model

Chapter 6

EVALUATION AND RESULTS

We develop two different interfaces to interact with the model. First one is used to interact with the outer surface of the extracted model. We used quaternion [24] to achieve a good interaction with user. Second interface is used for viewing the surface of the extracted model from inside, which is similar to endoscopy. User can use a fly-through to view the inner surface of the extracted model. Our tool currently supports model view (fig-6.1), external view (fig-5.1), fly-through (fig-6.2) and automated walk-through of upper GI tract. It is also able to map any texture patch to the obtained model. In external view, a dummy model of GI tract and stomach is used with human skeleton to show the position of used organs.

We compute the running time of algorithms, which are described in section 4.3, on a 2.0 GHz 2 CPU Intel Xeon machine with 8 GB of RAM. It has an nVidia GeForce 8800 GTX with 4 GB of video memory. Results are given in Table-6.1. Performance of the tool varies with the view modes. We got an average of 912 frames per second in the model view. Which is expected as the triangle count in model view is 28547. However, we got an average of 21 frames per second in external and internal view because the triangle count is 667900. Frame rate is calculated at a resolution of 1280 by 1024 pixels.

Currently our tool does not detect collisions while using fly-through mode. It can be really helpful if the tool responds with a force feedback on collision. However that will require interactive devices with force feedback such as Haptic device [14].

