3D Spectral Imaging

Eyad Khader

MASTER'S THESIS

University of Joensuu

Department of Computer Science and Statistics

P.O. Box 111, FIN-80101 Joensuu, Finland

May 31, 2007

Abstract

The process of mapping 2D textures to a 3D model is an important step to make 3D models more realistic and more informative. This process is an essential step in many applications of the 3D graphics, such as virtual reality, 3D medical application and video games.

This thesis tackles: Automatic texture mapping of uncalibrated cameras, 3D-3D registration, Automatic spectral texture mapping to 3D model, lightning, 3D Digitizers, mesh triangulations, spectral analyzing tools and model manipulation.

We give an overview of some of the possible approaches we investigated to achieve the automatic 2D-3D registration "Texture mapping" along with a discussion on the difficulties in those approaches. In the second part of the 2D-3D registration section we propose our algorithm and the solution for the automatic 2D-3D registration.

The 2D-3D registration algorithm solves the problem of calibration by using feature points obtained from the spectral image and the scanner RGB image. The algorithm depends on finding the transformation matrix that can reorient the 3D model to match the position in which the spectral image was taken from. The algorithm showed very good results along with fast processing time "less than 2 seconds".

To obtain the 3D model we used Vivid9i Digitizer to scan the real model, and thus we had to rotate the model 90 degrees on each scan to get a full representation of that model. Those scans needed to be merged and that what the 3D-3D registration algorithm does, it merges the different scans of the model into one complete model. This algorithm is semi-automatic where the user has to choose five different points on both of each two scans and the algorithm will compute the transformation matrix to merge them together. The algorithm results were very good with fast and easy use for the user.

Finally we added more features such as mesh triangulations, lightning, spectral analysis tools and model manipulation to increase the usability of our research. We integrated the use of Vivid9i scanner into the software along with ability to export the triangulated textured mesh into different types of 3D formats.

Keywords: 2D-3D registration, Automatic texture mapping, spectral images, 3D-3D registration, 3D Digitizer, mesh triangulation.

Table of contents

Chapter 1: Introduction……1

1.1 Motivations……..……..……..……1
1.2 Background……..……..……1
1.3 Main Contributions…...…….2
1.4 Summary of Chapters……..……..2

Chapter 2: Related Work……...……..……4

Chapter 3: 2D-3D Texture Mapping……7

3.1 Problem Formulation and Different Investigated Solutions….……..……7

3.2 The Proposed Method and the Mathematical Concepts……..……13

3.3 The 2D-3D Registration Algorithm…….……15

Chapter 4: 3D-3D Registration……..……24

4.1 Problem Formulation……..……25

4.2 Finding the Transformation Matrix……..……26

4.3 Acquiring the Reference and Target Vectors…….……33

Chapter 5: Data Files, Mesh Triangulation, Lightning System and Other Features…….….34

5.1 Data Files…….……34

5.2 Triangulating the Mesh…….37

5.3 Lighting System……..……40

5.4 Other Features……..…….43

Chapter 6: Experiments……..……55

6.1 The 2D- 3D registration Results……55
6.2 3D-3D Registration……56
6.3 Spectral Analysis Tools..……57
6. 4 Model Manipulation..……58
6. 5 Data Files……..……58
6. 6 Controlling the Digitizer……..……59
6. 7 The Lightening System…/……59
Chapter 7: Conclusions……59

Bibliography……..……..……62

C h a p t e r 1: Introduction

In this chapter we will talk about our motives, give a brief background on the thesis and the main problem that this thesis address, we will also give a brief summary of chapters.

1.1 Motivations.

So far the study of spectral images has been focused on 2D images obtained from spectral cameras. Our main objective in this thesis is to open the possibilities of studying the spectral images in real time 3D view. We proposed the use of the 3D scanner to obtain the model and the use of Nuance LCTF spectral camera to obtain the spectral images. This approach will allow the researchers interested in color field to simulate the model in a 3D virtual environment. This approach will also allow observing the real time changes on the spectral images when they interact with the virtual lightning system we integrated in this project.

The use of a spectral camera from another position other than the calibrated RGB scanner camera gave birth to a new problem. The problem is mapping the spectral images to the 3D model.

One of our objectives in this thesis is to make the use of the 3D spectral imaging practical by introducing an automated process for mapping the spectral images to the 3D model. This objective motivated us to create the 2D-3D registration algorithm.

1.2 Background.

The main problem in the automatic 2D-3D registration is finding correspondent pixels on the spectral image for each of the vertices on the 3D model. In other words if we projected a vertex from the 3D model using perspective projection then where this ray will hit on the spectral image plane. This problem is due the difference in the positions between the spectral camera and the 3D scanner.

A simple solution for the problem will be in placing the camera in the exact location where the scanner camera took the image for the model. But this solution is far from being practical since

there are many variables need to be taken into consideration. These variables can be summarized in the focal length of the camera, the resolution and the model position in the resultant image. Another problem with this solution is the near impossible for positioning the camera in the exact same place after removing the scanner itself and since any small change will lead in errors in the mapping process.

Other solutions can be accomplished using manual registration. In manual registration the user will have to interact with the software in order to compute camera parameters and then maps the texture to the model. Though this method solves the problem but still it has withdraws such as the loss of texture information and the user dependency.

1.3 Main contributions.

Our main contribution in this field is the new method for automating the mapping of spectral texture to a 3D model without losing the texture information. We implemented this method into 3D Spectral Imaging System. This system made it easier, automated and much faster to take spectral images from different angles and different positions and map them to the 3D model automatically. The system supports SPB format “SPectral Binary images” and automatically converts, registers and maps the texture to the 3D model.

The system has many tools to control the Vivid9i scanner, manipulates the model, changes the lightning of the 3D model and a 3D-3D registration. The 3D-3D registration merges two or more scans for the same model into one complete 3D model. This functionality is implemented with semi-automatic transformation.

The system also allows the users to navigate through the spectral image channels and view each channel as a separate texture to the 3D model. The user can export their work into three different 3D file formats; DirectX .X, Wavefront .Obj and VVD Konica Minolta File formats.

1.4 Summary of chapters:

Chapter 2 talks about some alternative methods, their advantages and withdraws. It also briefly describes our proposed algorithm and solution for automating the texture mapping process.

Chapter 3 talks about the first part of our algorithm and discuss the following subjects in details and with their mathematical concepts: Image preparation and enhancement, Image registration: Finding feature points and correspondence and Computing the transformation Matrix and applying the transformation.

Chapter 4 talks about the 3D – 3D registration in details and discuss the underlying mathematical concepts. Chapter 5 talks about the mesh triangulation, lightning system, face normal computation and their mathematical concepts in details. It also talks about the other features portrayed in the software.

Chapter 6 discusses the results of our research, the challenges and how the system can be enhanced in the future. Chapter 7 talks about the conclusion of our research and what was accomplished in this thesis.

C h a p t e r 2: Related work

3D modeling is an essential part of many applications such as Virtual Reality VR, Virtual Museum, Urban scenes, 3D games, and many other applications. However in order to increase the realism of any model a texture maps have been used to achieve this realism.

There are many ways to produce 3D model such as 3D modeling using a graphics designer to produce such model, 3D reconstructions techniques and 3D scanning. The later method "3D scanning " is by far the most sophisticated method for creating realistic 3D models by using 3D Scanners "Digitizers" to obtain the actual model [1],[2],[3],[4],[5].

The use of texture maps on 3D models has become an essential process of any 3D modeling System [6],[7]. Texture mapping is usually done by a graphic designer in non critical system such as Virtual Reality "VR" and video games. But for critical systems where the texture mapping should be 100% accurate such as medical applications and in large projects the need for automation of the process becomes a must.

However the process of automating the texture mapping in robust way is yet to be achieved. The main problem in texture mapping is in knowing which pixel on the texture image corresponds to which vertex on the 3D model. Many methods and algorithms have been proposed to achieve the automation of texture mapping [8],[9],[10],[11],[12],[5].

In this thesis we propose a 2D-3D texture mapping algorithm that can automatically map spectral texture to a 3D model. Our algorithm is based on feature points to find the transformation matrix that can change the 3D model pose to match the spectral image. Finding the pose allows us to use the same camera parameters of the scanner calibrated camera and this will let us map the texture without losing the spectral information. Unlike our method some methods used feature points for texture mapping by flattening the 3D model and wrapping texture images [13],[14],[15],[16]. However we used the transformation matrix to change the model position to match the texture.

Simultaneous 2D images and 3D geometric model registration for texture mapping utilizing reflectance attribute[17] is an attempt to achieve this automation by using reflectance images that

produced by the 3D scanner. They took the edge points of both the reflectance and color images and then they aligned the edges by using the iterative pose estimation.

The reflectance images in [17] are actually a collection of the strength of the returned laser energy at each pixel, and since the scanner images and the reflectance images were obtained from the same scanner this means they are already aligned. So the returned time provides the depth and the strength provides the reflectance measurement.

The main method used in [17] consists of three stages:
1-Extract possible observable reflectance edges on 3D reflectance images and edges along the occluding boundaries based on the current viewing direction.
2- Establish correspondences between 3D edges and 2D intensity edges.
3- Determine the relative relationship iteratively based on the correspondences.

The method that was used in [17] depended on 3D edges to find the match with 2D intensity images. However this is not the case in our algorithm since we used 2D-2D correspondence to determine the optimal pose of the model.

Another approach to map texture from uncalibrated camera to a 3D model such as [18] uses the same idea as [17] in which the feature extraction occurs directly between the 3D model and the texture images.

The use of spectral images as texture to our 3D model allows us to observe the spectral images in 3D space, and thus allows us to manipulate the lightning and the model in real time manner [19]. Few systems have dealt with 2D-3D spectral imaging for example [20] proposed a full system for obtaining 3D model in historical buildings. The spectral textures were mapped to them using region matching. Yet in our algorithm we used feature points to find the optimal pose in which if it was projected it will give us the spectral image pose.

With the use of the 3D Scanner to obtain the 3D model the 3D-3D Registration problem rise to surface. The 3D Scanner can only obtain a partial region of the model and thus force us to take more than one scan of the same model from different positions. These scans will result in more

than one separated surfaces of the same model and they need to be merged into one complete model.

Many approaches were proposed to solve the 3D-3D registration problem for instant [21] proposed a method for aligning the 3D model scans. This method is based on Iterative Closest Point (ICP) algorithm, originally proposed by [22]. Another approaches for solving the 3D-3D registration proposed by [23][24], the main problem in 3D-3D registration is in finding the transformation matrix that can transform on mesh to another. And that's what [23] main purpose though the searching algorithm to find these transformations is used. In [24] the semi-automatic approach to find the 3D feature points and their correspondences was used to obtain the transformation matrix. The approach in [24] is similar to our approach except in the way the user can find the feature points. We used 2D images and left the user to interact with the software to find the feature points then we searched for the equivalence for these points in the 3D model.

In this thesis we also addressed the lightning in 3D scene and their effects over the 3D Spectral images. One problem in lightning in 3D virtual scenes is when taking the texture images you should consider light changes in all directions. Some solutions for this problem were proposed by [25][26], by using Color Alignment based on chromaticity consistency, Color Alignment based on Illumination sphere. However in our work we used spectral images and for this we kept the illumination constant by rotating the model itself instead of the spectral camera to avoid the lightning problem.

C H a p t e r 3: 2D – 3D texture mapping

In this chapter we will discuss in details the methods and algorithms used to complete the 2D-3D texture mapping. We will also explain and discuss the underlying principles and methods which lead to those results.

3.1 problem formulation and different investigated solutions

The main problem in this research is finding the texture coordinates that map each vertex on the 3D model to its correspondence pixel on the image plane, in other words texture mapping the model using an obtained image without camera calibration, and this is what the 2D-3D texture mapping algorithm does.

To understand the algorithm we need to revisit the problem in details. First we should imagine a 3D model projected using perspective projection through a pinhole camera placed on the following position:

1- The Lens center is placed at a positive point on the Z axis and it is called Viewpoint O.
2- The optical axis is aligned along the Z axis toward the negative direction.
3- The image plane is perpendicular to the Z axis and is placed at a distance of the focal length from the viewpoint toward the negative direction to eliminate the inverse case.

For more information on the Vivid9i perspective projection see [27].

Now that is said we can make a graph to illustrate the situation as in figure 3.1.

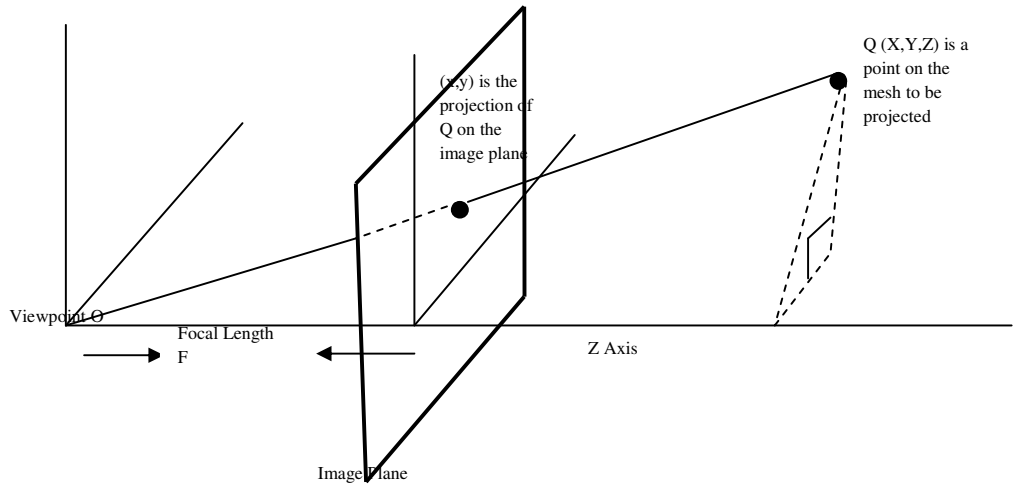

Figure 3.1: The perspective projection through a pinhole camera.

In the previous illustration “figure 1” we can see how the model is projected through the scanner camera. And since we know the Viewpoint O coordinate, focal length F and mesh points position Q; we can predict where the projection of Q will be on the image plane through the projection formula:

$$x = \frac{f(Q-O)X}{(Q-O)Z} \tag{3.1}$$

$$y = \frac{f(Q-O)Y}{(Q-O)Z} \tag{3.2}$$

Where:

1- f is the focal length.
2- Q a vector and it’s the mesh point position.
3- O a vector and it’s the viewpoint.
4- Z a vector and it’s the projection direction or where camera is placed.
5- Y is the up vector.
6- X is the gaze vector and its equal to $X = \frac{Z\ x\ Y}{|Z\ x\ Y|}$ where “x” refer to cross product operation and | | refer to normalizing the vector.
7- x y is the projection coordinate on the image plane.

However this formula works on the image taken from the scanner camera since we know the parameters. But we cannot use this projection formula on the images taken from the spectral camera since we don’t know the viewpoint or the focal length of the spectral camera.

To solve this problem we investigated tow approaches:

1- The first approach is to compute the camera parameters and find the viewpoint and the focal length of the spectral camera and then apply the projection formula.
   We can achieve this by finding a two feature points on the RGB scanner image and their correspondence on the spectral images. The RGB scanner image and the spectral image will be referred to as the reference and target images respectively.

After getting the feature points we can apply the projection formula as follow to get the camera parameters:

First we get two feature points refPoint1(x,y) and refPoint2(x,y) on the reference image plane and get from which vertices on the mesh they were projected. Let us call these two points Q1 and Q2 respectively.

