

Real-Time Stereopsis

By

Sayed Ali Kasaei Zadeh Mahabadi

Dissertation submitted in partial fulfillment of
the requirements for the
Bachelor of Technology (Hons)
(Information Communication Technology)
JANUARY 2008

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

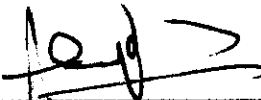
Real Time Stereopsis

By

Sayed Ali Kasaei Zadeh Mahabadi

A project dissertation submitted to the
Information communication Technology Programme
Universiti Teknologi PETRONAS
In partial fulfillment of the requirement for the
BACHELOR OF TECHNOLOGY (Hons)
(Information Communication Technology)

Approved by,



(Assoc. Prof. Dr. Abas Md Said)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

January 2008

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



SAYED ALI KASAEI ZADEH MAHABADI

ABSTRACT

The computerized development always was on the center of attraction. More accurate result generation in shorter time period has done by computers, make humans eager to gives computers more responsibility and tasks. Computer vision is one of these tasks that for past centuries take lots of time and effort one the rode of perfection.

One of the areas in the computer vision is stereo vision or Stereopsis. This area start in early 1970's and still up to day is one of the mysteries part of computer vision. Peoples try to make computer sees as human see. Up today there are many algorithm developed and invented by scientist but it's long way to go.

The main purpose of this report is to shows how it's possible for computer to calculate depth from 2D images and then base on some algorithms, it tries to construct 3D result. But why we need to make all these efforts and why it's so important to make computer sees as human see.

One of the strongest effects of this process is for 3D animation development. Generating 3D libraries and assist game developers or generally graphic developers. Imagine for development of one hour movie 60 computers work for one year and see how fast it will be to have all the models available in very short time.

Another effect of this system if for virtual realities systems. Beside 3D modeling which require real-time rendering it also require certain amount of tracking for user interaction. This process normally handle by sensors, where its limited to few sensors and also it makes users uncomfortable and its harder for virtual environment to be more realistic for users.

Finally in robot vision 3rd dimension is essential. Finding exact location of object, obstacles and etc. is the main goal of robots before they can perform any tasks. So base on understanding of depth the robots can easily move without any collision.

ACKNOWLEDGMENT

All my gratitude is towards God Almighty for giving me the strength, wisdom and patience in order to complete this project in the most efficient and timely manner.

I would like to express my utmost gratitude towards the assistance and generosity of all the people to whom which without this project would have not achieved its completion. In this process I would like to thank my supervisor Assoc. Prof. Dr. Abas Md Said for being as generous and understanding as human can be. Without his helpful guidance, advice and motivations, I might have not been able to complete this project with the required quality.

To Dr. Mohamed Nordin Zakaria who guided me with patients and caring, to my dear family, my supporting parents who have helped me to grow as the independent man I am and guided me in the darkest and most tough times. Lastly but not least my respectful friends who have truly proven their friendship and support throughout the entire process by motivating me, raising me up when I got down and being there for me when I needed a hand.

And to all those people who have shared their experience and ideas which I have forgotten to mention in this small but gratified acknowledgment,

Thank you.

TABLE OF CONTENTS

CERTIFICATION OF APPROVAL	i
CERTIFICATION OF ORIGINALITY	ii
ABSTRACT	iii
ACKNOWLEDGMENT	v
TABLE OF CONTENTS.....	vi
TABLE OF ILLUSTRATION.....	ix
Figures.....	ix
Tables	xi
Equations.....	xi
Chapter 1 INTRODUCTION.....	1
1.1 BACKGROUND OF STUDY	1
1.2 PROBLEM STATEMENT	1
1.3 OBJECTIVE AND SCOPE OF STUDY	5
Chapter 2 LITERATURE REVIEW.....	6
2.1 INTRODUCTION.....	6
2.2 Depth Perception	7
2.3 Correspondence.....	9
2.3.1 Introduction.....	9
2.3.2 Discontinuity.....	9
2.3.3 Occlusion	10

2.4 General information about CCD cameras	11
2.4.1 Introduction:.....	11
2.4.2 Charge-coupled device (CCD).....	13
2.4.3 Bayer filter	15
Chapter 3 METHODOLOGY	16
3.1 RESEARCH METHOD	16
3.2 Depth	17
3.2.1 Introduction.....	17
3.2.2 Depth calculation	17
3.3 Correspondence.....	21
3.3.1 Introduction.....	21
3.3.2 Block matching	21
3.3.3 Finding good features to track	21
3.3.4 Detecting points in second image	22
3.4 3D reconstruction	24
3.4.1 Introduction.....	24
3.4.2 Dot cloud.....	24
3.4.3 Wired reconstruction.....	24
3.4.4 Solid reconstruction	25
3.5 Experimental system design.....	26
3.5.1 Class Diagram	26
3.5.2 Flow chart	27
3.5.3 GUI	30
3.6 Experimental system Implementation.....	31
3.6.1 Main file.....	31
3.6.2 Image processing	32

3.7 TOOLS	37
Chapter 4 RESULT AND DISCUSSION.....	38
4.1 Depth calculation.....	38
4.1.1 Simple object	38
4.1.2 Transparent Bottle.....	40
4.2 Fining correspondence	41
4.3 3D reconstruction	42
4.3.1 Dot cloud.....	42
4.3.2 Wired result.....	43
4.3.3 Solid result	43
Chapter 5 CONCLUSION	44
Chapter 6 REFERENCES.....	45
APPENDIX I	I
1) Schedule FYP I.....	I
2) Schedule FYP II.....	II
APPENDIX II.....	III
1) Main class header:.....	III
2) Main Class source:.....	IV
3) Image processing class header:.....	VIII
4) Image processing class source:.....	VIII
5) Graphic class header:.....	XVII
6) Graphic class source:	XVIII