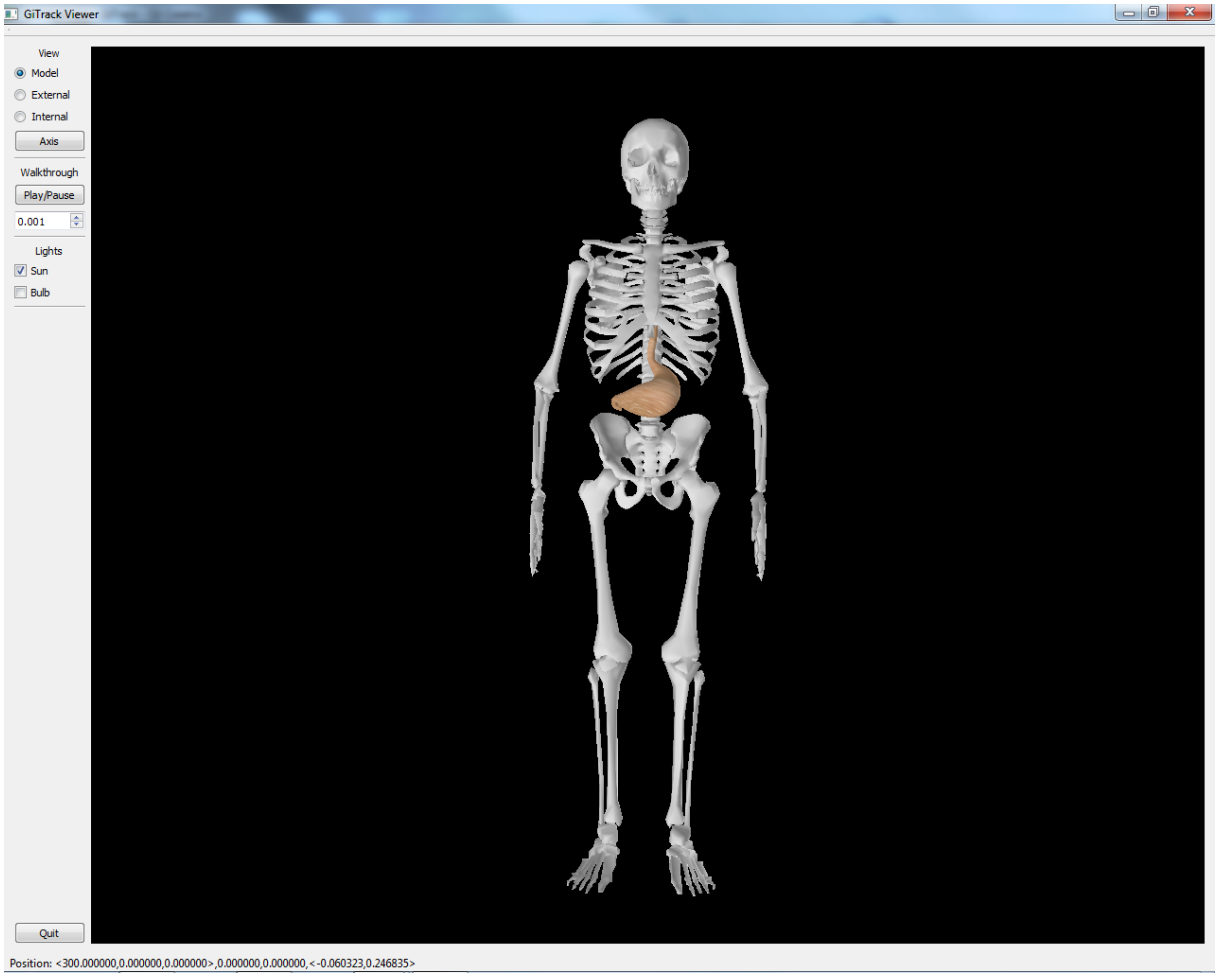


Figure 6.1: Snapshot of the tool showing model view

Table 6.1: Running time of $generateGraphFromMesh$ and $mappingTexture$.

Size of 3D mesh	Running Time (in seconds)	
(in number of triangles)	Algorithm-1	Algorithm-2
2560	0.016	0.001
667900	1.919	0.328
4007400	12.015	1.950