Next we find the correspondent points tarPoint1(u1,v1) and tarPoint2(u2,v2) on the target image. Now let us assume that these points were projected from the same points Q1 and Q2 respectively. So we will get the following formula:

$$x_1 = \frac{f(Q_1 - O)X}{(Q_1 - O)Z} \tag{3.3}$$

$$y_1 = \frac{f(Q_1 - O)Y}{(Q_1 - O)Z} \tag{3.4}$$

To be able to use the raster u v coordinates from the reference and target points we should convert them into projection plane coordinate as the following:

$$x_1 = px * u_1 - px * ox$$

(3.5)

$$y_1 = py * oy - v_1 * py$$

(3.6)

$$x_2 = px * u_2 - px * ox$$

(3.7)

$$y_2 = py * oy - v_2 * py$$

(3.8)

Where:

ox is image width/2

oy is image height/2

px is always 0.0074;

py is always 0.0074;

And Z is always (0,0,-1)

And Y is always (0,1,0)

And X is always (1,0,0)

Q1 will be (X1,Y1,Z1)

Q2 will be (X2,Y2,Z2)

O will be (Ox,Oy,Oz)

The formula will be:

$$x_1 = \frac{fX_1 - fO_x}{O_z - Z_1} \tag{3.9}$$

$$y_1 = \frac{fY_1 - fO_y}{O_z - Z_1} \tag{3.10}$$

$$x_2 = \frac{fX_2 - fO_x}{O_z - Z_2} \tag{3.11}$$

$$y_2 = \frac{fY_2 - fO_y}{O_z - Z_2} \tag{3.12}$$

By solving for Ox,Oy and Oy we get the following:

$$O_x = \frac{(x_2 * y_1 * X_1 - x_2 * x_1 * Y_1 - y_2 * x_1 * X_2 + x_1 * x_2 * Y_2)}{x_2 * y_1 - x_1 * y_2} \tag{3.13}$$

$$O_y = \frac{y_1 * O_x - y_1 * X_1 + x_1 * Y_1}{x_1} \tag{3.14}$$

$$f = \frac{y_1 * y_2 * Z_2 - y_2 * y_1 * Z_1}{y_2 * Y_1 - y_2 * O_y + y_1 * O_y - y_1 * Y_2} \tag{3.15}$$

$$O_z = \frac{f * X_1 - f * O_x + x_1 * Z_1}{x_1} \tag{3.16}$$

Now this approach will work, however in this approach will lose some texture data since this will only change the location of the texture on the mesh. In other words some parts of

the texture will not appear on the mesh since we just moved the location of the texture on the mesh.

2- The second approach to solve this problem can be done if we place the target image over the reference image. In this case we will get a new point on the projection plane and a new a point from the mesh that match our new projected point from the target image as in figure 3.2:

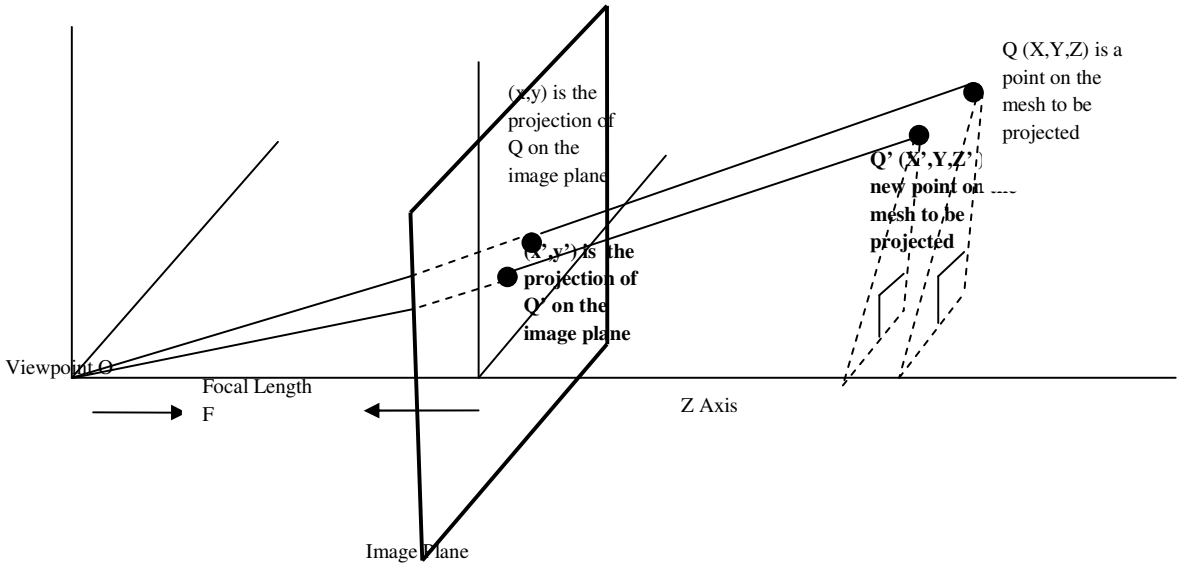

Figure 3.2: Merged image plans in perspective projection.

So the problem now can be solved if we found a transformation matrix that can be applied to point Q in order to give us Q' which its projection will give us (x',y'). So to write the transformation matrix we will consider only X rotation and Y rotation since we assumed that the reference and target images have been scaled to match each other.

The Z rotation can be easily controlled during the camera shot, and this will make it easier by solving for less unknowns.

$$R_x \times R_y = R_{xy} \tag{3.17}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{x} \quad \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad =$$

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ -\sin\varphi\sin\theta & \cos\varphi & -\sin\varphi\cos\theta & 0 \\ -\sin\theta\cos\varphi & \sin\varphi & \cos\varphi\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now if we have to formulate the problem in terms of mathematics we have the following formula:

$$QT = Q' \tag{3.18}$$

Where Q is the mesh vertex, T is the transformation matrix of X rotation and Y rotation and Q' is the transformed vertex. However we don't know the Q' and we don't know the transformation matrix, so it's impossible for us to solve this formula as it is. So we can use the projection formula again to solve this problem.

$$x' = \frac{f(QT - O)X}{(QT - O)Z} \tag{3.19}$$

$$y' = \frac{f(QT - O)Y}{(QT - O)Z} \tag{3.20}$$

Where x',y' are the target image correspondent points and T is the transformation matrix. However this equation is non linear equation and it can be solved by many methods for example optimization methods.

One reason for not using this method is the computation time required to solve this equation using optimization methods and the difficulty in solving it.

3.2 The proposed method and the mathematical concepts

The proposed method is the solution for finding the transformation matrix T that if it was multiplied by a vertex and projected onto the target image plane it will be projected on the correspondent point that is equivalent to the reference point.

Now to look at the solution let us observe figure 3.3:

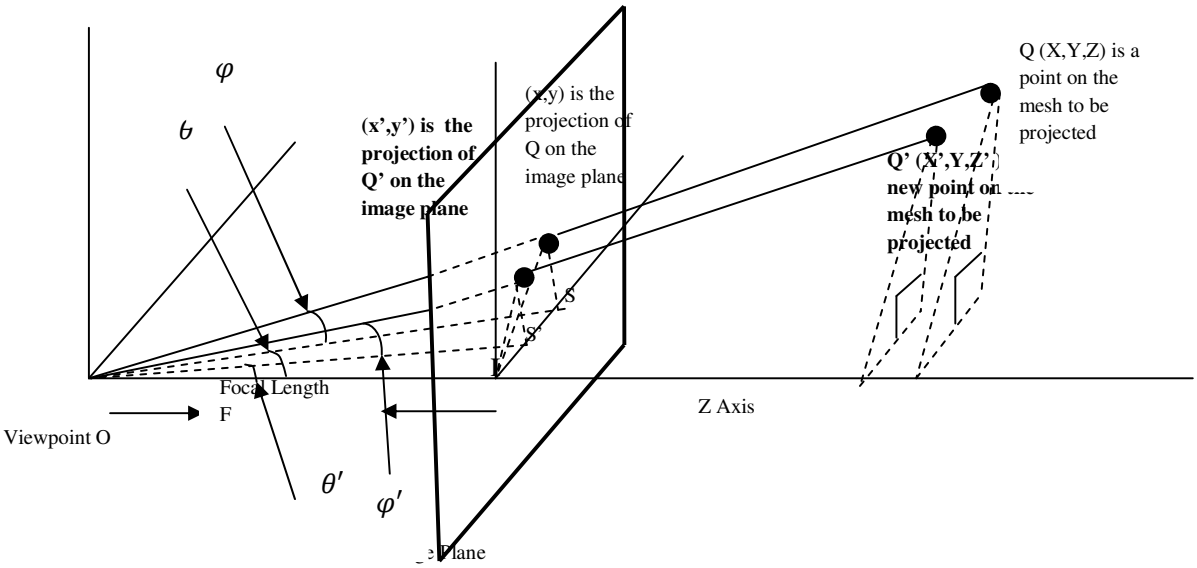

Figure 3.3: Angles calculation in merged image plans through perspective projection.

Let us assume the transformation matrix T is composed of X and Y axes' rotation. We need to find the rotation angles of X and Y by finding θ, θ' and then finding the delta theta of these angles and so we do for φ, φ' and finding delta phi. After getting these angles we can rebuild the transformation matrix and then apply the transformation by multiplying each vertex in the mesh "3D model" with this transformation matrix.

The computation of these angles are discussed later in this chapter in section 3.3.3 Computing the transformation matrix and applying the transformation.

The results of 2D-3D Registration algorithm are shown in the next figure 3.4:

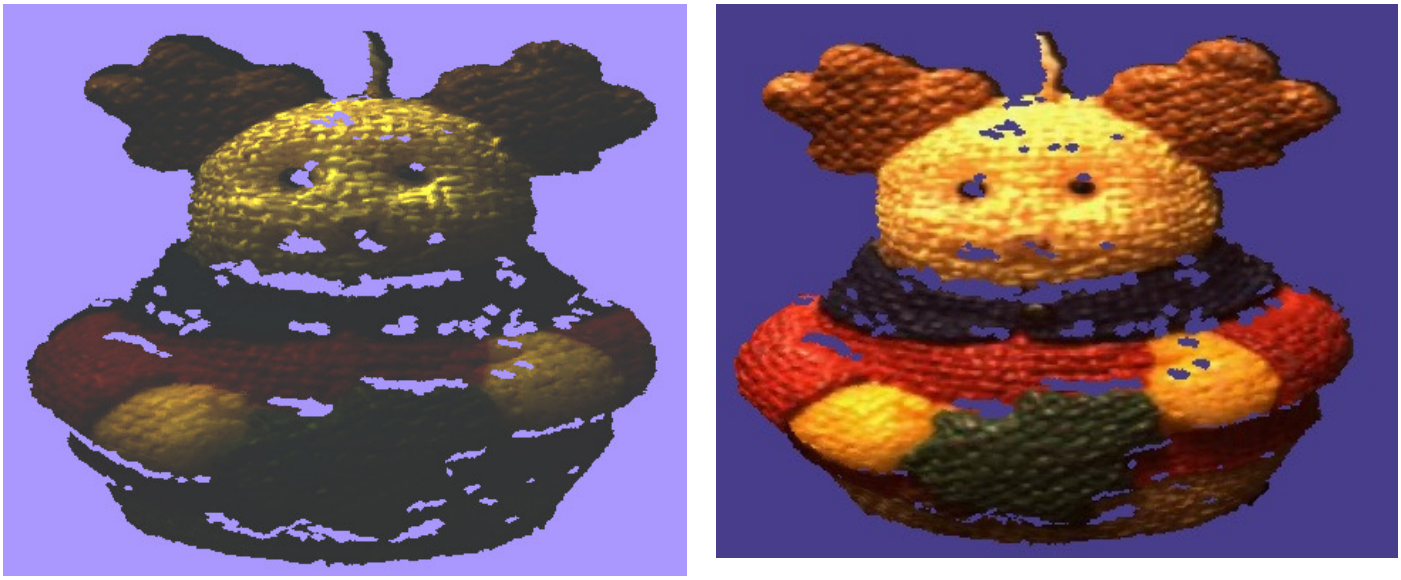

Figure 3.4: ***this figure shows spectral image registered to the model on the left in compare with RGB scanner image registered to the mesh. The color differences are due the bad lightning when the spectral image was taken however the spectral texture was successfully registered to the mesh as shown in the pictures.***

The holes in the two meshes are during the 3D scanning process and have nothing to do with registration process. And the small difference in the geometry is due the angle from which the screen shot was taken since these two shots have been taking from different views.

For more information on 3D projection and transformation see [28].

3.3 The 2D-3D Registration Algorithm

The proposed algorithm consists of the following parts:

1- Image preprocessing and enhancement.
2- Image registration: feature points detection and correspondence points.
3- Computing the transformation matrix and applying the transformation.

The following diagram "figure 3.5" shows algorithm workflow and how all the parts interact with each other:

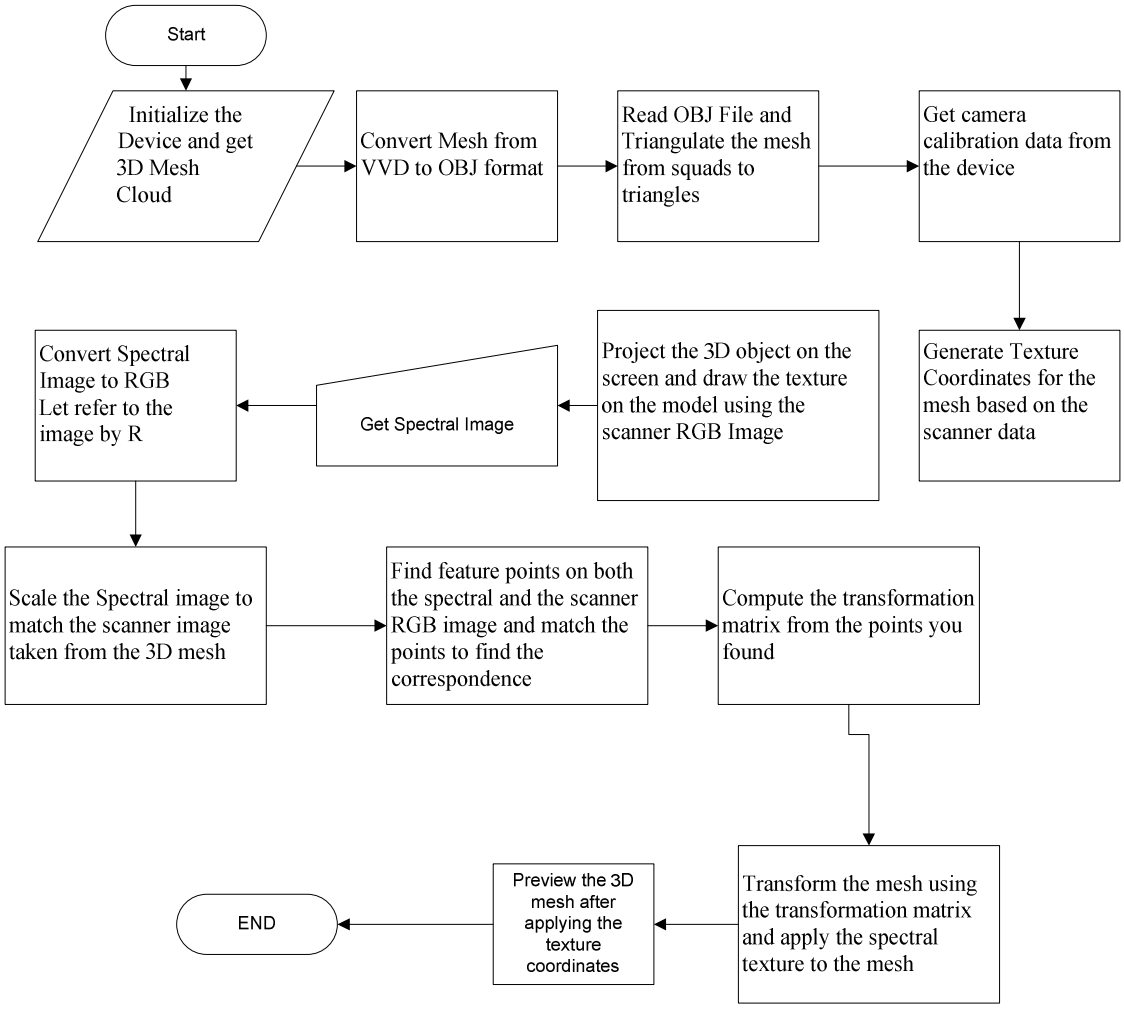

Figure 3.5: show the interaction between the system parts.