TABLE OF ILLUSTRATION

Figures

Figure 1: Some sensors performance comparison ^[1]	2
Figure 2: Magnetic tracker accuracy degradation ^[1]	3
Figure 3: Sensors ^[2]	4
Figure 4: Data glove ^[2]	4
Figure 5: Different view point ^[3]	6
Figure 6: Stereopsis for human eye ^[5]	7
Figure 7: Depth in human eyes ^[7]	8
Figure 8: Epipolar Geometry ^[8]	9
Figure 9: Sample of Digital camera single-lens reflex camera	11
Figure 10: Single-Lens Reflex camera cross section ^[26]	12
Figure 11: charge-coupled device (CCD)	13
Figure 12: Two-dimensional CCD-sensor	14
Figure 13: CCD color sensor Bayer pattern on sensor ^[27]	14
Figure 14: Profile/cross-section of sensor ^[28]	15
Figure 15: Methodology	16
Figure 16: Three-plane view	17
Figure 17: Cameras view and object triangle ^[29]	18
Figure 18: Dot cloud	24
Figure 19: Wired structure	25
Figure 20: Solid Structure	26

Figure 21: Active Stereo Vision Class Diagram.....	26
Figure 22: Main Class Flow chart.....	27
Figure 23: Image processing main flow chart.....	28
Figure 24: Transfer Flow Chart	29
Figure 25: 3D Display.....	29
Figure 26: Error Message.....	30
Figure 27: 3D Display.....	30
Figure 28: Main file interface	31
Figure 29: Camera Selection Form	33
Figure 30: image division	33
Figure 31: optical fellow and good Features to track result.....	35
Figure 32: Delaunay Triangulation.....	35
Figure 33: Points on Image	38
Figure 34: Object Depth Result	39
Figure 35: Transparent Bottle	40
Figure 36: Transparent Object Depth Graph	41
Figure 37: Sample of good feature to track	42
Figure 38: Dot cloud Result.....	42
Figure 39: Wired Structure	43
Figure 40: Solid Structure.....	43

Tables

Table 1: Simple Object	39
Table 2: Transparent Object Result	40
Table 3: Schedule and milestone FYP I.....	I
Table 4: Schedule and milestone FYP II	II

Equations

Equation 1: Similar Triangle.....	19
Equation 2: Similar Triangle.....	19
Equation 3: Similar Triangle.....	19
Equation 4: Similar Triangle.....	19
Equation 5: calculation of o_1h (A) length	20
Equation 6: Calculation of object depth.....	20
Equation 7: Calculation of object depth from Camera view.....	21

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND OF STUDY

Stereopsis (from “stereo” which means solidity and “opsis” which means vision or sight) is the process in visual perception that leads to perception of stereoscopic depth. In turn, stereoscopic depth is the sensation of depth that emerges from the fusion of the two slightly different projections of the world on the two retinas.

In computer vision, structure from motion refers to the process of building a 3D model from video of a moving rigid object. Algorithmically, this is very similar to stereo vision where a 3D model is built from two simultaneous images of the same object. In both cases, multiple images are taken of the same object and corresponding features are used to compute 3D locations. In structure from motion, the images are taken at different points in time comparing to stereo vision where images are taken at different points in space.

Colloquially, structure from motion is sometimes used for any 3D reconstruction built from 2D images of a rigid (or static) object. Because of this colloquial usage, structure from motion has significant overlap with stereo vision.

1.2 PROBLEM STATEMENT

3D modeling has significant impact on virtual realities, game development, simulations environment and many other graphical fields. There are many methods to develop 3D models and model database but current methods have many problems as follow:

Time: Even by using many advance computer tools to develop models but still each model requires lots of time to develop. Normally these tools only provide basic shape like cube or

calendar and developer requires developing the models by combining and reshaping these basic shapes. Even when special tools have capabilities like laser scanner, etc. because of design the scanning is too slow.

ACCURACY (mm ³)	RANGE (m)	LATENCY (sec×10 ⁻³)	UPDATE RATE* (datasets/sec)
Best performance			
0.5/0.03	30 × 30	0.0002	2,000
HiBall	IS-900	Push	HiBall
0.8/0.15	12.2 × 12.2	1	256
Fastrack	HiBall	HiBall	InterTrax2
1/0.5	2	4	240
laserBIRD	laserBIRD	InterTrax2	laserBIRD
2/0.5	1.52	7	180
FlockBIRDS	Logitech	laserBIRD	IS-900
4/0.2	1.2	7.5	160
IS-900	FlockBIRDS	flockBIRD	3-D BIRD
4/NA	0.75	8.5	144
Push	Fastrack	Fastrack	FlockBIRDS
NA/4	NA	10	120
3-D BIRD	3-D BIRD	IS-900	Fastrack
NA/5	NA	15	70
Intertrax2	Intertrax2	3-D BIRD	Push
30	NA	30	50
Logitech	Push	Logitech	Logitech
Worst performance			

* For single sensing element

Figure 1: Some sensors performance comparison ^[1]

Cost: Since the production too slow companies require hiring many 3D developers and spending lots of money on expensive tools. However, the model is not completely similar to reality. The cost will increase as the quality of models getting better.

Efficiency: Using sensors is another method used for Virtual Environment (VE) and 3D animation. These devices are normally having lots of problem, like accuracy (the tracker position), jitter (change position of tracker because of noises), drift (decrease accuracy base on time) and latency (also known as delay). Another problem is environmental affect on the output. For example, for magnet sensors the metals affect the result.