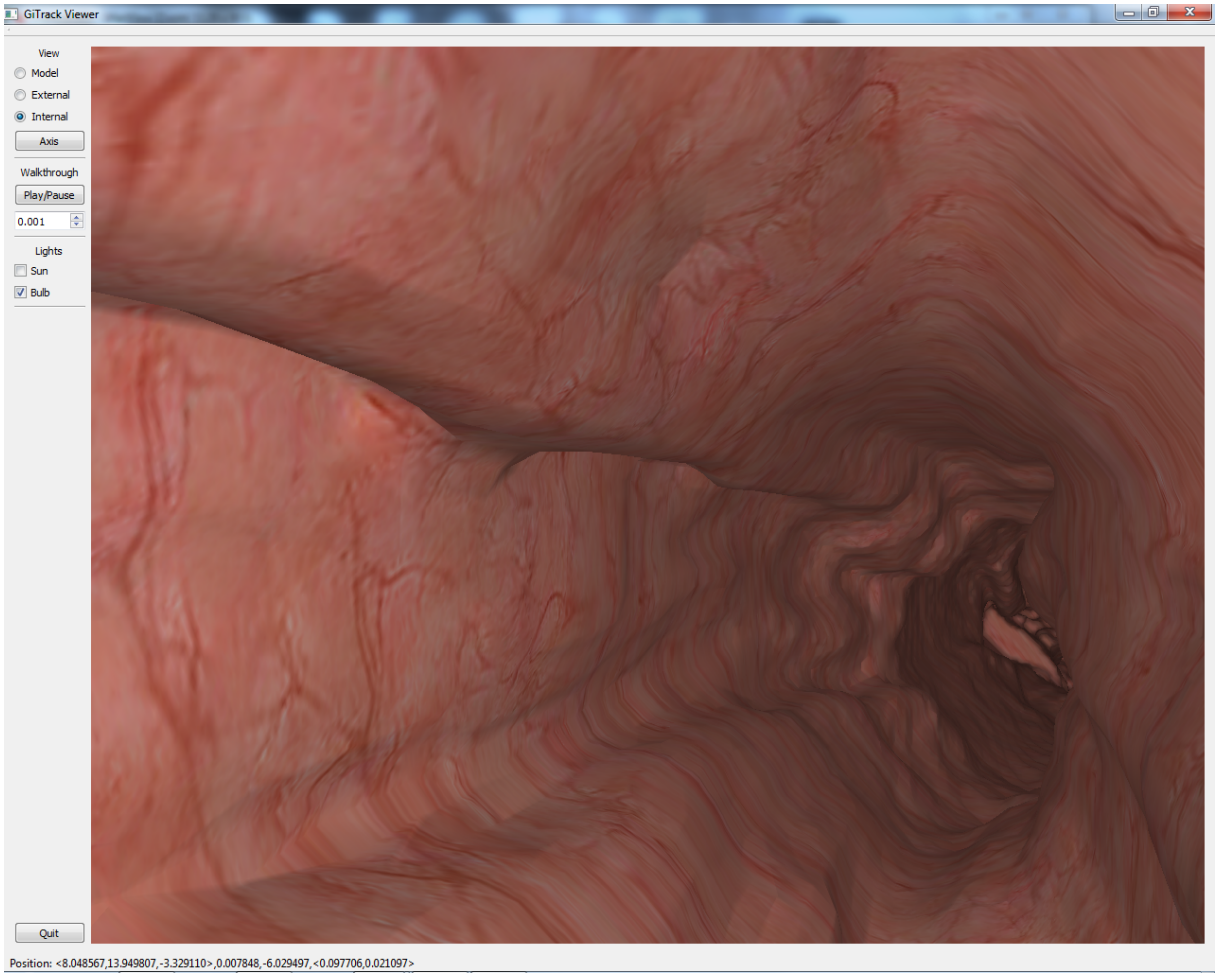


Figure 6.2: Snapshot of the tool showing interior of the extracted model

Chapter 7

Conclusions

We have proposed an interactive virtual endoscopy system. We extracted a good quality model of upper GI tract and stomach from a high resolution data and we developed an algorithm to calculate the automated walk-through of upper GI tract from this model. We also developed the tool with basic functionalities.

We proposed an interactive endoscopy viewer that highlights the internal organ. Currently our tool does not have a hardware support for force-feedback. Viewing part can be improved by applying custom shaders.

Appendix A

Framework

The tool **GiTract Viewer** can be viewed as two major layers of abstraction as shown in figure-A.1.

The first layer is **QGLWidget**, which is responsible for handling user operations and displaying the rendered output on the screen. It handles all the GUI tools, mouse and keyboard interactions. It basically includes $glwidget.h$.

Second layer is **3D engine**, which is responsible for rendering the objects. It handles all the objects related to the OpenGL rendering, B-Spline curve and required mathematical classes. It include $gmath.h$,$curves.h$, $trackball.h$ and $3deng.h$. This layer act as an abstraction of OpenGL. It appropriatly calculates the OpenGL function calls along with their parameters and automatically handles their sequence.