3.3.1 Image preprocessing and enhancement

First to be able to use the enhancement methods we need the two images to be scaled to match each other. We already assumed that the transformation needed is only rotation not scaling, so we had to scale one image to another. To do this, a drawing area detection algorithm was proposed.

The algorithm for finding the drawing area of the images is as follow:

1- We have to use a background behind the image consisted of one color, this can be easily done using a black paper behind the model when taking the pictures.
2- Then we have to remove that back color from the image using threshold which can be done in this algorithm:

```
Detect first pixel color and set the back color=first pixel color
Foreach pixel p in Image I
    If p.Color – back color=0 or threshold then
        p.Color=White
    end if
end foreach
```

by doing this we remove these colors from the background of both images.
3- Now search in columns order until you first hit the non white colored pixel and save its x coordinate
4- Search in rows order until you first hit the non white colored pixel and save it as y coordinate
5- Serach in columns order but start from the last column to first column and save the first non white colored pixel
6- Search in rows order but start from the last row to the first until you find the first non white colored pixel
7- Construct a rectangle for both of the images around the detected drawing area.
8- Find the width and height ratio differences between the two rectangles and scale the target rectangle to match the reference rectangle

9- Initialize a new image with the size of the reference image and align the target rectangle that holds the target drawing area in the same position where the reference rectangle left top corner is.

10- Pass the image after applying gamma correction function on the image to the feature points detection algorithm

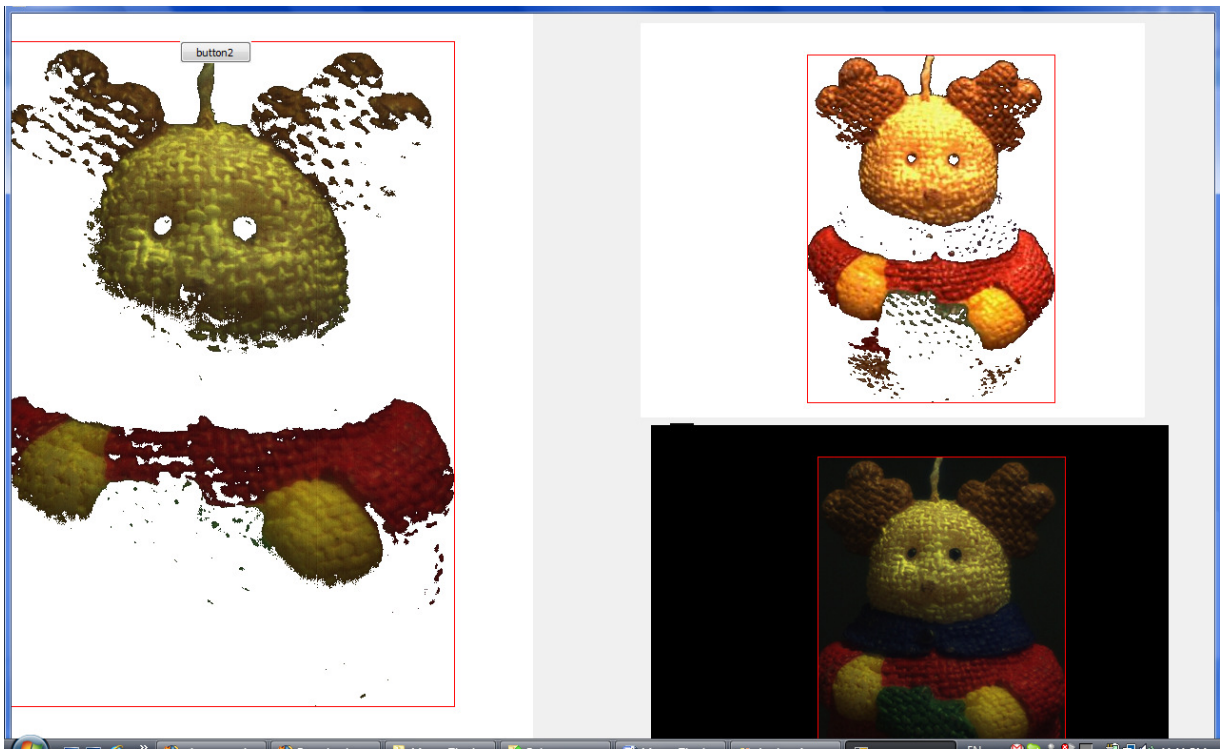

Figure 3.6: Shows the scaling and gamma correction in action where on the left is detected area of the spectral image "the target image" and on the right is the detected area of the reference image "the scanner image", on the right bottom is the resulted scaled image after the aligning and gamma correction and its ready to be passed to the image registration algorithm.

Note*: for the code of this part of the algorithm please refer to the Developer Manual.*

3.3.2 Image registration: feature points detection and correspondence points

The feature points' detection is the most crucial part of the algorithm. To find the feature points a new algorithm was proposed and it's based on the intensity of the image colors.

The algorithm takes each pixel in the image and selects the neighbors of this pixel by building a square around it. Then we sum the intensity values for all the pixels in this square, we also build similar squares around this square in all directions and sum each square intensity values.

Now assuming that I(x,y) is the centered pixel of the square then we get

$$s_{center} = \sum_{i=x,j=y}^{m+x,m+y} I(i,j) \tag{3.21}$$

Where s is the summation of this square and m is the square dimension

$$s_{Neighbor\ Square} = \sum_{1}^{m} I_{Neighbor\ Square}(i,j) \tag{3.22}$$

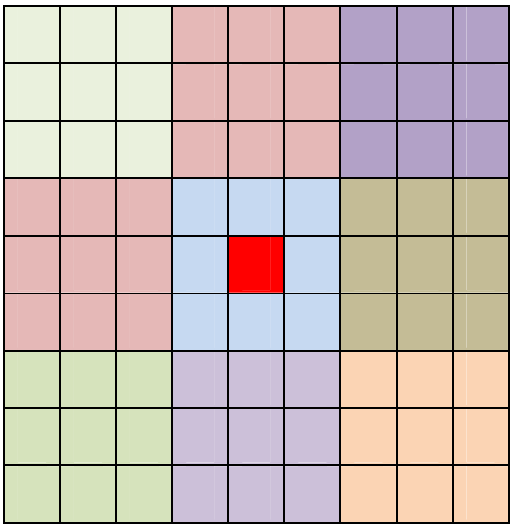

Figure 3.7: this figure shows the start pixel with its square surrounding it, it also shows the surrounding squares.

After we got the sum of all squares surrounding the start pixel square we divide the summed value of the start square s_{center} with each of the surrounding squares $s_{neighbor\ square}$,

$D = \frac{S_{center}}{S_{Neighbor\ Square}}$ (3.23) If D is larger than threshold for all the surrounding squares then we mark this square s_{center} as a feature point, by marking the center pixel. We then assign a value

for this pixel in relative to its neighbors' values. The threshold is selected by given a start up value and increasing this value until we end up with only three highest D values.

We repeat that operation for all the pixels in the image, finally we save the candidates feature points.

We do that operation for both of the reference and target images. The last step in the operation is to establish correspondence between the reference feature points and the target feature points.

The correspondence is established through the closest match between both the Euclidean distance and assigned values for each of the feature points detected. The feature point that has the closest Euclidean distant plus the closest value in the assigned values for the feature points is considered the correspondence of a point.

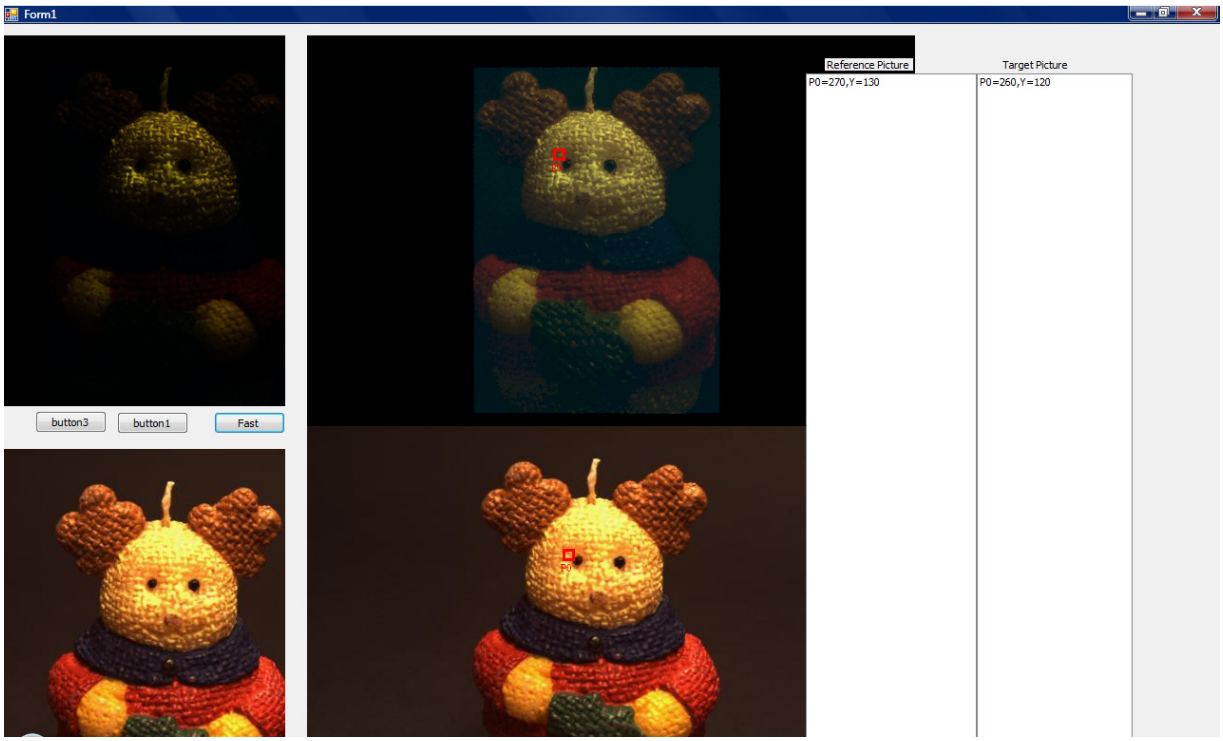

Figure 3.8: shows the algorithm results after finding a feature point and its correspondence in the both target and reference images, however in this illustration the requested number of points to be found is one, since that all what we need for the registration.

Now after we found the feature point and its correspondence we can compute the transformation matrix and then apply the transformation.

Note: *for the code of this algorithm please refer to the Developer Manual. For more information on Image Registration and feature points' detection see* [29][30][31]

3.3.3 Computing the transformation matrix and applying the transformation

In section 3.2 of this chapter we discussed the mathematical concept behind 2D-3D registration algorithm. Here we will discuss the computation of the angles, so let us take a look at the following figure 3.9:

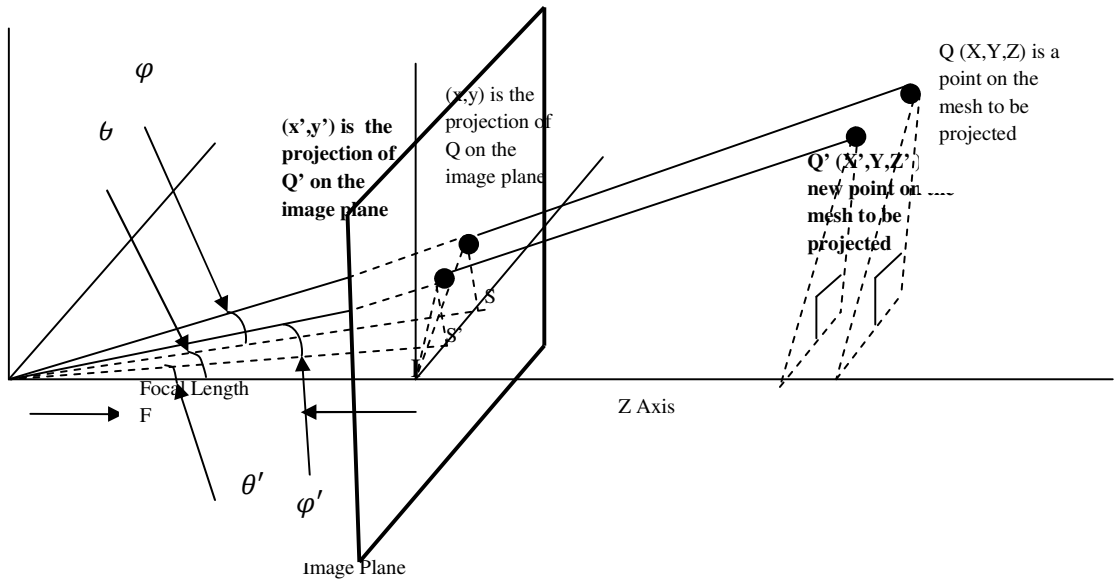

Figure 3.9: Angles calculation in merged image plans through perspective projection

Now to find the angles let us first define the main points:

1- I is the image plane point that reside on the optical axis which is aligned with the Z axis so its equal to (0,0,Oz-f).
2- s is equivalent to (x,0,Oz-f)
3- s' is equivalent to (x',0,Oz-f)
4- O is the viewpoint and its (0,0,Oz)
5- p is the reference feature point
6- p' is the target correspondent point to p.