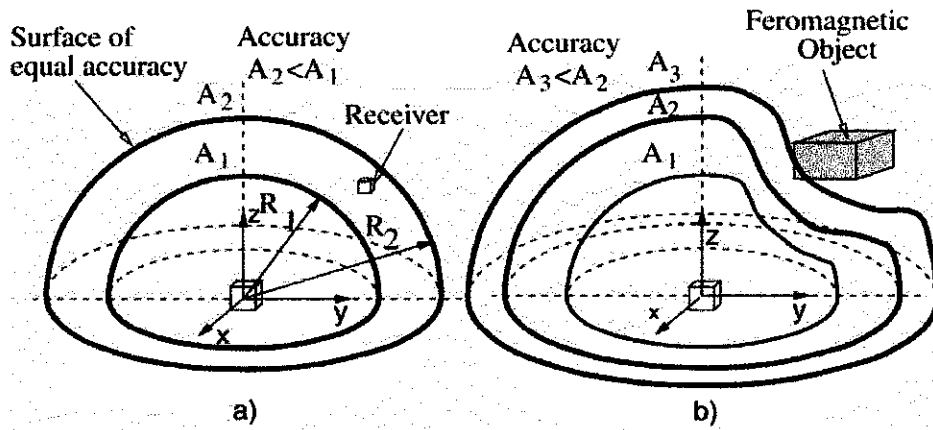


Figure 2: Magnetic tracker accuracy degradation ^[1]

Weight: Sensors are normally heavy devices. The movement in these devices is really hard and uncomfortable. This has negative impact on immerse in VE.



Figure 3: Sensors ^[2]

Others: There are many other problems such as sensors coverage. Normally tracker and sensors are designed for specific part of body like head, hand and etc. A sample of these sensors is data glove.

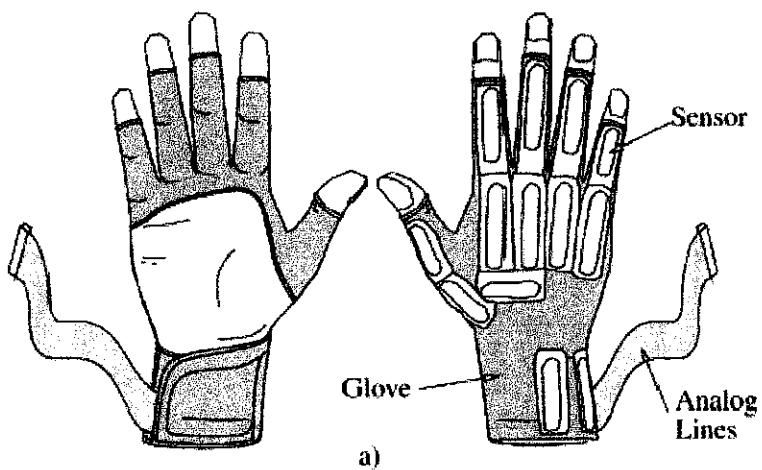


Figure 4: Data glove ^[2]

1.3 OBJECTIVE AND SCOPE OF STUDY

The objectives of the projects are:

Design a system to calibrate the camera and extract the epipolar geometry of the image sequence automatically.

Calculate stereoscopic depth of objects

Design a system which displays the result in three dimension environment.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

Nowadays simulation environments, virtual reality systems, game developing and animation companies are highly dependent on the 3D construction. Currently these developments are done manually and it requires large amount of time to increase the quality of the system. Also it requires more sensors to build more realistic 3D motions which increase the cost of production.

By using one or two cameras it allows us to extract 3D information simply based on the different angle view point as shown in *Figure 5*. This can be done by using two cameras or by rotating one camera in some angle(s). There is also another way to simulate the camera rotation which calculates the rotation angle of the object without rotating the camera.

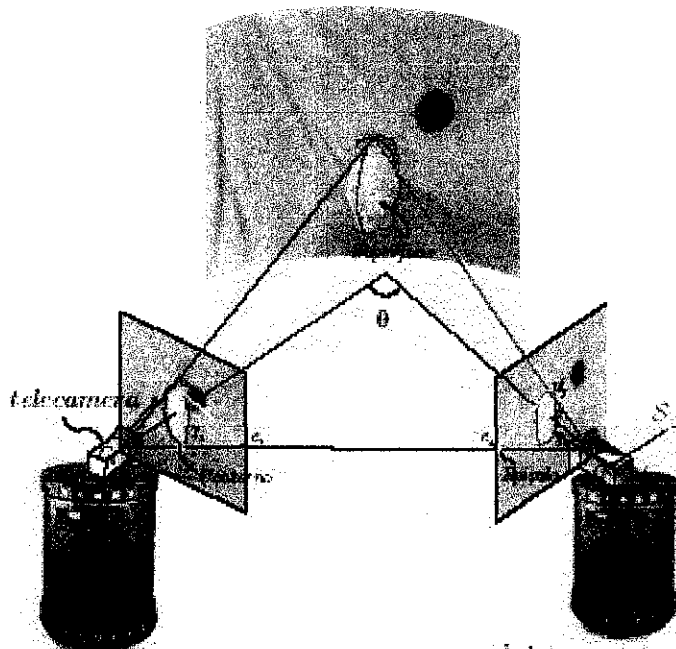


Figure 5: Different view point ^[3]

2.2 Depth Perception

Human normal inter-papillary distance is 2.5 to 2.6 inches. If we could increase this distance we would increase our perception of depth. Stereo pairs greatly stretch this normal eye base (inter-papillary distance) and give us the exaggerated 3-D photographic effect we perceive when viewing the stereo pairs [4] *Figure 6*.