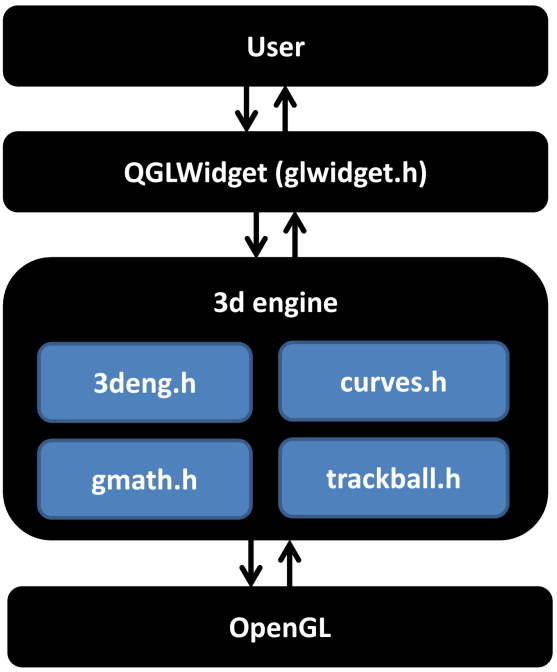


Figure A.1: Abstraction of the tool

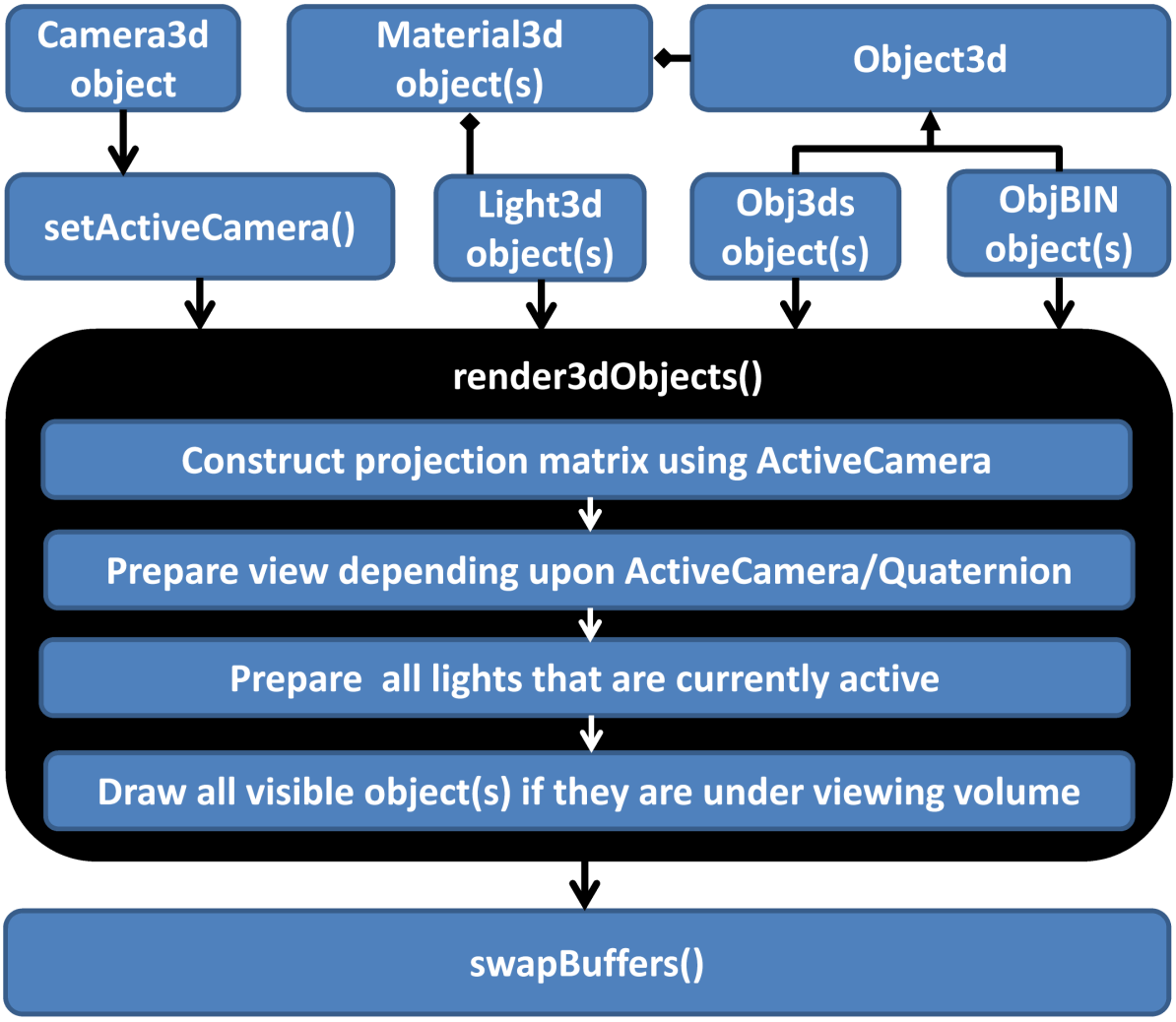


Figure A.2: Internal working of 3D engine

Appendix B

Headers

B.1 cmath.h

B.1.1 Class Vector2f

Stores a 2D vector and perform operations related to 2D vector.

Data members of Vector2f

- float x , y

Table B.1: Methods of Vector2f

Method/Operator	**Comment**
Vector2f()	constructor x = y = 0
Vector2f(a)	constructor x = y = a
Vector2f(a,b)	constructor x = a, y = b
set(a,b)	assign x = a and y = b
operator =	overloading assignment operator
operator +	does vector addition
operator -	does vector subtraction
operator *	does element by element multiplication
operator /	does element by element division

B.1.2 Class Vector3f

Stores a 3D vector and perform operations related to 3D vector.

Data members of Vector3f

- float x, y, z

Table B.2: Methods of Vector3f

Method/Operator	**Comment**
Vector3f()	Constructor x = y = z = 0
Vector3f(a)	Constructor x = y = z = a
Vector3f(a,b,c)	Constructor x = a, y = b, z = c
Vector3f(Vector4f)	Constructor from first three dimensions of Vector4f
set(a,b,c)	Assign x = a, y = b and z = c
operator =	Overloading assignment operator
operator + operator +=	Does vector addition
operator - operator -=	Does vector subtraction
operator * operator *=	Does element by element multiplication
operator / operator /=	Does element by element division
operator &	Returns dot product of two vectors
operator ^	Returns cross product of two vectors
distance(Vector3f a)	Returns distance between this vector and vector a
magnitude()	Returns the magnitude of this vector
normalize()	Returns the normalized vector of this vector
rotate(xy,yz,zx)	Returns rotated vector along xy, yz and zx planes in that order

B.1.3 Class Vector4f

Stores a 4D vector and perform operations related to 4D vector.