Now we compute the angles:

First we compute the distance between I-s and I-O then we compute the distance between I-s'. So we have now the following triangles I s O and I s' O, we know that I-s is perpendicular on I-O and so the side s-O is the hypotenuse of the triangle. And so for the triangle I s' O we do the same.

$$Is = \sqrt{(x-0)^2 + (0-0)^2 + ((Oz-f)-(Oz-f))^2} \quad (3.24)$$

$$Is' = \sqrt{(x'-0)^2 + (0-0)^2 + \big((Oz-f)-(Oz-f)\big)^2} \quad (3.25)$$

$$IO = \sqrt{(0-0)^2 + (0-0)^2 + (Oz-(Oz-f))^2} \quad (3.26)$$

$$Os = \sqrt{Is^2 + IO^2} \quad (3.27)$$

$$Os' = \sqrt{Is'^2 + IO^2} \quad (3.28)$$

Now we can get theta by trigonometric functions as follow:

$$\sin\theta = \frac{Is}{Os} \quad (3.29)$$

$$\sin\theta' = \frac{Is'}{Os'} \quad (3.30)$$

$$\Delta\theta = \theta - \theta' \quad (3.31)$$

Now we got delta theta we should find delta phi:

$$Ps = \sqrt{(x-x)^2 + (0-y)^2 + ((Oz-f)-(Oz-f))^2} \quad (3.32)$$

$$P's' = \sqrt{(x'-x')^2 + (0-y')^2 + \big((Oz-f)-(Oz-f)\big)^2} \quad (3.33)$$

$$Os = \sqrt{(0-x)^2 + (0-0)^2 + (Oz-(Oz-f))^2} \quad (3.34)$$

$$Os' = \sqrt{(0-x')^2 + (0-0)^2 + \big(Oz-(Oz-f)\big)^2} \quad (3.35)$$

$$Op = \sqrt{Ps^2 + Os^2} \tag{3.36}$$

$$Op' = \sqrt{Ps'^2 + Os'^2} \tag{3.37}$$

Now we get phi by trigonometric function:

$$\sin\varphi = \frac{Ps}{Op} \tag{3.38}$$

$$\sin\varphi' = \frac{Ps'}{Op'} \tag{3.39}$$

$$\Delta\varphi = \varphi - \varphi' \tag{3.40}$$

After getting the delta theta and delta phi now we can construct the transformation matrix:

$$T = \begin{bmatrix} \cos\Delta\theta & 0 & \sin\Delta\theta & 0 \\ -\sin\Delta\varphi\sin\Delta\theta & \cos\Delta\varphi & -\sin\Delta\varphi\cos\Delta\theta & 0 \\ -\sin\Delta\theta\cos\Delta\varphi & \sin\Delta\varphi & \cos\Delta\varphi\cos\Delta\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.41}$$

Now we can iterate through all the vertices in the mesh and multiply them by the transformation matrix $V'_{i=}V_i * T$ (3.42) .

To compute the texture coordinates we use the projection formula of the scanner which is derived from the perspective projection formula as the following:

$$x = \frac{f(V' - O)X}{(V' - O)Z} \tag{3.43}$$

$$y = \frac{f(V' - O)Y}{(V' - O)Z} \tag{3.44}$$

We got now the x y projection coordinate, so we have to convert them into raster u v coordinates by:

$$u = \frac{x}{px} + ox \tag{3.45}$$

$$v = \frac{-y}{py} + oy \tag{3.46}$$

Where px, py are the pichX, pitchY and the ox, oy are the centerX, and centerY respectively. Finally we have to scale these values between 0 and 1 so the DirectX can use them by:

$$u' = \frac{u}{image\ width} \tag{3.47}$$

$$v' = \frac{v}{image\ height} \tag{3.48}$$

The overall computation of applying the transformation and finding the texture coordinates in pseudo code:

Input: Mesh, Feature and Correspondent Point FP

T=compute the transformation matrix (FP)
For each vertex V in Mesh
{

V'=VT

$x = \frac{f(V\prime - O)X}{(V\prime - O)Z}$

$y = \frac{f(V\prime - O)Y}{(V\prime - O)Z}$

$u = \frac{x}{px} + ox$

$v = \frac{-y}{py} + oy$

$u' = \frac{u}{image\ width}$

$v' = \frac{v}{image\ height}$

V'.Texture Coordinate.u=u'

V'.Texture Coordinate.v=v'

}

Figure 3.10: the result of the algorithm shows the spectral image as a texture to the mesh

Note: *for the source code of this algorithm please refer to the Developer Manual. For more information on DirectX programming see* [32][33][34]

Chapter 4: 3D – 3D Registration

In this chapter we will discuss the process of registering one scan of the model to another scan of the model. The process of scanning the model is usually done by scanning the model from different angles then merges these scans into one complete model. We mean by registering one model to another is translating, rotating and scaling a model to merge one scan model with another. For example in the scanning process we usually take different scans of the same model to cover all the sides of the model as in the following figure 4.1:

Figure 4.1: Two 3D scans for the same model from different perspectives

The idea behind the registration process is to match one of the scans to the other to produce a complete model as in the following figure 4.2:

Figure 4.2: Registered model composed of more than one scan

4.1 problem formulation:

To achieve the registration between the two models we need to understand the concept in more mathematical way. So let us assume we have two vectors in 3D Euclidean sub space, and these two vectors exist in the world space which is 3D Euclidean system. We need to transform one of these points to overlay the other as in the following figure 4.3:

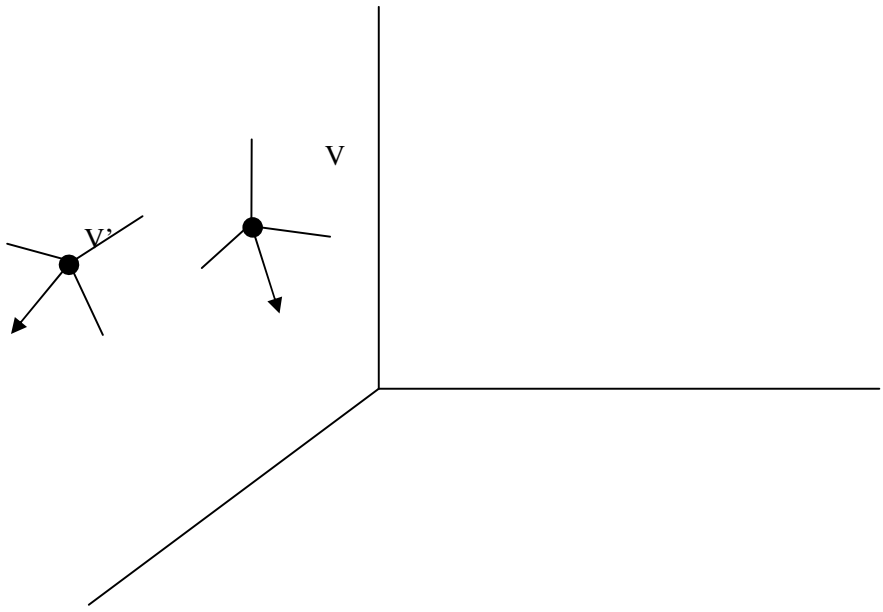

Figure 4.3: shows two vectors V and V' reside in the 3D Euclidean system, it also shows their sub spaces

The transformations needed to change vector V to overlay vector V' are translation, rotation and scaling and thus we need a transformation matrix of 4x4 to be multiplied with vector V to produce vector V'.

V'=VT where V is the vector in the target mesh and V' is the vector in the reference mesh and T is 4x4 transformation matrix. $T = \begin{bmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ T_{41} & T_{42} & T_{43} & T_{44} \end{bmatrix}$ (4.1)

4.2 Finding the transformation matrix:

To solve the 4x4 transformation matrix T let us multiply the vector V by the transformation matrix T; so we will get:

$$V' = VT \tag{4.2}$$

$$[X \quad Y \quad Z \quad 1]\, x \begin{bmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ T_{41} & T_{42} & T_{43} & T_{44} \end{bmatrix} = [X' \quad Y' \quad Z' \quad 1] \tag{4.3}$$

$$XT_{11} + YT_{21} + ZT_{31} + T_{41} = X' \tag{4.4}$$

$$XT_{12} + YT_{22} + ZT_{32} + T_{42} = Y' \tag{4.5}$$

$$XT_{13} + YT_{23} + ZT_{33} + T_{43} = Z' \tag{4.6}$$

$$XT_{14} + YT_{24} + ZT_{34} + T_{44} = 1 \tag{4.7}$$

Now we got four linear equations with 16 unknowns assuming we already know V and V'. So what we need now is to get four vectors on each of the reference mesh and the target mesh and establishing the correspondence between all of these vectors.

So assuming we found the four vectors and their correspondence we get another 16 equations four equations for each four unknowns. However we solve now for the first four unknowns:

$$X_1T_{11} + Y_1T_{21} + Z_1T_{31} + T_{41} = X'_1 \tag{4.8}$$

$$X_2T_{11} + Y_2T_{21} + Z_2T_{31} + T_{41} = X'_2 \tag{4.9}$$

$$X_3T_{11} + Y_3T_{21} + Z_3T_{31} + T_{41} = X'_3 \tag{4.10}$$

$$X_4T_{11} + Y_4T_{21} + Z_4T_{31} + T_{41} = X'_4 \tag{4.11}$$

We will use gauss elimination method and by subtracting (4.8) from (4.9), (4.10) and (4.11)

$$T_{11}(X_1 - X_2) + T_{21}(Y_1 - Y_2) + T_{31}(Z_1 - Z_2) = X'_1 - X'_2 \tag{4.12}$$

$$T_{11}(X_1 - X_3) + T_{21}(Y_1 - Y_3) + T_{31}(Z_1 - Z_3) = X'_1 - X'_3 \quad (4.13)$$

$$T_{11}(X_1 - X_4) + T_{21}(Y_1 - Y_4) + T_{31}(Z_1 - Z_4) = X'_1 - X'_4 \quad (4.14)$$

Let us assume the following for easier reference:

$$a = (X_1 - X_2)\ ,\ b = (Y_1 - Y_2)\ ,\ c = (Z_1 - Z_2)\ ,\ d = X'_1 - X'_2,\ e = (X_1 - X_3),\ f = (Y_1 - Y_3) \quad (4.15)$$

$$g = (Z_1 - Z_3)\ ,\ h = X'_1 - X'_3\ ,\ i = (X_1 - X_4)\ ,\ j = (Y_1 - Y_4)\ ,\ k = (Z_1 - Z_4)\ ,\ l = X'_1 - X'_4 \quad (4.16)$$

Now by multiplying equation (4.12) by *a* and equation (4.13) by *e* and subtracting (4.13) from (4.12)

$$eaT_{11} + ebT_{21} + ecT_{31} = ed \quad (4.17)$$

$$-aeT_{11} - afT_{21} - agT_{31} = -ah \quad (4.18)$$

$$T_{21}(eb - af) + T_{31}(ec - ag) = ed - ah \quad (4.19)$$

Now multiply equation (4.12) by *i* and equation (4.14) by *a* and subtract (4.14) from (4.12)

$$iaT_{11} + ibT_{21} + icT_{31} = id \quad (4.20)$$

$$-aiT_{11} - ijT_{21} - ikT_{31} = -il \quad (4.21)$$

$$T_{21}(ib - ij) + T_{31}(ic - ik) = id - il \quad (4.22)$$

Let's assume that:

$$o = (eb - af), p = (ec - ag), q = ed - ah,\ r = (ib - ij), s = (ic - ik),\ t = id - il \quad (4.23)$$

And by multiplying equation (4.19) by *r* and equation (4.22) by *o* and subtracting (4.22) from (4.19) we get:

$$roT_{21} + rpT_{31} = rq \quad (4.23)$$

$$-orT_{21} - osT_{31} = -ot \quad (4.24)$$

Finally we get that:

$$T_{31} = \frac{rq - ot}{rp - os} \tag{4.25}$$

And now by back substituting T_{31}in (4.22) we get that:

$$oT_{21} + \frac{rq - ot}{rp - os} = t \tag{4.26}$$

$$T_{21} = \frac{otrp - to^2s - orq + o^2t}{rp - os} \tag{4.27}$$

Now we continue the back substituting T_{21} and T_{31}in (4.14) we get:

$$iT_{11} + j\frac{otrp - to^2s - orq + o^2t}{rp - os} + k\frac{rq - ot}{rp - os} = l \tag{4.28}$$

$$iT_{11} + \frac{jotrp - jto^2s - jorq + jo^2 + krq - kot}{rp - os} = l \tag{4.29}$$

$$T_{11} = \frac{ilrp - ilos - ijotrp + ijto^2s + ijorq - ijo^2 - ikrq + ikot}{rp - os} \tag{4.30}$$

Finally we get the T_{41} by substituting T_{11}, T_{21} and T_{31}in equation (4.8):

$$X_1\frac{ilrp - ilos - ijotrp + ijto^2s + ijorq - ijo^2 - ikrq + ikot}{rp - os} + Y_1\frac{otrp - to^2s - orq + o^2t}{rp - os} + Z_1\frac{rq - ot}{rp - os} + T_{41} = X_1' \tag{4.31}$$

$$T_{41} = \frac{X_1'rp - X_1'os - X_1ilrp + X_1ilos + X_1ijotrp - X_1ijto^2s - X_1ijorq + X_1ijo^2 + X_1ikrq - X_1ikot - Y_1otrp + Y_1to^2s + Y_1orq - Y_1o^2t - Z_1rq + Z_1ot}{rp - os} \tag{4.32}$$

Now we have solved the first system of equations and we got the first four unknowns and so we will have to solve for the next 12 equations left.

However to make it easier for us we implemented a function to solve these 16 equations. The function depends on our solution of the first four equations:

So the function takes 4x4 matrix of coefficients of the same four unknowns and a 1x4 vector containing the right side of the four equations:

```
SolveLinearEquation(Matrix m,Vector4 v)
```

The function returns 1x4 vector containing the value of the four unknowns.

Now we can use this function to solve our equations :

```
m.M11 = refVector1.X;
m.M12 = refVector1.Y;
m.M13 = refVector1.Z;
m.M14 = 1;

m.M21 = refVector2.X;
m.M22 = refVector2.Y;
m.M23 = refVector2.Z;
m.M24 = 1;

m.M31 = refVector3.X;
m.M32 = refVector3.Y;
m.M33 = refVector3.Z;
m.M34 = 1;

m.M41 = refVector4.X;
m.M42 = refVector4.Y;
m.M43 = refVector4.Z;
m.M44 = 1;

//Solve for X
Vector4 vec = new Vector4();
vec.X = tarVector1.X;
vec.Y = tarVector2.X;
vec.Z = tarVector3.X;
vec.W = tarVector4.X;
```

```
Vector4 col1 =SolveLinearEquation(m,vec);

//Solve for Y
vec = new Vector4();
vec.X = tarVector1.Y;
vec.Y = tarVector2.Y;
vec.Z = tarVector3.Y;
vec.W = tarVector4.Y;

Vector4 col2 = SolveLinearEquation(m, vec);

//Solve for Z
vec = new Vector4();
vec.X = tarVector1.Z;
vec.Y = tarVector2.Z;
vec.Z = tarVector3.Z;
vec.W = tarVector4.Z;

Vector4 col3 = SolveLinearEquation(m, vec);

//Solve for W
vec = new Vector4();
vec.X = 1;
vec.Y = 1;
vec.Z = 1;
vec.W = 1;

Vector4 col4 = SolveLinearEquation(m, vec);
Matrix T = new Matrix();
// we finally here fill the transformation matrix T
```

```
T.M11 = col1.X;
T.M21 = col1.Y;
T.M31 = col1.Z;
T.M41 = col1.W;

T.M12 = col2.X;
T.M22 = col2.Y;
T.M32 = col2.Z;
T.M42 = col2.W;

T.M13 = col3.X;
T.M23 = col3.Y;
T.M33 = col3.Z;
T.M43 = col3.W;

T.M14 = col4.X;
T.M24 = col4.Y;
T.M34 = col4.Z;
T.M44 = col4.W;
```

In the previous code refVector refers to V and tarVector refers to V' and the numbers following the name is the number of the selected vectors. However now we got the transformation matrix T all what we need to do is multiplying eavh vertex or vector in the mesh with T to transform the target mesh to the reference mesh.