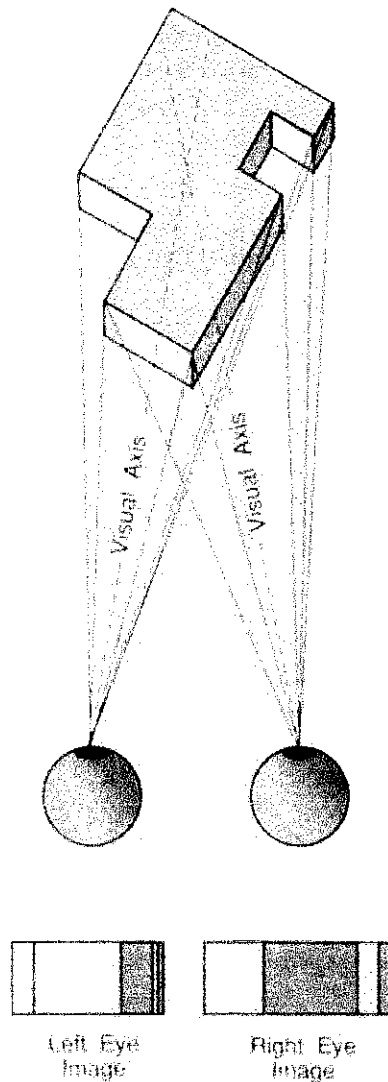


Figure 6: Stereopsis for human eye [5]

In other words each eye captures its own view and the two separate images are sent on to the brain for processing. When the two images arrive simultaneously at the back of the brain, they are united into one picture. The mind combines the two images by matching up the similarities and adding in the small differences. The small differences between the two images add up to a big difference in the final picture! The combined image is more than the sum of its parts. It is a three-dimensional stereo picture [6] *Figure 7*.

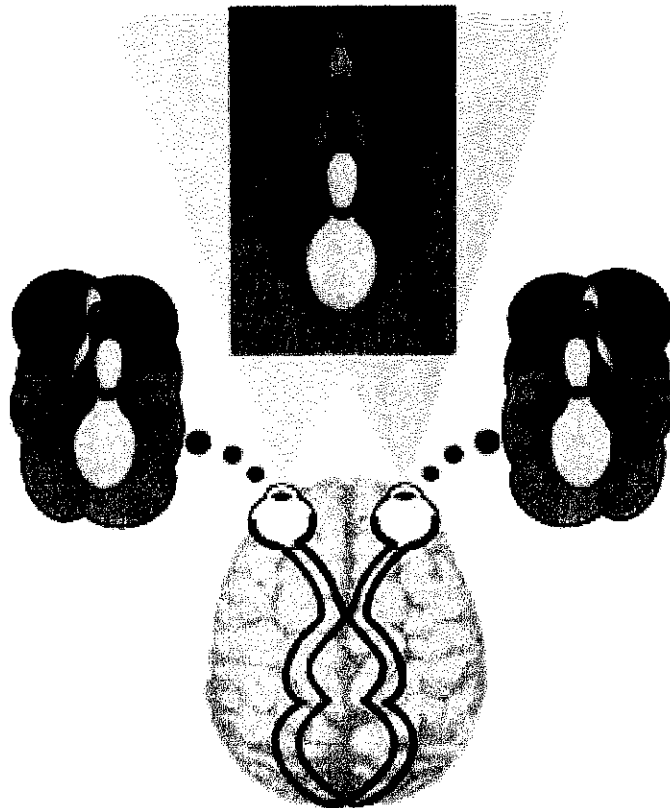


Figure 7: Depth in human eyes [7]

In this project we try to simulate human eyes by using two cameras and extract 3D depth from two different viewing points of these two cameras and each time there are two shots from the scene. The method used to compute the depth of a point is called triangulation.

Figure 8 shows a simple illustration of how triangulation works based on the epipolar geometry. The focal length of the cameras (f), the angles Θ_1 and Θ_2 , the camera center points (c_1 and c_2) in the image planes (IP1 and IP2), the image points (of three-dimensional point P) p_1 and p_2 , and

the horizontal distance (v_1 and v_2) between the image points and the camera center image point for each image are known. This leaves the perpendicular distance D from the baseline to the point P as the only unknown.

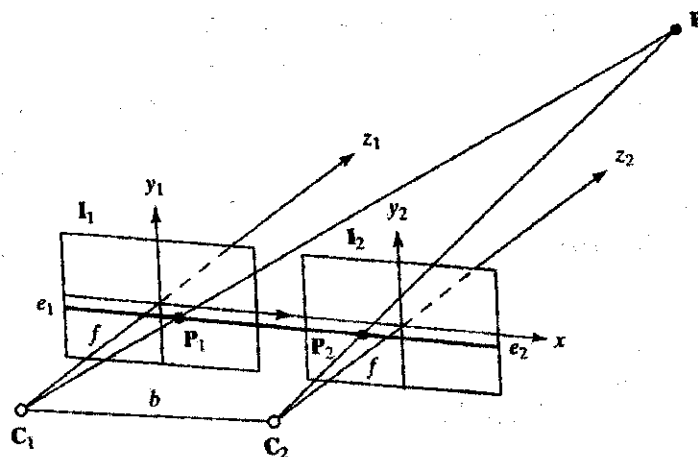


Figure 8: Epipolar Geometry ^[8]

2.3 Correspondence

2.3.1 Introduction

stereo correspondence is considered as a classical difficult problem due to its significance in computer vision and inherited ambiguity. It takes two or more images simultaneously captured by cameras from different viewpoints as its input. The resultant output is a dense disparity map that represents the correspondence between points in different images. The obtained disparity map can be used to recover the three-dimensional structure in the scene.

Two of the main challenges in stereo are discontinuity and occlusion problems.