Data members of Vector4f

- float x, y, z, u

Table B.3: Methods of Vector4f

Method/Operator	**Comment**
Vector4f()	constructor x = y = z = 0
Vector4f(a)	constructor x = y = z = a
Vector4f(a,b,c,d)	constructor x = a, y = b, z = c, u = d
Vector4f(Vector3f a)	constructor x = a.x, y = x.y, z = a.z, u = 0
set(a,b,c,d)	assign x = a, y = b, z = c and u = d
operator =	overloading assignment operator
operator +	does vector addition
operator -	does vector subtraction
operator *	does element by element multiplication
operator /	does element by element division

B.2 curves.h

B.2.1 Type Definitions

- typedef double POINTTYPE

B.2.2 Structures

struct $Point$

- POINTTYPE x, y, z
- method **set**(X, Y, Z) sets $x = X$, $y = Y$ and $z = Z$

B.2.3 Class BSpline

Represents a B-Spline curve. Contain data members and methods to approximate a B-Spline curve from external data points. For more information on B-Spline curve see section-3.3 and section-4.1.

Table B.4: Data members of BSpline

int $\_n$	Number of control points are $\_n + 1$
Point* cp	Array of control points
int $\_m$	Number of knots are $\_m + 1$
double* kv	Array of knots
int $\_d$	Degree of the curve

Table B.5: Methods of BSpline

Method/Operator	**Comment**
BSpline()	Constructor _m = _n = _d = 0, cp = kv = NULL
~BSpline()	Destructor
initCurve(controlPoints, degree)	Initializes the curve
getD()	Returns the degree of the curve
getN()	Returns the number of control points
getM()	Returns the number of knots
N(i,p,u)	Returns the result of basis function $N_{i,p}(u)$
C(u)	Returns the coordinate of a point on curve at position u, where $0 \leq u \leq 1$
derivative(u)	Returns the derivative of curve at position u, where $0 \leq u \leq 1$
lsApprox(h, Point*)	Constructs the B-spline curve, given h+1 data points

B.3 trackball.h

B.3.1 Class TrackBall

Data members of TrackBall

- QQuaternion $m\_rotation$ = QQuaternion() (constructor default)
- QVector3D m_axis
- float $m\_angularVelocity$
- QPointF $m\_lastPos$
- QTime $m\_lastTime$ = QTime::currentTime() (constructor default)
- bool m_paused = false (constructor default)
- bool $m\_pressed$ = false (constructor default)
- TrackMode m_mode

Table B.6: Methods of TrackBall

Method/Operator	**Comment**
TrackBall(mode)	constructor m_angularVelocity = 0, m_mode = mode, m_axis = QVector3D(0, 1, 0)
TrackBall(angularVelocity, QVector3D axis, mode)	constructor m_angularVelocity = angularVelocity, m_mode = mode, m_axis = axis
push(p,transformation)	reset m_lastPos to 'p'
move(p,transformation)	move according to current position 'p' and given transformation
release(p,transformation)	move and set m_pressed to false
start()	records the m_lastTime as current time
stop()	rotate according to the time and pause
QQuaternion rotation()	apply rotation according to the time difference between current time and m_lastTime

B.4 3deng.h

B.4.1 Constants

- HALF_PI = 1.57079632679
- PI = 3.14159265358
- OBJINFINITY = 99999.9

B.4.2 Structures

struct $Texture2d$

- GLuint id
- long w, h
- unsigned char* buf

struct $TriMesh$

- unsigned long a, b, c

B.4.3 Hierarchy of 3D Object Classes

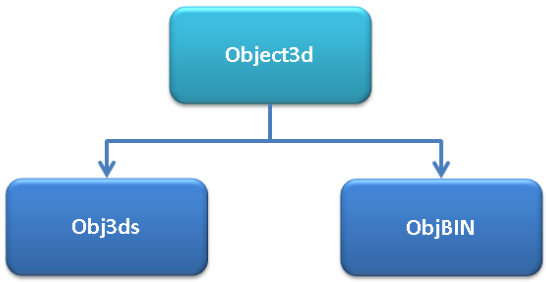


Figure B.1: Derivative classes of Object3d

B.4.4 Class Camera3d

Table B.7: Data members of Camera3d

Vector3f pos	position of the camera
Vector3f eye	where they are pointing
Vector3f ori	orientation of the camera
float ang	viewing angle of camera
float $cnear$	camera's near cut-off distance
float $cfar$	camera's far cut-off distance
float ar	aspect ratio
bool $isOrtho$	project orthogonal or not
Vector3f $orthoMin$	left, bottom and near cut-offs
Vector3f $orthoMax$	right, top and far cut-offs

B.4.5 Class Material3d

Table B.8: Data members of Material3d

Vector4f $ambi$	Ambient illumination
Vector4f $diff$	Diffuse illumination
Vector4f $spec$	Specular light
Vector4f $emis$	Emission color of material
GLfloat $shin$	Specular exponent

void setColor(const Vector4f& color)

Set 'ambi' and 'diff' properties of this Material3d object based on given color.