Figure 4.4: Results after computing and applying the transformation matrix on two unregistered meshes

4.3 Acquiring the reference and target vectors

To find reference and target vectors on meshes we let the user select them by choosing four points on both of the images as shown in the following figure 4.5:

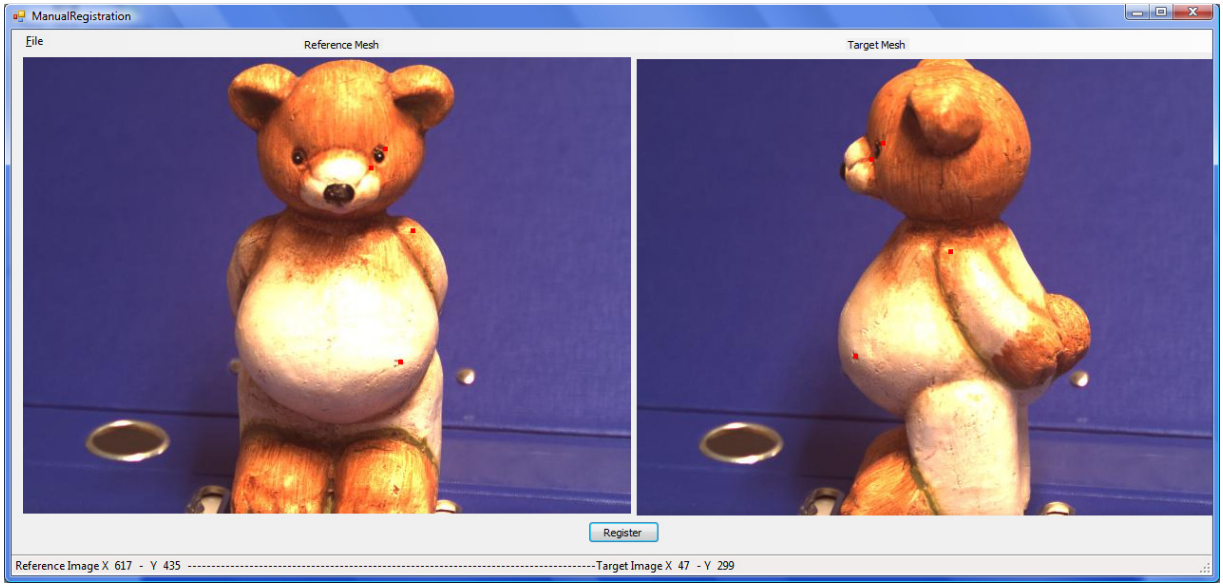

Figure 4.5: shows the selection tool where user can select point on both target and reference meshes.

The user chooses the four points by using the mouse and clicking on the area that has a potential feature point and then initiates the registration method. We then search for the vectors for which their texture coordinates equivalent to those that the user chose. We search for these vectors in the mesh files and their saved or computed texture coordinates.

We save the vectors for those points and start computing the transformation matrix as discussed in section 4.2 of this chapter.

Chapter 5: Data files, Mesh triangulation, Lightning System and other features.

In this chapter we will talk about the data files and their conversions, the triangulation of the mesh, the lightning and other feature portrayed in the software.

5.1 Data files:

As we mentioned earlier in chapter 1 the Vivid9i provide us with the mesh cloud for the real model. The mesh is saved by default in .CDK format Camera Data files these files contain one or more scans of the same object as well as the texture images taken by the scanner RGB camera.

In order to use these files in DirectX we had to convert these files into different formats where we can manipulate them to be finally used with DirectX. The CDK format was introduced by Konica Minolta Company and it is a customized format for their softwares. However the CDK format is not widely supported by other 3D softwares.

The Vivid9i SDK has a conversion tools that can convert the CDK format into another type of format such as .VVD which is also from Konica Minolta Company. Yet this format comes with one significant difference from CDK format; which supports the access to the internal data and thus allowing us to access the vertex data.

Getting the vertex data means we can retrieve the mesh and of course convert it to any format we wish to have. The format chosen for this software was Wavefront .OBJ files. These files are plain text files, and they are very straight forward to deal with.

However the vertex data is not the only data we need from Vivid9i scanner. We also need the texture coordinates and the faces vertices. One drawback about Vivid9i SDK is that it doesn't directly give you the texture coordinates. The SDK only gives the camera parameters that will help you computing the texture coordinates.

To compute these coordinates we have to follow the scanner projection formula which is mentioned in their SDK help file. The scanner SDK however has a very poor documentation and

it took a considerable amount of time to solve its true functionality. The SDK is written in native C and C++ libraries, with very poor comments on its functionality or use.

Now to convert from .VVD files into .OBJ file we need the following information to be computed from the VVD files:

1- The vertex data.
2- Texture coordinates.
3- Faces vertices
4- Camera parameters.

The SDK helps in finding the vertex data, faces and camera parameters. So getting these data allows us to construct .OBJ files. The file format of the .OBJ file is like the following:

1- # refers to a comment line that will be usually ignored by the 3D softwares. We used this comment to add the camera parameters data, and read it later from the software and compute the texture coordinates.
2- V followed by x y z coordinates; this V refers to vertex and it holds the x y z coordinates of the vertex data.
3- VT u v [w] refers to the texture coordinates which was not used in the conversion since we left computing the texture coordinates outside the file conversion, for easier manipulation later in the main software.
4- VN followed by z y z is the vertex normal and it is used to compute the light reflection on the mesh surface. And this also wasn't used in the file conversion since we wanted to compute this in the main software.
5- F V1/VT1/VN1 … Vn/VTn/VNn is the face or in other terms is the polygon. The number followed each of these commands such as V, VT and VN is the number of the vertex in according to its appearance in the file.

The output .OBJ file will be like this:

```
#height 480
#width 640
#sx 0.007400
```

```
#sy 0.007400
#f 25.572695
#ox 320.000000
#oy 240.000000
#Z 0.000000 0.000000 -1.000000
#O 0.000000 0.000000 14.812405
#Y 0.000000 1.000000 0.000000
v -5.557044 58.119133 -944.354937
v -5.832908 57.822115 -944.033533
v -5.553076 57.799996 -943.666385
v -5.274394 57.788145 -943.469538
f 1 2 3 4
f 2 8 9 23
f 2 9 3 65
```

5.2 Triangulating the mesh

Polygons in DirectX are processed in terms of triangles. To draw a mesh we need to divide the mesh in many connected triangles to form the model. As in the following figure 5.1:

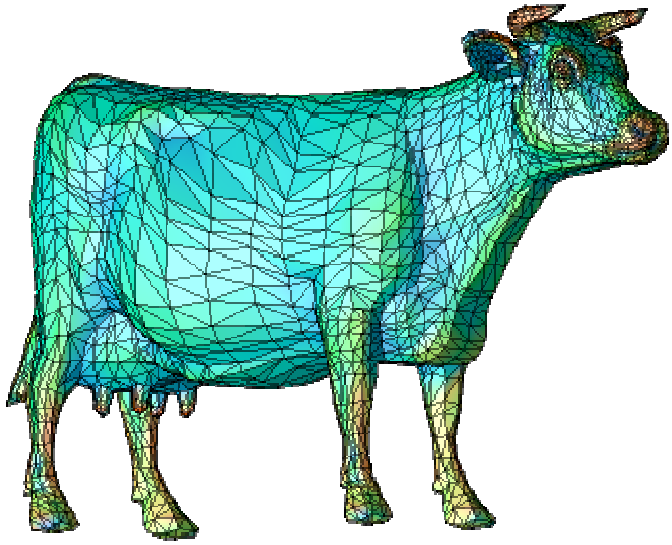

Figure 5.11: Triangulated mesh

However the Vivid9i gives us a polygon with squads and triangles; and thus will lead in errors while drawing the mesh as in the following figure 5.2:

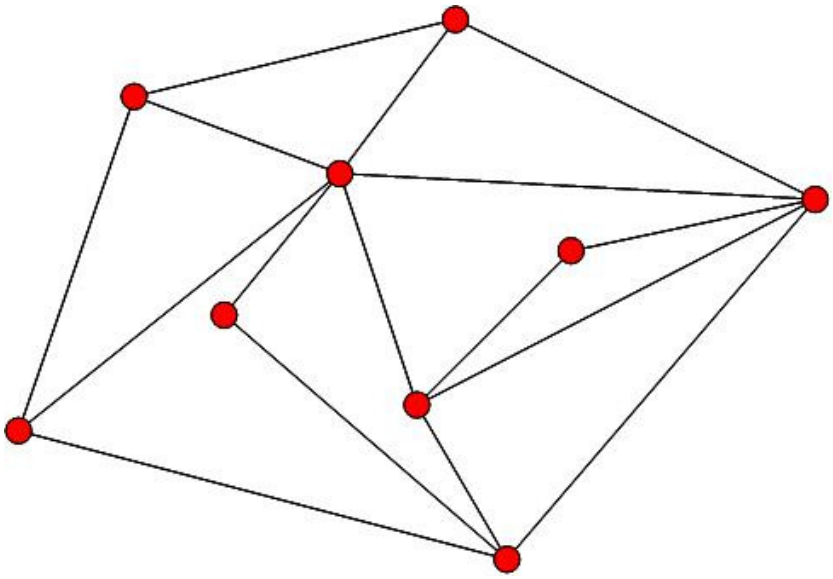

Figure 5.22: Shows triangles and squads obtained from the Vivid9i scanner.

To solve this problem we need to triangulate the mesh; so let us assume we have the following 9 vertices as in figure 5.3.

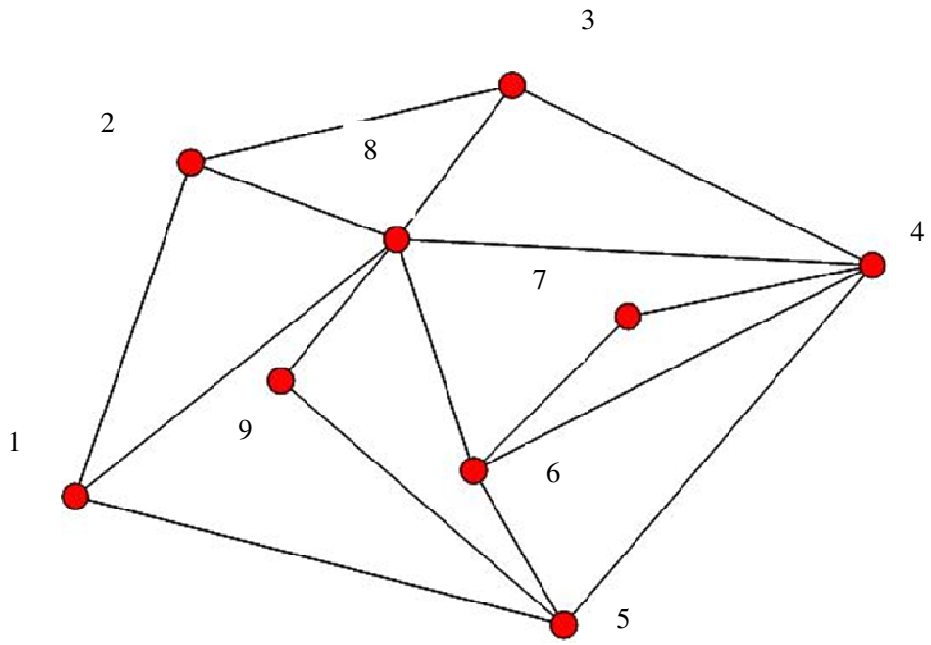

Figure 5.3: Shows a number of vertices on the mesh

So the faces of this mesh will be like the following:

F 1 2 8
F 2 8 3
F 3 4 8
F 4 5 6
F 6 7 4
F 4 8 7 6
F 5 6 9 8
F 5 9 1 8

Now to triangulate these squads we have to divide each squad into two triangles, so we do the following:

1- We connect the first 3 vertices in the squad; for example the polygon F 4 8 7 6 will be F 4 8 7.
2- Then we repeat the polygon and connect the last 3 vertices so it will be F 8 7 6.
   So the final result of F 4 8 7 6 will be equal to F 4 8 7 and F 8 7 6

And so on we do for all the squads in the mesh, finally the faces in the previous example will be like this:

F 1 2 8
F 2 8 3
F 3 4 8
F 4 5 6
F 6 7 4
F 4 8 7
F 8 7 6
F 5 6 9
F 6 9 8
F 5 9 1
F 9 1 8

Finally we obtain the triangulated mesh as in figure 5.4.

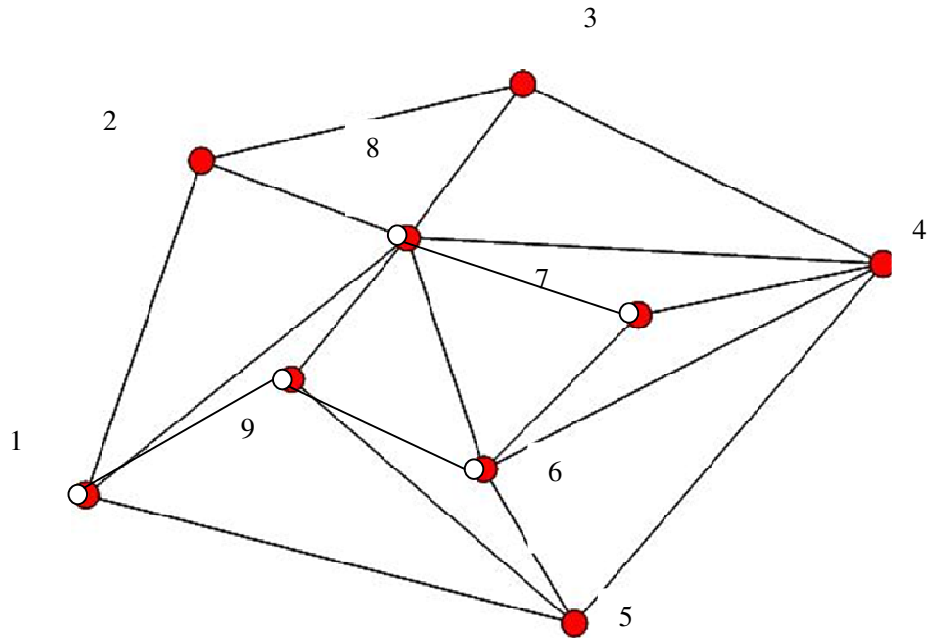

Figure 3: Show the mesh after Triangulation

5.3 Lighting System

To simulate the lights in the real world we used lights' physics to compute the light direction and reflection on the surface of the mesh. This section is based on the DirectX SDK Documentation [32]

5.3.1 Computing the surface normal

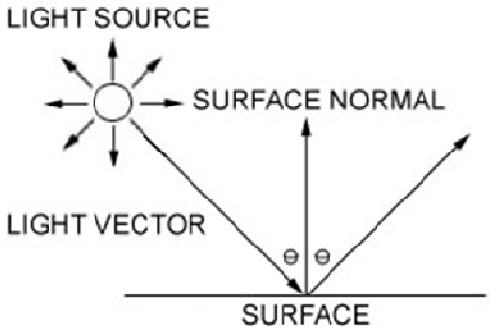

Figure 5.5: lights reflection on the polygon surface.

To simulate the light we need to compute the face normal. And to do so we need to compute the vertex normal to all the shared vertices.