2.3.2 Discontinuity

The discontinuity issue stems from a smoothness assumption, which is explicitly or implicitly used in many dense stereo approaches [9]. It assumes the disparity map to be smooth almost everywhere. However, this is violated at the boundary of the object. The convex smoothness

function entails a significant penalty for large discontinuity and, therefore, leads to poor object boundary results [10], [11]. To cure this, some discontinuity-preserving smoothness functions are designed to improve the accuracy at discontinuity areas [9]. Common discontinuity preserving smoothness functions include the Potts function [12] and the truncated function [13]. A fixed amount of penalty is imposed for large discontinuity in these methods. Moreover, the intensity differences between neighboring pixels are also used to guide the smoothness criteria [14], [13] so that neighboring pixels with similar colors are given harder smoothness constraints because they are more likely to have similar disparities. Recently, several segment-based methods have been proposed [15], [16], [17], [18], [19], [20]. Tao et al. [15] provided a global matching framework using image segmentation information. Hong and Chen [16] used graph-cuts to provide a global solution for segment matching, whereas a region-growing strategy was used by Wei and Quan [17]. Bleyer and Gelautz [18] formulated the correspondence problem in combination with the pixel and segment levels. The correspondence problem is modeled in the segment level and the occlusion is detected in the pixel level using the uniqueness constraint. In all these algorithms, a color segmentation process initially separates the reference image into several regions with uniform (or similar) colors and each region is assumed to correspond to a plane in the scene. With this polyhedral approximation of the scene, matching is performed using a segment as a unit. The discontinuity is constrained to be at the boundaries of a segment. The untextured area is matched as a large unit, so more information than as individual pixels can be gathered and improved performance can be obtained when processing images from a natural scene. Although impressive results are reported, only the segmentation information in the left (or reference) image is used and the occlusion result is still not accurate. Moreover, the violation of the discontinuity assumption still causes obvious artifacts in the result.

2.3.3 Occlusion

The second challenge in stereo correspondence is occlusion handling. Due to the structure of the scene, some parts of an object within it may be visible in only one of the cameras. These points are called half-occlusion points [21], and their projection onto the image is known as occluded points or occlusions, since their corresponding points in other images are not visible. The main difficulty for the occlusion problem is that occluded points cannot be detected directly, and we

can only use the correspondence of visible (opposite of occluded) points with other assumptions to detect them. Methods using ordering and uniqueness constraints are two traditional ones for occlusion handling. The ordering constraint inhibits the ordering change of corresponding points in different images. It is often used in a dynamic programming framework [21] because it can reduce the solution space and allow for a more efficient algorithm. But, it is often violated when thin, front objects exist in the scene. The uniqueness constraint, however, only prevents a point in one image from being matched with more than one point in the other image and ordering change is allowed. Zitnic and Kanade [22] used the uniqueness as the inhibition in their cooperative framework, while Ishikawa and Geiger [11] imposed it in a max-flow framework. Kolmogorov and Zabih [14] used the pixel assignment formulation for the correspondence problem and tried to find an optically unique configuration using graph-cuts. Sun et al. [23] used a variant version of uniqueness constraint, the visibility constraint, to detect occlusions in an iterative belief propagation framework. The visibility constraint can avoid some problems raised from the sampling problem pointed out by Ogale and Aloimonos [24] when horizontally slanted planes exist in the scene. Promising improvements on occlusion results are reported in the above papers. For other occlusion handling techniques, readers can refer to surveys by Egnal and Wildes [21] and by Brown et al. [25].

2.4 General information about CCD cameras

2.4.1 Introduction:



Figure 9: Sample of Digital camera | single-lens reflex camera

A digital camera as shown in *Figure 9* is an electronic device used to capture and store photographs digitally, instead of using photographic film like conventional cameras, or recording images in an analog format to magnetic tape like many video cameras. All digital cameras use either a charge-coupled device (CCD) or a CMOS image sensor to sense the light intensities across the focal plane.

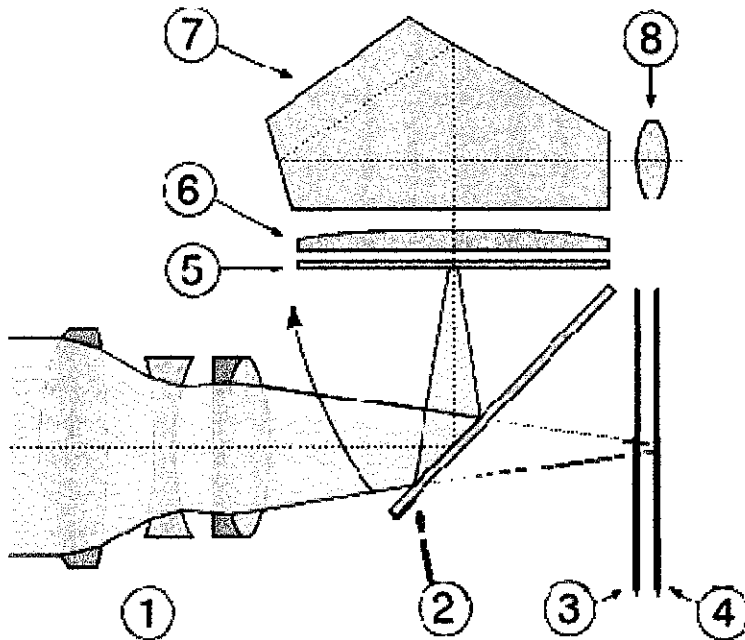


Figure 10: Single-Lens Reflex camera cross section ^[26]

Figure 10 displays basic structure of digital camera. This camera uses a mirror to show the image that will be captured in a viewfinder. The cross-section (side-view) of the optical components of a SLR shows how the light passes through the lens assembly (1), is reflected by the mirror (2) and is projected on the matte focusing screen (5). Via a condensing lens (6) and internal reflections in the roof pentaprism (7) the image appears in the eyepiece (8). When an image is taken, the mirror moves in the direction of the arrow, the focal-plane shutter (3) opens, and the image is projected in the sensor (4) in exactly the same manner as on the focusing screen.