B.4.6 Class Light3d

Table B.9: Data members of Light3d

bool $castShadow$	Toggle shadow
bool $isOn$	Toggle light
Material3d mat	Material properties of light
Vector4f pos	Position or direction (in case of directional light)
Vector3f att	Attenuation $x + y * dist + z * dist * dist$
bool $isSpot$	True for Spot-light
Vector3f $sdir$	Direction of Spot-light
GLfloat $cutoff$	Spot-light cutoff, default:45
GLfloat exp	Spot-light exponent, default:2.0
GLfloat $cone$	Spot-light cone alpha, default:0.0

void runGLcmd(int light)

Call OpenGL fuctions for current this Light3d object

B.4.7 Class Object3d

Contains data member and methods to represent a 3D model. Only Object3d instances are rendered by the function 'void render3dObjects()'.

Table B.10: Data members of Object3d

bool $objType$	Default: 0 (none)
bool $isCollidable$	Make object collidable; Default: false
bool $isExtreamObj$	True(Default) if it's an immovable object or has unstoppable force
bool $isHidden$	True value does not render the object; Default: false
bool $queryCollision$	True: if it's a hit
bool $castShadow$	Toggle shadow
Vector3f $cubeBoundMin$	Bounding Box Minimum
Vector3f $cubeBoundMax$	Bounding Box Maximum
Vector3f col	color, Default: 0.7, 0.7, 0.7
Vector3f $texCol$	texture color, Default: 1,1,1
Texture2d* tex	Pointer to the texture of this object
GLfloat $texEnvParam$	Texture environment parameter; Default: GL_MODULATE
Material3d* mat	Material properties of this object
GLuint $dispList$	Display list id
Vector3f $factorTra$	3d offset from it's position
Vector3f $factorRot$	Orietation relative to it's original orientation
Vector3f $factorMul$	Scaling of object relative to it's original size

Table B.11: Methods of Object3d

Method/Operator	**Comment**
Object3d()	Constructor
˜Object3d()	Destructor
void operatorAssign(Object3d)	Copy properties from another object
void bindTexture(Texture2d *t)	Bind the texture to this object
void drawBounds()	Draws cuboid bounds around this object
void makeBoundsEql()	Make bounds a cube
$virtual$ void updateDisplayList()	Recreate display list for this object
$virtual$ void draw()	Handles draw method for this object
$virtual$ void calcBoundingCube()	Calculate cuboid bounds for this object

B.4.8 Class Obj3ds

Extends the class Object3d to load triangle mesh from a **3DS** (©Autodesk, Inc.) file. To read specification visit `http://www.martinreddy.net/gfx/3d/3DS.spec`.

Table B.12: Data members of Obj3ds

char $name[24]$	Object name
unsigned long $numVert$	Number of vertices
Vector3f* $vertex$	Array of vertices
Vector3f* $normal$	Array of normals
unsigned long $numPoly$	Number of Polygons (in 3ds it's triangles)
TriMesh* $triangle$	Array of triangles, containing indices of vertices
Vector2f* $mapcoord$	Texture mapping coordinates
bool $smooth$	Use precalculated normals

Table B.13: Methods of Obj3ds

Method/Operator	**Comment**
Obj3ds()	Constructor
bool Load3ds(char *filename)	Load the 3d object from the given file
calcNormals()	Calculate normal per vertex for smooth mesh
operator =	Overloading assignment operator
operator += (Vector3f)	Translate object by given amount
scale(Vector3f)	Scale object by given amount
rotate(xy,yz,zx)	Rotat object along xy, yz and zx planes in that order
draw()	Render the object with current user setting
calcBoundingCube()	Calculate cuboid bounds for the object

B.4.9 Class ObjBIN

Extends the class Object3d to load binary dump of 3D object. Used for fast loading of huge 3D triangle mesh.