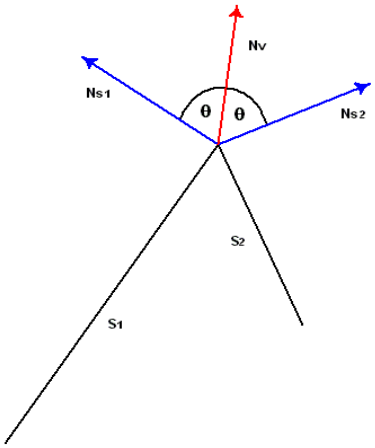

Figure 5.6: shows two surfaces S1 and S2 sharing a vertex where Ns1 and Ns2 are the normal of the vertex with respect of the surface S1 and S2. The Nv is the normal of the shared vertex and it has the same angle between Ns1 and Ns2.

Now we have computed the normal of the vertices as shown in the following figure 5.7 we then need to compute the face normal.

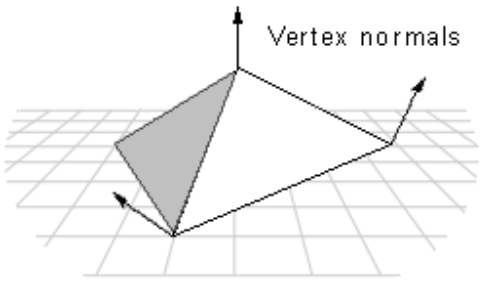

Figure 5.7: Vertices normal computed for the mesh

To compute the face normal we need to compute cross product of the two vertices on the triangle.

V1 x V2=N (5.1)

Finally we get the normal of face as shown in the following figure 5.8:

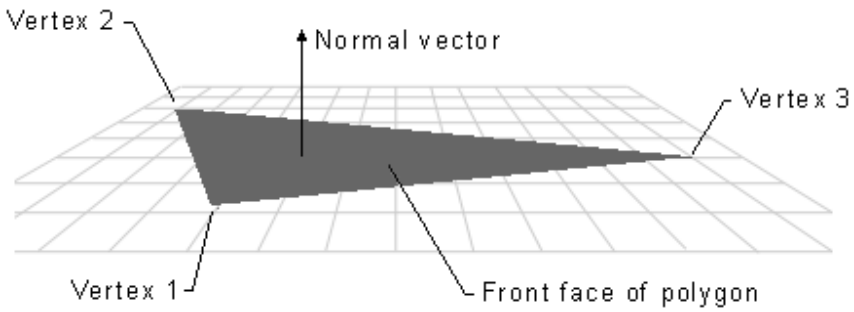

Figure 5.8: Shows the Normal of the face on the triangle polygon.

Finally the DirectX calculates the color and intensity values for the vertices and interpolates them for every point across the surfaces.

5.3.2 Light sources

The lights used in this project are two types of lights. The first is all Directional Light and the second is Spot Light.

The all Directional light is a light that came from a very far place and now it hits the mesh surface almost in parallel lines as in figure 5.9:

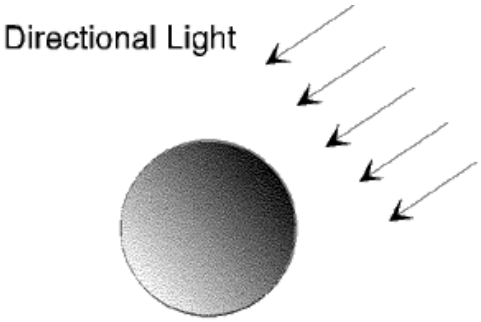

Figure 5.9: Shows all directional lights hit in parallel the surface of the mesh

Figure 5.10: Show the mesh after applying white all directional light to it

The spot light is a light that illuminate from a specific location to a specific location as in the following figure 5.11:

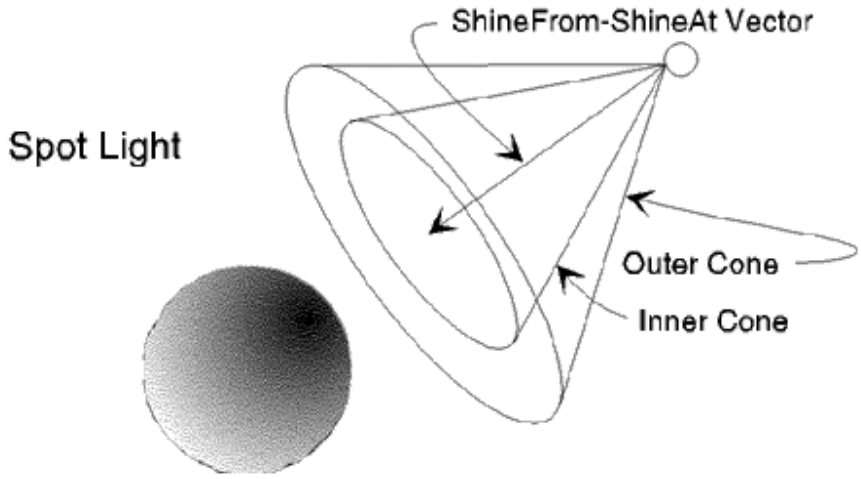

Figure 5.11: Show the spot light on the mesh, where we can the outer and inner cone of the spot light.

5.4 other features

In the 3D imaging system many features have been added to enrich functionality of the system. These functionalities will be discussed in this section and we will show the mathematical concepts behind these functionalities as well as their use.

5.4.1 Model manipulation

In order to make the system more flexible we added model manipulation features. Such features are scaling, rotation and translating. These features help the user to investigate the model from all the direction as well as investigating the different effects on the model.

5.4.1.1 Scaling the model

The system supports two types of scaling:

1- Uniform scaling where the model is scaled with the same factor in all directions
2- Directional scaling where the model is scaled in one axis or more by some factor different for each Axis.

To scale the model we need 4x4 Matrix containing the scaling factors vector F(Fx,Fy,Fz,1), then multiplying this scaling Matrix S by each vertex in the model. The use of the 4x4 matrix is due the need of translation later in other features.

$$S = \begin{bmatrix} F_x & 0 & 0 & 0 \\ 0 & F_y & 0 & 0 \\ 0 & 0 & F_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$V'_{scaled} = SV \text{ , where } V \text{ is a Vertex and } S \text{ is the scaling matrix} \qquad (5.2)$$

$$\begin{bmatrix} XF_x \\ YF_y \\ ZF_z \\ 1 \end{bmatrix} = \begin{bmatrix} F_x & 0 & 0 & 0 \\ 0 & F_y & 0 & 0 \\ 0 & 0 & F_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

To make the scaling uniform we set the Fx, Fy and Fz to an equal values; and thus will make the model be scaled in all the directions with the same amount. However if we wanted a directional scaling we set at least one of these factors to a different value.

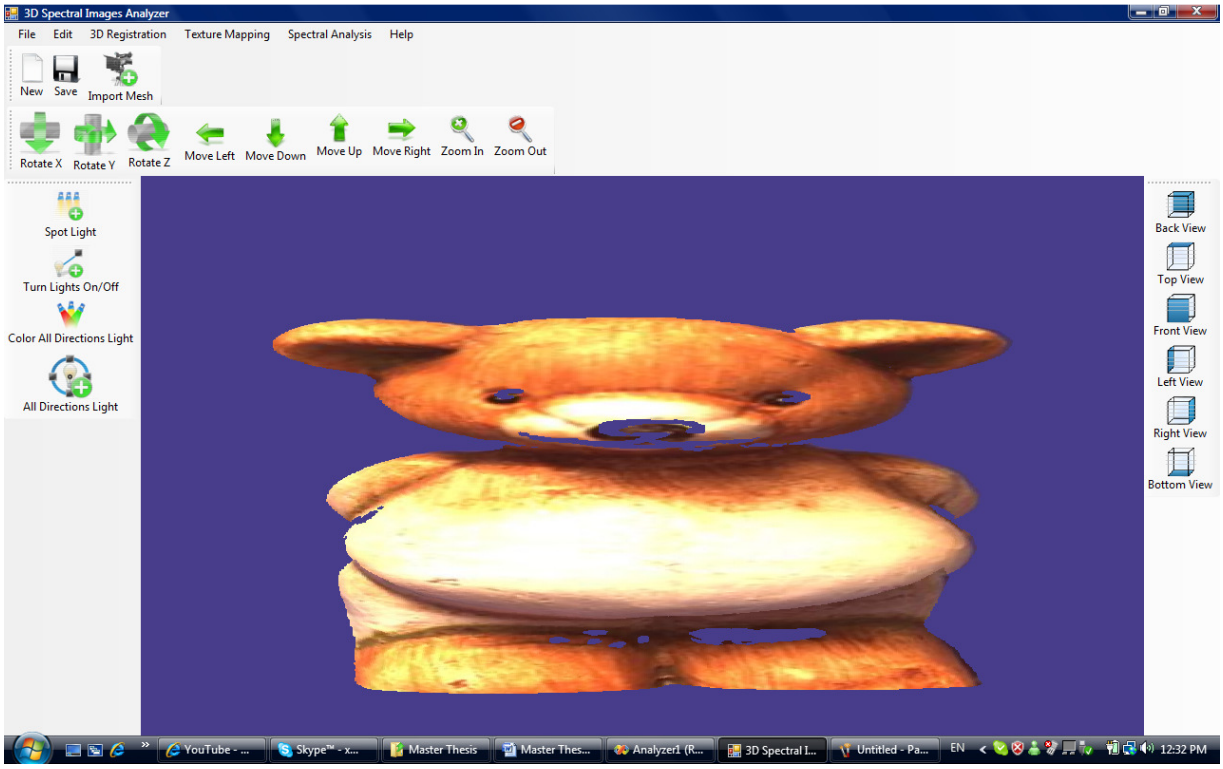

Figure 5.12: Shows the mesh after applying directional scaling along the X axis

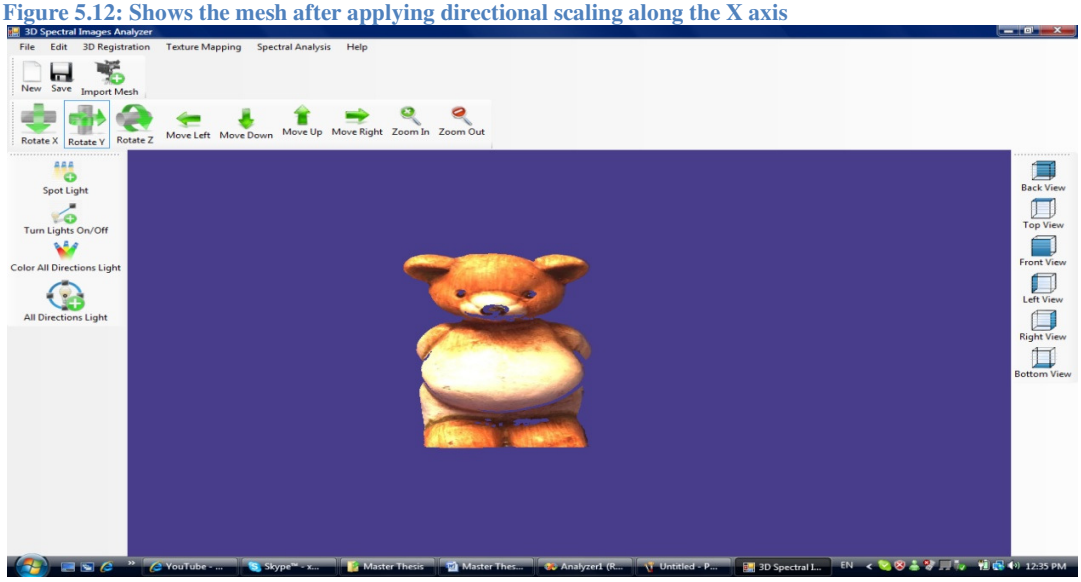

Figure 5.13: the model after applying uniformed scaling

5.4.2 Rotating the model

To rotate the model in the scene we can either rotate every vertex in the model where this will rotate the whole model or we can rotate the world where the model resides. So in this case of rotation we used the world transformation to rotate the model.

DirectX define a scene by creating a world which is infinite 3D Euclidean coordinate where all the models reside. Each of these models has its subspace inside this world coordinate. So if we wanted to transform one model by itself we will use the model subspace. But however if we want to rotate the whole scene we will use the world coordinates.

That is said we can define the rotation matrix R which is the rotation of Rx, Ry and Rz. To rotate the world we multiply the world matrix by R as in the following figure 5.14:

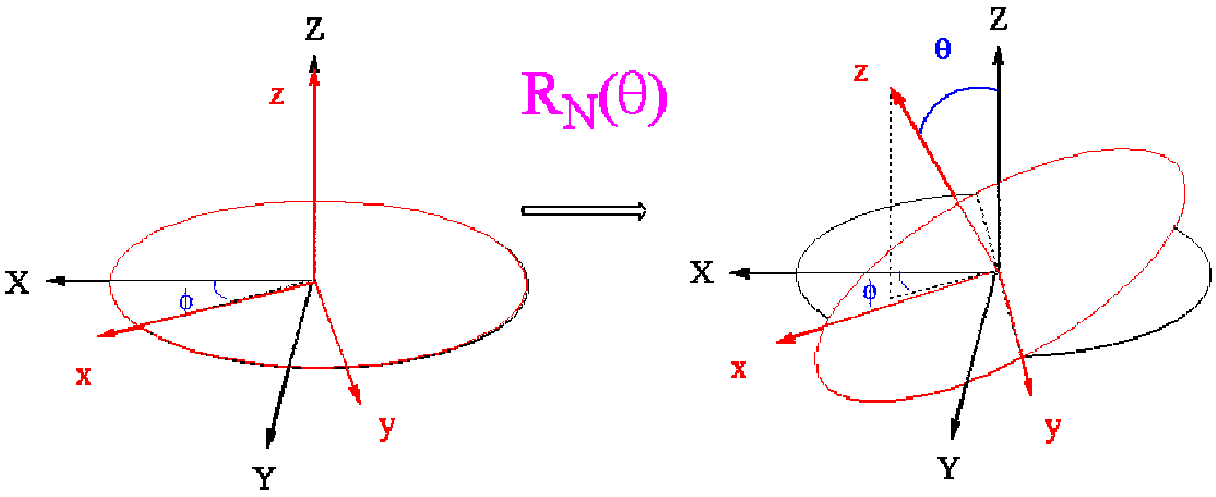

Figure 5.14: Shows the Rotation effects on the world coordinates of the main scene.