2.4.2 Charge-coupled device (CCD)

A charge-coupled device (CCD) as in *Figure 11* is an analog shift register, enabling analog signals (electric charges) to be transported through successive stages (capacitors) controlled by a clock signal. CCDs which contain grids of pixels are used in digital cameras, optical scanners and video cameras as light-sensing devices. They commonly respond to 70% of the incident light (meaning a quantum efficiency of about 70%) making them far more efficient than photographic film, which captures only about 2% of the incident light. As a result, CCDs were rapidly adopted by astronomers.

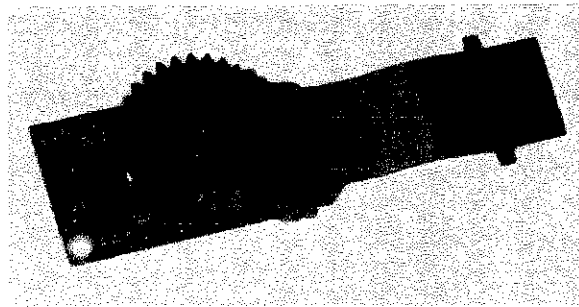


Figure 11: charge-coupled device (CCD)

In One-dimensional CCD, an image is projected by a lens on the capacitor array, causing each capacitor to accumulate an electric charge proportional to the light intensity at that location. An one-dimensional array, used in line-scan cameras, captures a single slice of the image, while a two-dimensional array, used in video and still cameras, captures the whole image or a rectangular portion of it as shown in *Figure 12*.

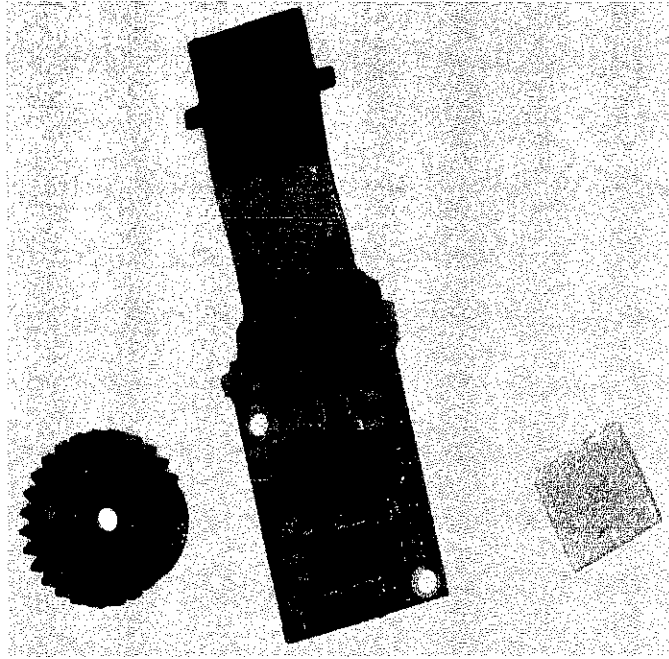


Figure 12: Two-dimensional CCD-sensor

Once the array has been exposed to the image as in *Figure 13*, a control circuit causes each capacitor to transfer its contents to its neighbor. The last capacitor in the array dumps its charge into an amplifier that converts the charge into a voltage. By repeating this process, the control circuit converts the entire contents of the array to a varying voltage, which it samples, digitizes and stores in memory. Stored images can be transferred to a printer, storage device or video display. CCDs are also widely used as sensors for astronomical telescopes, and night vision devices.

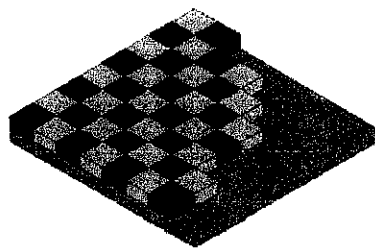
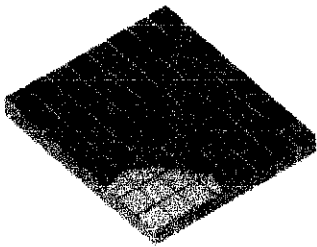


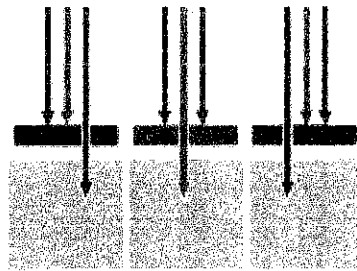
Figure 13: CCD color sensor | Bayer pattern on sensor ^[27]

2.4.3 Bayer filter

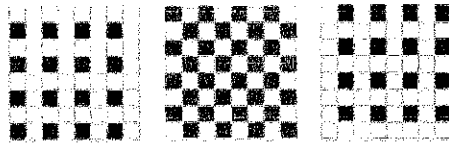
A Bayer filter mosaic is a color filter array (CFA) for arranging RGB color filters on a square grid of photo sensors, shown in *Figure 14*. The term is derived from the name of its inventor, Dr. Bryce E. Bayer of Eastman Kodak, and refers to a particular arrangement of color filters used in most single-chip digital image sensors used in digital cameras, camcorders, and scanners to create a color image. The filter pattern is 50% green, 25% red and 25% blue, hence is also called RGBG or GRGB.