Table B.14: ObjBIN File Format Specifications

Offset (in bytes)	**Type/Size**	**Comment**
0	32bit uint	Contains number of vertices (numVert)
4	32bit uint	Contains number of polygons (numPoly)
8	$12 \times numVert$	Array of vertices in Vector3f data type
$8 + 12 \times numVert$	$12 \times numPoly$	Array of triangles in TriMesh data type
$8 + 12 \times numVert + 12 \times numPoly$	$8 \times numVert$	Array of mapping coordinates in Vector2f data type

Table B.15: Data members of ObjBIN

unsigned long $numVert$	Number of vertices
Vector3f* $vertex$	Array of vertices
Vector3f* $normal$	Array of normals
unsigned long $numPoly$	Number of Polygons (in binary dump it's triangles)
TriMesh* $triangle$	Array of triangles, containing indices of vertices
Vector2f* $mapcoord$	Texture mapping coordinates
bool $smooth$	Use precalculated normals

Table B.16: Methods of ObjBIN

Method/Operator	**Comment**
ObjBIN()	Constructor
bool LoadBIN(char *filename)	Load the 3d object from the given file
bool saveBIN(char *filename)	Dump the 3d object from the given file
calcNormals()	Calculate normal per vertex for smooth mesh
operator =	Overloading assignment operator
operator += (Vector3f)	Translate object by given amount
scale(Vector3f)	Scale object by given amount
rotate(xy,yz,zx)	Rotat object along xy, yz and zx planes in that order
draw()	Render the object with current user setting
calcBoundingCube()	Calculate cuboid bounds for the object

B.4.10 Class Obj3dGraph

Extends ObjBIN to implement the lapping of texture in binary dump.

Data members of Obj3dGraph

- TriMesh* $elist$

Table B.17: Methods of Obj3dGraph

Method/Operator	**Comment**
Obj3dGraph()	Constructor elist = NULL
~Obj3dGraph()	Destructor
calcGraph()	Calculate the dual graph of the triangle mesh
lap()	Implementation of lapping texture, see section-4.3

B.4.11 Methods

Texture2d* loadBMP(char *filename)

Loads the Bitmap file (*.bmp) into the texture structure.

On success a pointer to the Texture2d structure is returned. On failure a null pointer is returned.

bool appendBMP(Texture2d *tex, char *filename)

Loads the alpha component from the given Bitmap file (*.bmp) in the given texture.

Bitmap file must be a grayscale image.

On success a returns true otherwise false is returned.

int getTotalTextures()

Returns the total number of textures read.

void setRenderTexture(bool)

Sets whether to render the textures or not.

bool isRenderTexture()

Returns the status that textures are currently rendering or not.

void setDrawBounds(bool)

Sets whether to draw cuboid boundary around objects or not.

bool isDrawBounds()

Returns the status to draw bounds or not.

void setLighting(bool v)

Sets the lighting for the OpenGL.

bool getLighting()

Returns the status of OpenGL lighting.

void updateDisplayList()

Calls the method 'updateDisplayList()' for all 'Object3d' instances.

void runForAllObject3d(void(*func)(Object3d*))

Calls the function 'void func(Object3d*)' for all 'Object3d' instances.

void setActiveCamera(Camera3d)

Sets the given Camera3d object as the default camera for 'render3dObjects'.

void setQuaternion(bool v)

Sets the default camera to behave like trackball view using quaternion.

bool getQuaternion()

Returns the status of quaternion.

void setActiveViewTrackBall(TrackBall)

Sets the given TrackBall object as the current trackball for 'render3dObjects'.

void render3dObjects()

Renders all the Object3d instances under the default Camera3d and other user setting.

unsigned int getNumRenderedPolygons()

Returns the number of rendered polygons in the last call of 'render3dObjects'.

References

[1] M.J. Ackerman. The visible human project. *Proceedings of the IEEE*, 86(3):504–511, 1998.

[2] Chakib Bennis, Jean-Marc Vézien, and Gérard Iglésias. Piecewise surface flattening for non-distorted texture mapping. In *ACM SIGGRAPH computer graphics*, volume 25, pages 237–246. ACM, 1991.

[3] James F Blinn and Martin E Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, 1976.

[4] Kambiz Darabi, KDM Resch, J Weinert, Udo Jendrysiak, and A Perneczky. Real and simulated endoscopy of neurosurgical approaches in an anatomical model. In *CVRMed-MRCAS'97*, pages 323–326. Springer, 1997.

[5] Carl De Boor, Christian Gout, Angela Kunoth, and Christophe Rabut. Multivariate approximation: theory and applications. an overview. *Numerical Algorithms*, 48(1-3):1–9, 2008.

[6] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1033–1038. IEEE, 1999.

[7] Gerald Farin and Phillip J Barry. Link between bézier and lagrange curve and surface schemes. *Computer-Aided Design*, 18(10):525–528, 1986.

[8] A. Ferlitsch, P. Glauninger, A. Gupper, M. Schillinger, M. Haefner, A. Gangl, R. Schoefl, et al. Evaluation of a virtual endoscopy simulator for training in gastrointestinal endoscopy. *Endoscopy*, 34(9):698–702, 2002.