The transformation matrix R is equal to Rx*Ry*Rz (5.3):

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x \begin{bmatrix} \cos\psi & -sin\psi & 0 & 0 \\ sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now we need three angles to compute the rotation of the world matrix, these three angles can be given by user interaction by using the mouse. However after obtaining the rotations angles we multiply the world matrix by R to get:$W' = WxR$ (5.6)

$$W' = \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix} X \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ R_{21} & R_{22} & R_{23} & R_{24} \\ R_{31} & R_{32} & R_{33} & R_{34} \\ R_{41} & R_{42} & R_{43} & R_{44} \end{bmatrix}$$

Now we managed to rotate the world matrix but if we wanted to scale the world itself instead of the model; then we should concatenate the R and S matrices together and also we should keep the value of the World matrix saved so we achieve this by:

$$W'$$

$$= \begin{bmatrix} W'_{11} & W'_{12} & W'_{13} & W'_{14} \\ W'_{21} & W'_{22} & W'_{23} & W'_{24} \\ W'_{31} & W'_{32} & W'_{33} & W'_{34} \\ W'_{41} & W_{,42} & W'_{43} & W'_{44} \end{bmatrix} x \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix} X \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ R_{21} & R_{22} & R_{23} & R_{24} \\ R_{31} & R_{32} & R_{33} & R_{34} \\ R_{41} & R_{42} & R_{43} & R_{44} \end{bmatrix} x \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

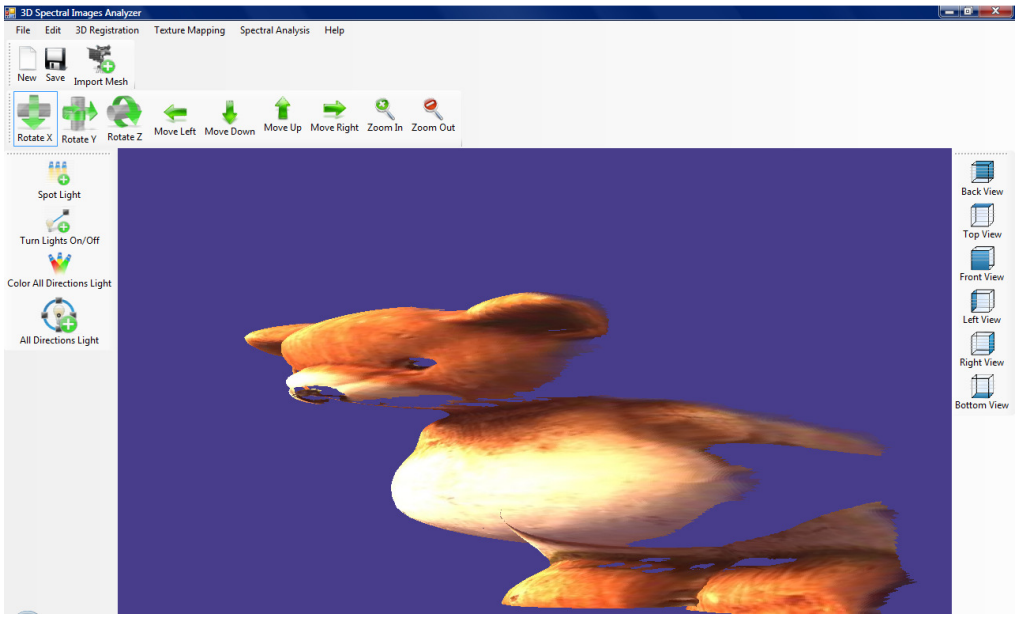

Figure 5.15: Shows the model after apply rotation and scaling to the world coordinate

5.4.3 Translating the model

The model translation is the process of repositioning every vertex in the model by a distance. This process can be achieved either by applying it on the model or on the world coordinates.

To translate the model or the world coordinate we need 4x4 Translation matrix, this matrix is then multiplied by the either the world matrix or mesh vertices.

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$W' = \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix} X \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.7)$$

Or by multiplying each vertex by the Translation matrix T

$$V' = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} x \begin{bmatrix} V_x \\ V_y \\ V_z \\ 1 \end{bmatrix} = \begin{bmatrix} V_x + T_x \\ V_y + T_y \\ V_z + T_Z \\ 1 \end{bmatrix} \quad (5.8)$$

Now to keep the other transformation we had so far we need to concatenate the translation matrix to the world matrix:

$$W' * = x \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix} X \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ R_{21} & R_{22} & R_{23} & R_{24} \\ R_{31} & R_{32} & R_{33} & R_{34} \\ R_{41} & R_{42} & R_{43} & R_{44} \end{bmatrix} x \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.9)$$

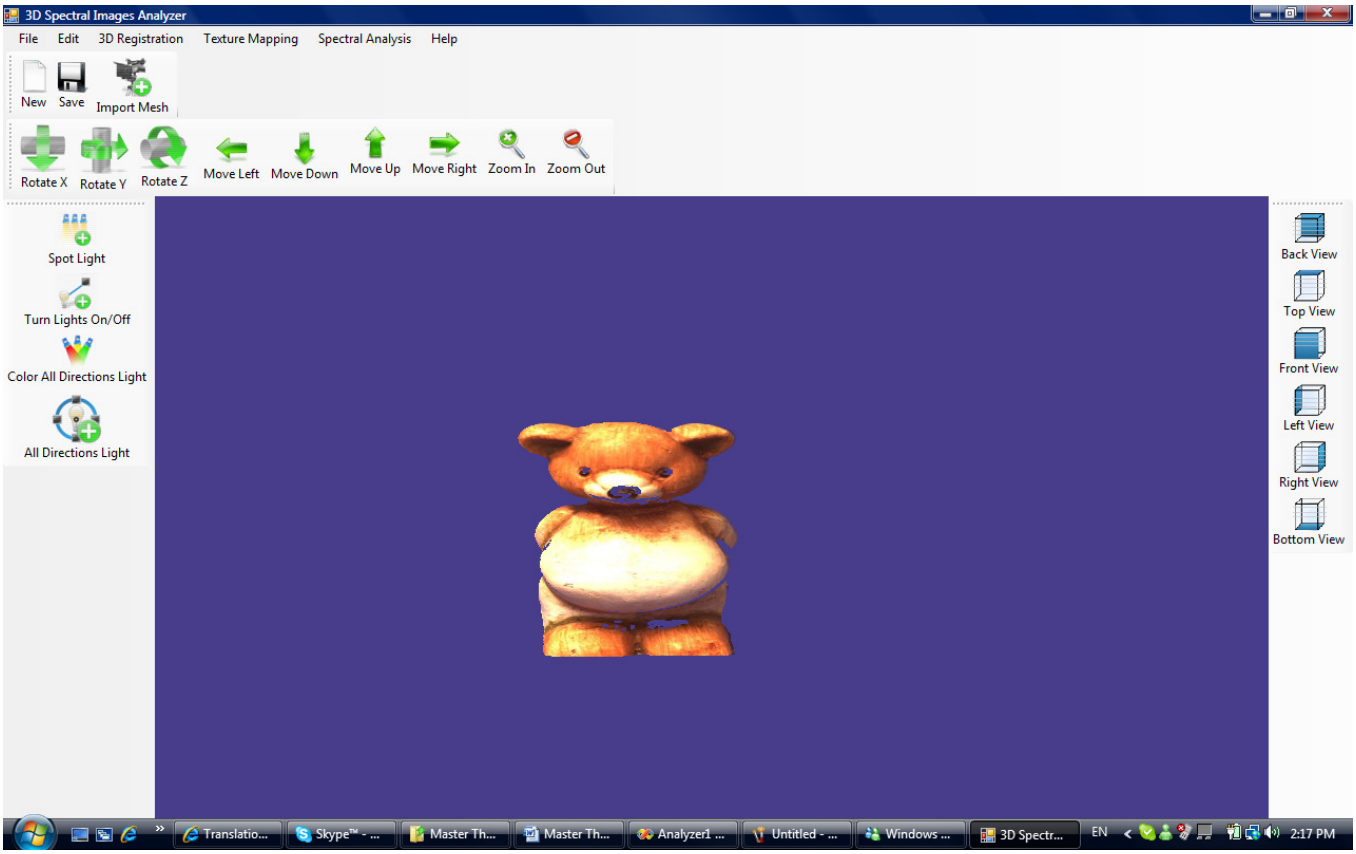

Figure 5.16: Shows the model after it has been translated.

5.4.4 Spectral Images &their tools

Spectral images are images where the reflectance light values are stored at different wavelengths, so for each pixel there is a number of channels that represent the same pixel at different wavelengths. The structure of the spectral images is like a 3D matrix where each layer represents the light at different wavelengths as in the following figure 5.17:

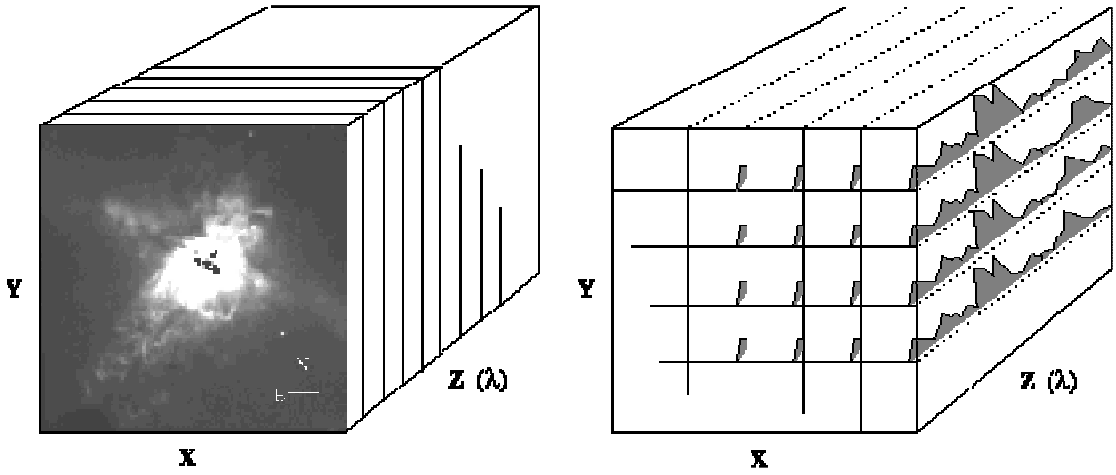

Figure 5.17: Shows the layers of the spectral images on the left, it also shows the spectral matrix cube on the right where all the wavelengths are aligned behind each other.

Each layer in the spectral image is a representation of the light at some wavelengths. And the number of layers "channels" depends on the spectral camera and its settings, where every channel refers to the steps at which the spectral range has been taken.

To view the Spectral image in RGB we have to convert all the channels altogether to produce a full RGB image. The process of converting from spectrum to RGB is one way and you cannot reconvert the RGB image into spectral image in an accurate way.

To convert the spectral image into RGB we first should convert it into XYZ Color Space so we get:

$$X = \frac{1}{N}\int_{\lambda} \overline{x}(\lambda)S(\lambda)I(\lambda)\mathrm{d}\lambda \quad (5.10)$$

$$Y = \frac{1}{N}\int_{\lambda} \overline{y}(\lambda)S(\lambda)I(\lambda)\mathrm{d}\lambda \quad (5.11)$$

$$Z = \frac{1}{N}\int_{\lambda} \overline{z}(\lambda)S(\lambda)I(\lambda)\mathrm{d}\lambda \quad (5.12)$$

$$N = \int_{\lambda} \overline{y}(\lambda)I(\lambda)\mathrm{d}\lambda \quad (5.13)$$

Note: see [35]

Where X Y Z are the tristimulus values for a color with a spectral reflectance $S(\lambda)$, and N is the number of channels in the spectral image. $\overline{x}$, $\overline{y}$ and $\overline{z}$ are the standard observation functions. $I(\lambda)$ is the spectral power distribution of a reference illuminant. The integral part can be replaced with summation so the equations will be:

$$X = \frac{1}{N}\sum_{i} \overline{x}_i S_i\, I_i \quad (5.14)$$

$$Y = \frac{1}{N}\sum_{i} \overline{y}_i S_i\, I_i \quad (5.15)$$

$$Z = \frac{1}{N}\sum_{i} \overline{z}_i S_i\, I_i \quad (5.16)$$

$$N = \sum_{i} \overline{y}_i\, I_i \quad (5.17)$$

Now to compute the XYZ for all the pixels the algorithm will be:

Input Spectral Image M, . $\overline{x}, \overline{y}, \overline{z}$ observation functions, $I(\lambda)$ is the spectral power distribution of a reference illuminant, Output 2D array of XYZ data structure.

Struct XYZ

double X,Y,Z

End Struct

For i=1 to n

N+=$\overline{y}_i * I_i$

End for

For i=1 to width

For j=1 to height

For w=1 to n: number of channels

$X+= \overline{x}_w S_w I_w$

$Y+= \overline{y}_w S_w I_w$

$Z+= \overline{z}_w S_w I_w$

End For

X=(1/N)*X

Y=(1/N)*Y

Z=(1/N)*Z

XYZ[i,j]=[X,Y,Z]

X=Y=Z=0

End For

End For

Now we have computed the XYZ tristimulus values we need to convert them to RGB color model in order to construct the RGB image, so we can convert them by:

$$[r\ g\ b] = [X\ Y\ Z][M]^{-1} \tag{5.18}$$

Where r g b are the RGB values and X Y Z are the tristimulus values. M is the transformation matrix calculated from the RGB reference primaries.

So for D65 reference white the inversed transformation matrix is:

$$[M]^{-1} = \begin{bmatrix} 2.04148 & -0.969258 & 0.0134455 \\ -0.564977 & 1.87599 & -0.118373 \\ -0.344713 & 0.0415557 & 1.01527 \end{bmatrix} \tag{5.19}$$

So now the code and the algorithm to compute the RGB values from the XYZ space:

```
float[,] inv ={ { 2.04148f, -0.969258f, 0.0134455f },
                { -0.564977f, 1.87599f, -0.118373f },
                { -0.344713f, 0.0415557f, 1.01527f } };


Color[,] RGB  = new Color[width1, height1];
float[] rgb = new float[3];
float sum = 0.0f;
for (int i = 0; i < width1; i++)
{
  for (int j = 0; j < height1; j++)
  {
    for (int u = 0; u <= 2; u++)
    {

      sum = xyzMatrix[i, j].X * inv[0, u];
      sum += xyzMatrix[i, j].Y * inv[1, u];
```

```
            sum += xyzMatrix[i, j].Z * inv[2, u];
            rgb[u] = sum * 255;
            sum = 0.0f;
        }

        if (rgb[0] > 255)
            rgb[0] = 255;
        if (rgb[1] > 255)
            rgb[1] = 255;
        if (rgb[2] > 255)
            rgb[2] = 255;
        if (rgb[0] < 0)
            rgb[0] = 0;
        if (rgb[1] < 0)
            rgb[1] = 0;
        if (rgb[2] < 0)
            rgb[2] = 0;
        RGB[i, j] = Color.FromArgb((int)rgb[0], (int)rgb[1], (int)rgb[2]);
    }
  }
```

To get a specific channel out of the spectral image then process is much easier. The spectral image consists of many channels each of these channels or layers is actually an independent image. To get a specific channel all what we need to do is extract the specific channel without any modification.

The code and the algorithm to extract a specific channel will be:

input 3D matrix containing the spectral image data

input channel number cNo

output RGB array containg the selected image channel.

```
Color [,] RGB = new Color[width1, height1];
float[] rgb = new float[3];
```

```
for (int i = 0; i < width1; i++)
 {
  for (int j = 0; j < height1; j++)
   {
     rgb[0] = matrix[cNo, i, j] * 255;
     rgb[1] = matrix[cNo, i, j] * 255;
     rgb[2] = matrix[cNo, i, j] * 255;

     if (rgb[0] > 255)
        rgb[0] = 255;

     if (rgb[0] < 0)
        rgb[0] = 0;

        RGB[i, j] = Color.FromArgb((int)rgb[0], (int)rgb[0], (int)rgb[0]);
   }
```

```
  }
```

Figure 5.18: channel number 2 from the spectral image registered on the 3D mesh

Chapter 6: Experiments

In this chapter we will discuss the results of the thesis and the software accuracy, performance, the advantages and the disadvantages of the methods and algorithms used in the software.

6. 1 the 2D- 3D Registration Results

The results we obtained from the 2D-3D Registration algorithm that was described in chapter 3, can be checked by observing the texture mapped from the scanner camera and the scanner parameters with the spectral camera texture that was mapped to the 3D model. The results show that the texture mapping was successful, yet there is still small margin of error in the mapping if we observed long enough. For example the next figure 6.1 compares the texture mapping of both the spectral camera and the RGB scanner camera images.