In conventional systems, color filters are applied to a single layer of photodetectors in a tiled mosaic pattern.



The filters let only one color of light—red, green or blue—pass through to any given pixel location, allowing it to record only one color.



As a result, mosaic sensors capture only 25% of the red and blue light, and just 50% of the green.

Figure 14: Profile/cross-section of sensor ^[28]

CHAPTER 3

METHODOLOGY

3.1 RESEARCH METHOD

Base on the methodology in *Figure 15*, depth Calculation is the first step in this project which calculates depth of specific point in two cameras view. The next step is to find all points future in first camera and find its corresponding points on the next camera view. Base on this finding it is possible to generate dot cloud. By connecting these points (dots), we generate basic structure of 3D model and it will be completed by texturing the polygons.

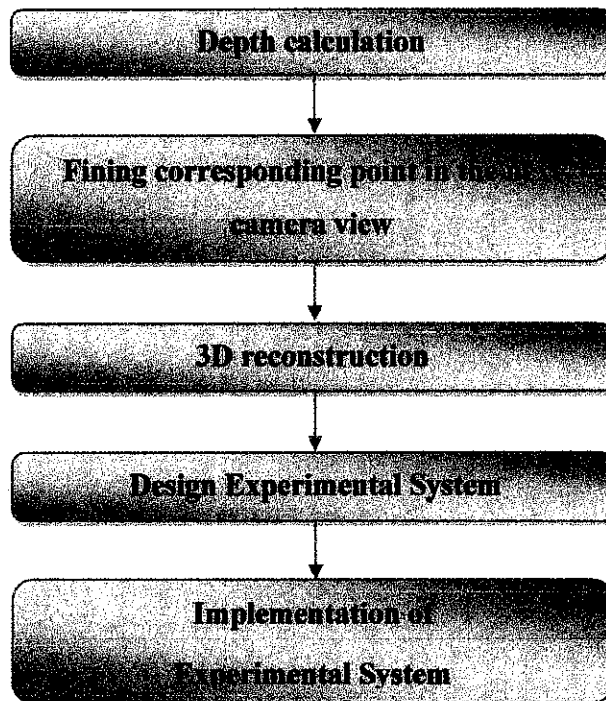


Figure 15: Methodology

3.2 Depth

3.2.1 Introduction

Depth is the most basic part of this project. This calculation is based on similar triangles. In this part we assume that two cameras are identical and have almost same focal length, and two cameras view are in the same plane, and the height of object in two views are the same. With those assumptions, two point views are in one line and parallel to x axis, as in *Figure 16*.

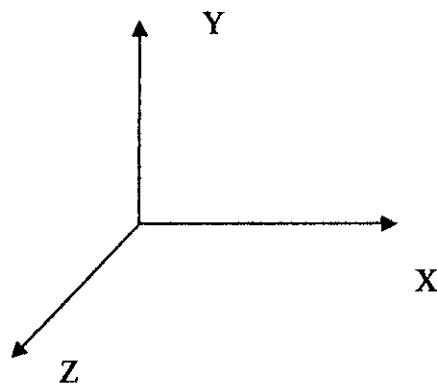


Figure 16: Three-plane view

3.2.2 Depth calculation

Based on these assumptions, every object creates a triangle $T_1[o_2, o_1, q]$ as shown in *Figure 17*.

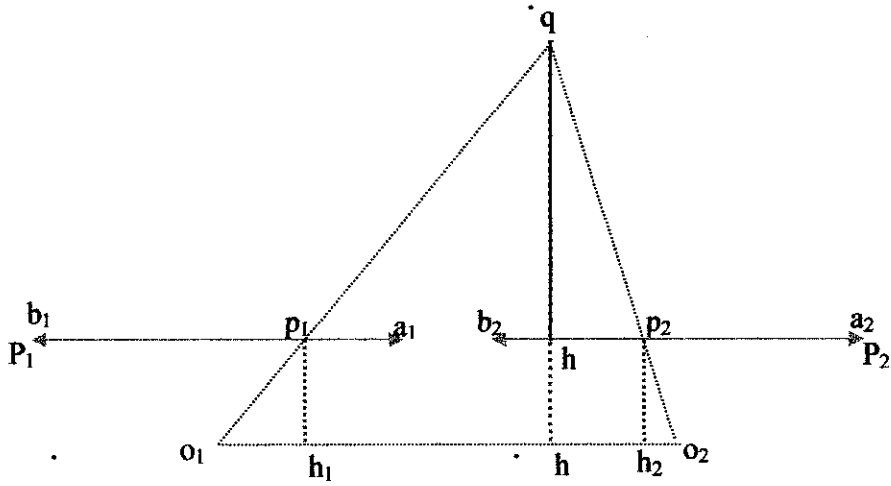


Figure 17: Cameras view and object triangle ^[29]

In this triangle q is the object and o_1 is focal point of camera one, where all the light enters to the camera will gather on that point o_2 is similar to o_1 and represents focal points of camera two. $P_1[a_1, b_1]$ is the first camera view and $P_2[a_2, b_2]$ is second camera view. The object reflection in camera one is p_1 and for camera two is p_2 . There are 4 triangles we work on.

$$T_1[o_1, q, h]$$

$$T_2[o_2, q, h]$$

$$T_3[o_1, p_1, h_1]$$

$$T_4[o_2, p_2, h_2]$$

T_1 and T_3 are similar triangles because $p_1h(Z)$ is parallel with $p_1h_1(F_1)$ since F_1 is altitude of T_3 and Z is altitude of T_1 . They have shared angle so the two triangles are similar to each other.