[9] Paul S Heckbert. Survey of texture mapping. *Computer Graphics and Applications, IEEE*, 6(11):56–67, 1986.

[10] L. Hong, A. Kaufman, Y.C. Wei, A. Viswambharan, M. Wax, and Z. Liang. 3d virtual colonoscopy. In *Biomedical Visualization, 1995. Proceedings.*, pages 26–32, 1995.

[11] Lichan Hong, Shigeru Muraki, Arie Kaufman, Dirk Bartz, and Taosong He. Virtual voyage: Interactive navigation in the human colon. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 27–34. ACM Press/Addison-Wesley Publishing Co., 1997.

[12] K. Ikuta, M. Takeichi, and T. Namiki. Virtual endoscope system with force sensation. *Medical Image Computing and Computer-Assisted Interventation—MICCAI'98*, pages 293–304, 1998.

[13] F.A. Jolesz, W.E. Lorensen, H. Shinmoto, H. Atsumi, S. Nakajima, P. Kavanaugh, P. Saiviroonporn, S.E. Seltzer, S.G. Silverman, M. Phillips, et al. Interactive virtual endoscopy. *American Journal of Roentgenology*, 169(5):1229–1235, 1997.

[14] H Kazerooni and Ming-Guo Her. The dynamics and control of a haptic interface device. *Robotics and Automation, IEEE Transactions on*, 10(4):453–464, 1994.

[15] S. LAKARE, WAN Ming, A. KAUFMAN, Z. LIANG, and WAX Mark. An automatic colon segmentation for 3d virtual colonoscopy. *IEICE TRANSACTIONS on Information and Systems*, 84(1):201–208, 2001.

[16] Hans Køhling Pedersen. Decorating implicit surfaces. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 291–300. ACM, 1995.

[17] Hans Køhling Pedersen. A framework for interactive texturing on curved surfaces. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 295–302. ACM, 1996.

[18] WK Pratt, OD Faugeras, and A Gagalowicz. Applications of stochastic texture field models to image processing. *Proceedings of the IEEE*, 69(5):542–551, 1981.

[19] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 465–470. ACM Press/Addison-Wesley Publishing Co., 2000.

[20] Christof Rezk-Salama, Peter Hastreiter, Christian Teitzel, and Thomas Ertl. Interactive exploration of volume line integral convolution based on 3d-texture mapping. In *Proceedings of the conference on Visualization'99: celebrating ten years*, pages 233–240. IEEE Computer Society Press, 1999.

[21] RA Robb. Virtual endoscopy: development and evaluation using the visible human datasets. *Computerized Medical Imaging and Graphics*, 24(3):133–151, 2000.

[22] S.P. Schrag, R. Sharma, N.P. Jaik, M.J. Seamon, J.J. Lukaszczyk, N.D. Martin, B.A. Hoey, S.P. Stawicki, et al. Complications related to percutaneous endoscopic gastrostomy (peg) tubes. a comprehensive clinical review. *Journal of Gastrointestinal and Liver Diseases*, 16(4):407, 2007.

[23] CK Shene. Introduction to computing with geometry. *Lecture Notes, Michigan Technological University, http://www. cs. mtu. edu/˜ shene/COURSES/cs3621/NOTES*, 2010.

[24] K. Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH computer graphics*, 19(3):245–254, 1985.

[25] OpenGL texture mapping. http://www.opengl.org/.

[26] OpenGL Webseite. http://www.opengl.org/.

[27] QT Webseite. http://qt-project.org/.

[28] Roni Yagel, Don Stredney, Gregory J Wiet, Petra Schmalbrock, Louis Rosenberg, Dennis J Sessanna, and Yair Kurzion. Building a virtual environment for endoscopic sinus surgery simulation. *Computers & Graphics*, 20(6):813–823, 1996.

[29] Fujio Yamaguchi and Fujio Yamaguchi. *Curves and surfaces in computer aided geometric design*. Springer-Verlag Berlin, 1988.

[30] S. You, L. Hong, M. Wan, K. Junyaprasert, A. Kaufman, S. Muraki, Y. Zhou, M. Wax, and Z. Liang. Interactive volume rendering for virtual colonoscopy. In *Visualization'97., Proceedings*, pages 433–436, 1997.

[31] P.A. Yushkevich, J. Piven, H.C. Hazlett, R.G. Smith, S. Ho, J.C. Gee, and G. Gerig. User-guided 3d active contour segmentation of anatomical structures: significantly improved efficiency and reliability. *Neuroimage*, 31(3):1116–1128, 2006.