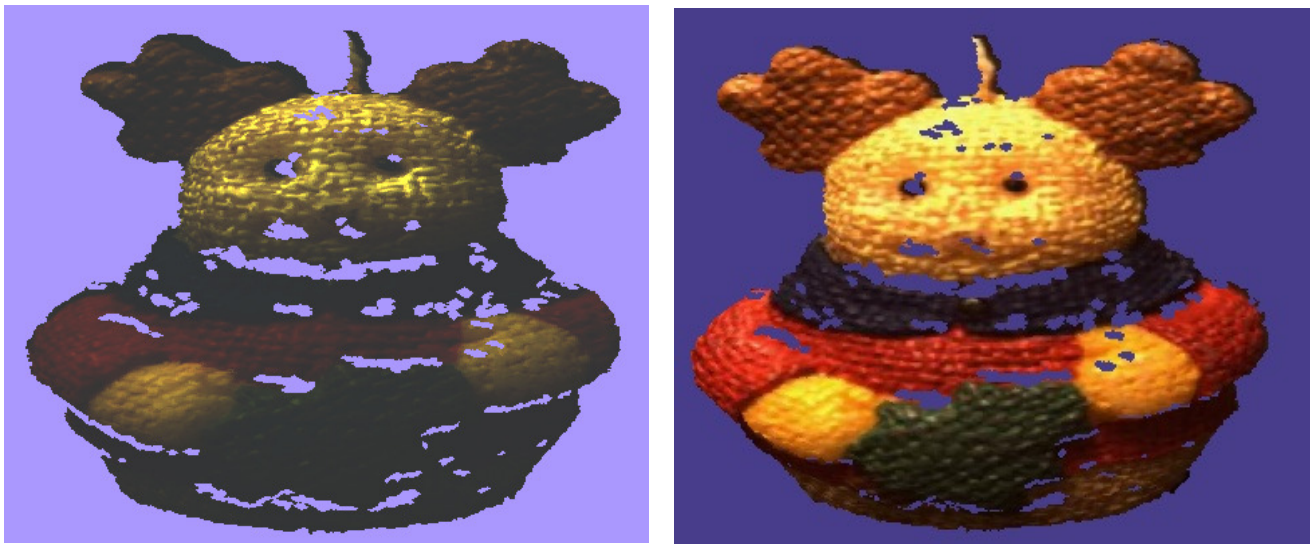

Figure 6.1: spectral image registered to the model on the left in compare with RGB scanner image registered to the mesh.

Note: The color differences are due the lightning when the spectral image was taken however the spectral texture was successfully registered to the mesh as shown in the pictures.

We observe from the two images (see figure 6.1) that there is a small error; this error is the result of the feature points' detection algorithm. The algorithm used can detect only patch of 3x3 or more but not with pixel and sub pixel accuracy; and as a results the transformation of the 3D

mesh lead to this inaccuracy. This error can be overlooked for non critical applications as this error is unnoticeable.

To increase the accuracy of the registration we can further refine our feature points' detection algorithm to achieve better results. Using the same algorithm but with better feature points' detector.

6. 2 3D-3D Registration

The process of the 3D to 3D registration was successfully accomplished using the Transformation matrix, yet as anything else it has its draw backs. Let us observe the following figure 6.2:

though the 3D-3D registration is mathematically approved and error margin should be up to zero, however that's not the case in this example. The reason for such an error is the selection of the feature points where the user has to interact with the software to choose the feature points on both of the meshes. The result of the 3D-3D registration depends on the selection of points made by the user.

Figure 6.2: shows 3D-3D registration results. This mesh is composed of two meshes.

One solution for this problem can be achieved by tuning location using cross correlation. We can use the normalized cross correlation to adjust the location of the selected points and then we can get the location with sub-pixel accuracy. This solution is mentioned in the Matlab 7.0 help files regarding the image registration procedure for fine tuning the selected control "feature" points.

6. 3 Spectral Analysis Tools

The spectral analysis tools in the software are yet very basic and need to be more developed. So far the only tool that has been added was changing the spectral channel of the texture to the 3D model. Other tools can be added to the software easily but for the lack of time they were postponed for future work.

Example of the changing the spectral channel can be seen in the following figure 6.3:

Figure 6.4: Spectral Channel mapped to the 3D Model.

Other features in the spectral analysis menu are the conversion from spectral to RGB and apply the texture to the model. The menu can extended by adding some options like choosing the light source and the inversed transformation matrix M.

Other suggested features for future work can be the addition of the histogram of a specific vertex texture in the model, where the user can click on a vertex and a pop up window with the histogram on it showing the spectral wavelength on that specific vertex.

6. 4 Model Manipulation

The added features of the model manipulation were the rotation, scaling and translation however these are only the basic editing features for the software. We can add model deformation tools which allow the user to edit the vertex location in the model. This feature will allow the user to use the software not only to study spectral and generating complete textured model but it can provide us with the tools to change the model shape.

This feature can be implemented easily, by selecting a specific vertex in the model and then drag it by the mouse to a new location and redraw the model with these new changes.

6. 5 Data Files

The data files formats used in this projected are Wavefront .Obj file, DirectX .X files and Konica Minolta .Vvd files. More 3D standard file formats can be added to facilitate importing/exporting from/to other softwares.

The reason for not using other data formats in the project is the time consuming process of implementing each importer/exporter of these files. The use of Wavefront .Obj format made it possible to import and export from the program to other programs since this format is widely used in most of the famous 3D modeling softwares such as 3DS Max and Maya.

Other formats are capable of holding more information about the object such as animation and bones such formats are 3DS from 3DS Max, MD2 and MD3 from Quake 2 and Quake 3 respectively. These formats were not considered for they are out of scope of the software since the use of the animation is not considered.

6. 6 Controlling the Digitizer

One of the main facilities in this project is controlling the Vivid9i Digitizer, The feature allows us to control the digitizer and initiate the scanning and automatically displaying it on the main window viewer.

This feature is using the default scanning parameters in the scanner and it can be further improved by allowing the user to control these parameters.

The adding of the parameters can be done by changing the VVDController.DLL library which we built for controlling the digitizer using the digitizer SDK. This library can be first modified to accept the coming parameters and then an interface can be build to change the parameter from inside the main software.

6. 7 the Lightening System

The lightening system in the software consists of 2 different types of the lights; the first is spot light and the second is all directional light. However these two lights systems are actually fixed in some point in space. To study the effect of the light on the different parts of the model we can rotate or translate the position of the model.

This translation and rotation can be avoided by letting the light itself moves in the scene instead of waiting the model to move.

Chapter 7: Conclusions

In this thesis we addressed some aspects of computer graphics; automatic 2D-3D Registration, semi-automatic 3D-3D Registration, spectral images as texture for the 3D models and Lightning system and other features for the software. The core of the thesis is the automatic 2D-3D Registration where a spectral texture was automatically mapped on a 3D model without camera calibration.

We used for the 2D-3D Registration an image processing technique which we developed for this purpose. We first started with image enhancement then we detected a feature points on both of the spectral and the RGB scanner images. We then transformed the model using the obtained transformation matrix to be aligned in such if a specific vertex was projected on the image plan then it will be on the exact pixel that correspond to it in the reference and target images of the spectral and RGB scanner image. We also discussed in details the mathematical background behind all the operations applied on the 2D-3D registration.

The spectral images were used in this project as texture for the 3D model. Different operations were applied, such as conversion from Spectral reflectance values to RGB values and extracting specific channel from the spectral images and applying it to the 3D model. We discussed the theoretical background behind the conversion and the use of the spectral images.

In the 3D-3D registration we allowed the user to be responsible of selecting feature points on both of the models. We computed the transformation matrix and aligned the two Models in order to merge the two models into one model. We also discussed the mathematical aspect behind the transformation.

We introduced a lightning system where the user can observe the light when it interacts with the 3D model. The light system includes Spot light and All Directional Light. We gave theoretical background on the lighting system.

Other features introduced in this project and thesis such as the various data files that have been used with this software. The data files varied from 3D files such as wave front .Obj files, DirectX

.X files and Konica Minolta .VVD to SPectral Binary files .SPB files. We talked about the files definition and how they were used in the software.

At the end of chapter 5 we talked about how we did the mesh triangulation and its theoretical background. We also introduced other features in the software such as model manipulation: scaling, rotation and translation and their mathematical background.

In this thesis we introduced the use of image registration and mesh transformation in texture mapping in order to register a 2D image on 3D model without loss in the texture quality. The results we obtained from this project and thesis were very good results, and yet as we discussed in chapter 6 the results can further enhanced.

The features of the software were good implemented and they cover the basic needs for studying 3D spectral imaging. These features can be extended to more generalizing the software such as for modeling, see chapter 6 for more information.

Bibliography

[1] H. Kawasaki, R. Furakawa, and Y. Nakamura, "3D acquisition system using uncalibrated line-laser project," IEEE, in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference*, vol. 1, pp. 1071-1075, 2006 , .

[2] R. B. Catalan, E. I. Perez, and B. Z. Perez, "Evaluation of 3D Scanners to Develop Virtual Reality Applications," IEEE, in *Electronics, Robotics and Automotive Mechanics Conference, 2007. CERMA 2007*, pp. 551-556, 25-28 Sept. 2007 , .

[3] M. Callieri, et al., "RoboScan: an automatic system for accurate and unattended 3D scanning," IEEE, in *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium*, pp. 805-812, 6-9 Sept. 2004 , .

[4] I. R. Khan, M. Okuda, and S. Takahashi, "Regular 3D mesh reconstruction based on cylindrical mapping," IEEE, in *Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on*, vol. 1, pp. 133-136, 27-30 June 2004 , .

[5] M. Levoy, et al., "The digital Michelangelo project: 3D scanning of large statues," ACM Press/Addison-Wesley Publishing Co, in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* , pp. 131-144, July 2000, .

[6] G. Elber, "Geometric texture modeling," , IEEEvol. 25, no. 4, July-Aug. 2005 .

[7] B. Lévy and J.-L. Mallet, "Non-distorted texture mapping for sheared triangulated meshes," ACM, in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, , July 1998.

[8] J. E. Dudgeon, *Algorithms For Graphics Texture Mapping*. Alabama: IEEE, Mar. 1991.

[9] J. K. Chittamuru, J. Euh, and W. P. Burleson, "An Adaptive Low Power Texture Mapping Architecture," , IEEEvol. 1, pp. I-21-24, Aug. 2002.

[10] R. G. Laycock and A. M. Day, "Automatic Techniques for Texture Mapping in Virtual Urban Environments," IEEE, in *Computer Graphics International, 2004. Proceedings*, pp. 586-589, 19-19 June 2004, .

[11] C. Poullis, S. You, and U. Neumann, "Generating High-Resolution Textures for 3D Virtual Environments using View-Independent Texture Mapping," IEEE, in *Multimedia and Expo, 2007 IEEE International Conference on*, pp. 1295-1298, 2-5 July 2007 , .

[12] L. Liu, G. Yu, G. Wolberg, and S. Zokai, "Multiview Geometry for Texture Mapping 2D Images Onto 3D Range Data," IEEE, in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2, pp. 2293-2300, 2006 , .

[13] J. Zhang and S. Luo, "image-Based Texture Mapping Method in 3D Face," IEEE, in *Complex Medical Engineering, 2007. CME 2007. IEEE/ICME International Conference on*, pp. 147-150, 23-27 May 2007 , .

[14] J. Maillot, H. Yahia, and A. Verroust, "Interactive Texture Mapping," ACM, in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* , pp. 27-34, September 1993, .

[15] B. Lévy, "Constrained Texture Mapping for Polygonal Meshes," ACM, in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* , pp. 417-424, August 2001 , .

[16] D. Piponi and G. Borshukov, "Seamless texture mapping of subdivision surfaces by model pelting and texture blending," ACM Press/Addison-Wesley Publishing Co, in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* , pp. 471-478, July 2000, .

[17] R. Kurazume, K. Nishino, Z. Zhang, and K. Ikeuchi, "Simultaneous 2D images and 3D geometric model registration for texture mapping utilizing reflectance attribute," The 5th Asian Conference on Computer Vision, 23–25 January 2002, Melbourne, Australia, in *Contributors: The Pennsylvania State University CiteSeer Archives* , , 2001.

[18] L. Liu and I. Stamos, "Automatic 3D to 2D registration for the photorealistic rendering of urban scenes," IEEE, in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, pp. 137-143, 20-25 June 2005 , .

[19] K. Itoh, W. Watanabe, H. Arimoto, and K. Isobe, "Coherence-Based 3-D and Spectral Imaging and Laser-Scanning Microscopy," , IEEEvol. 94, no. 3, pp. 608-628, Mar. 2006.

[20] N. B. Tondello, et al., "A System for 3D Modeling Frescoed Historical Buildings with

Multispectral Texture Information," , Springer-Verlag New York, Inc. Secaucus, NJ, USA vol. 17, no. 6, pp. 373-393, Oct. 2006.

[21] L. Ikemoto, N. Gelfand, and M. Levoy, "A hierarchical method for aligning warped meshes," IEEE, in , pp. 434-441, 2003, .

[22] P. J. Besl and H. D. McKay, "A Method for Registration of 3-d Shapes," , IEEE, Pattern Analysis and Machine Intelligence, IEEE Transactions, vol. 14, no. 2, pp. 239-256, Feb. 1992.

[23] H. Li and R. Hartley, "The 3D-3D Registration Problem Revisited," IEEE, in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference* , pp. 1-8, 14-21 Oct. 2007 , .

[24] C. Sinthanayothin and W. Tharanon, "3D-3D Registration: Surface Rendering Plus Skull and Soft Tissue Registration," IEEE, in *Industrial Electronics and Applications, 2006 1ST IEEE Conference* , pp. 1-5, May 2006, .

[25] H. Unten and K. Ikeuchi, "Color Alignment in Texture Mapping of Images under Point Light Source and General Lighting Condition," , IEEEvol. 1, pp. I-234-I-239, Jul. 2004.

[26] E. Beauchesne and R. Sbastien, "Automatic relighting of overlapping textures of a 3D model," , IEEEvol. 2, pp. II-166-73, Jun. 2003.

[27] KONICA MINOLTA SENSING,INC. (2006) VIVID Software Development Kit II Version 2.10 Programmer's Guide.

[28] D. Salomon, *Transformations and Projections in Computer Graphics*. Springer-Verlag London Limited , 2006.

[29] J. C. Bezdek, J. Keller, R. Krisnapuram, and N. R. Pal, *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Springer Science+Business Media, Inc, 2005.

[30] D. Phillips, *Image Processing in C*, 2nd ed.. Miller-Freeman, Inc, 2000.

[31] A. A. Goshtasby, *2-D and 3-D Image Registration: for Medical, Remote Sensing, and Industrial Applications*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2006.

[32] Microsoft. (June 2007) DirectX 9.0 Programmer's Reference.

[33] T. Miller, *Managed DirectX® 9 Kick Start: Graphics and Game Programming*. Sams Publishing, October 22, 2003.

[34] A. Ellen, S. Lobão, and E. HATTON, *.NET Game Programming with DirectX 9.0*. 2003: Apress.

[35] B. J. Lindbloom. Bruce Lindbloom. Useful Color Equations [Online]. http://www.brucelindbloom.com/index.html?Equations.html