So the ratio of $o_1h_1(N_1)$ over $o_1h(A)$ is equal to ratio of F_1 over Z . So we have:

$$\frac{N_1}{A} = \frac{F_1}{Z}$$

Equation 1: Similar Triangle

Same goes for T_2 and T_4 , $p_2h_2(F_2)$ is parallel with Z and they share an angle. The ration of $o_2h_2(N_2)$ over $o_2h(B)$ is equal to ratio of F_2 over Z

$$\frac{N_2}{B} = \frac{F_2}{Z}$$

Equation 2: Similar Triangle

Since the two camera are align to each other and the focal is same so b_1a_2 is parallel with $o_1o_2(Q)$ and F_1 and F_2 are perpendicular lines to Q , F_1 and F_2 are parallel and they are equal to each other. In this situation the two equations above are equal to each other:

$$\frac{N_2}{B} = \frac{F_2}{Z} = \frac{F_1}{Z} = \frac{N_1}{A}$$

Equation 3: Similar Triangle

In *Figure 17*, sum of A and B is equal to Q so we can replace B by $Q - A \rightarrow Q = A + B \rightarrow B = Q - A$, by putting this value in Equation 3 and equal the ration of N_2 over $(Q - A)$ and N_1 over A we have:

$$\frac{N_2}{Q - A} = \frac{N_1}{A}$$

Equation 4: Similar Triangle

By simplifying that equation we obtain:

$$N_2A = N_1(Q - A) \rightarrow$$

$$N_2A = N_1Q - N_1A \rightarrow$$

$$N_1A + N_2A = N_1Q \rightarrow$$

$$A(N_1 + N_2) = N_1 Q$$

Base on the above equations the A will be equal to:

$$A = \frac{N_1 Q}{N_1 + N_2}$$

Equation 5: calculation of o₁h (A) length

Now if we locate A to Equation 1 and simplify that equation we have:

$$N_1 Z = F_1 A \rightarrow$$

$$Z = \frac{F_1 A}{N_1} = \frac{F_1 N_1 Q}{N_1 + N_2} = \frac{F_1 Q}{N_1 + N_2}$$

Equation 6: Calculation of object depth

N_1 is the distance of object image from the camera center if the camera length is $2 * O$ and b_1, p_1 be B_1 , so the relation of N_1 and B_1 is:

$$N_1 = B_1 - O$$

The same result shows the relation of N_2 and $B_2 [b_2, p_2]$:

$$N_2 = O - B_2$$

So from above equations we have:

$$N_1 + N_2 = B_1 - O + O - B_2 \rightarrow$$

$$N_1 + N_2 = B_1 - B_2$$

This process simplifies the distance calculation (B_1 and B_2 are primary values that we can extract from cameras images).

Besides, Z is the object distance from the focal of camera where to simplify, we subtract the focal length from the result and we get the actual object depth (Z') from camera images.

$$Z = Z' - F$$

$$Z' = \frac{F_1 Q}{B_1 - B_2} - F_1$$

Equation 7: Calculation of object depth from Camera view

F_1 is the focal length and Q is the distance of two cameras, B_1 is the object view in camera one and B_2 is object view in camera two. To calculate the focal length of each camera, simply assigns value for Z' , B_1 , B_2 and Q to the above formula.

3.3 Correspondence

3.3.1 Introduction

As it discussed before stereo correspondence is considered as a classical difficult problem due to its significance in computer vision and inherited ambiguity. Stereo correspondence may be determined in a number of ways and by exploiting a number of constrains. one of these methods is block matching algorithm that implemented and tested in this system.

3.3.2 Block matching

Block matching method seek to estimate disparity at a point in one image by comparing a small region about that point (the template) with a series of small regions extracted from the other image (the search region). The Epipolar constrain reduce the search to one dimension. The first step in this algorithm is to find point with good features and then base on optical fellow search for correspondence in the next image.

3.3.3 Finding good features to track

The next step is to find the points automatically. In this step we need to ensure the selected points can be found in both images. The points normally are selected based on their neighbors. The neighbors should be evident. This process is implemented in OpenCV library and it is called

cvGoodFeaturesToTrack. The function *cvGoodFeaturesToTrack* finds corners with big Eigen values in the image. The function first calculates the minimal Eigen value for every source image pixel using *cvCornerMinEigenVal* function and stores them in temporary image. Then it performs non-maxima suppression (only local maxima in 3x3 neighborhoods remain). The next step is rejecting the corners with the minimal Eigen value less than $\text{quality_level} \cdot \max(\text{eig_image}(x, y))$. Finally, the function ensures that all the corners found are distanced enough one from another by considering the corners (the strongest corners are considered first) and checking that the distance between the newly considered feature and the features considered earlier is larger than min_distance . Then the function removes the features than are too close to the stronger features [30].

3.3.4 *Detecting points in second image*

The examination of visual cues in an image-such as shading an occlusion- often yields information about the relative distances of objects in a scene; however, it cannot provide a quantitative measurement of the distance to objects. When the scene is imaged simultaneously from two locations, stereo correspondence between the resulting images can be used to determine the distance of objects [31], [32].

$$\frac{x_L}{f} = \frac{x + d/2}{z}$$

$$\frac{x_R}{f} = \frac{x - d/2}{z}$$

$$\frac{y_L}{f} = \frac{y_R}{f} = \frac{y}{z}$$

After spotting points on the first picture, correspondence points are to be identified on the second picture. This task has not been done before with satisfactory precision. In order to increase the accuracy of this process, we aligned cameras so that they would be parallel to X -axis. As stated before, image is Bi-Dimensional matrix with every point having two elements, X and Y . Y is

height of the point and X is distance of the point from Y 's axis. In this system Y -Axis situated at most left part of the picture and X -Axis is at top. Thus the origin is at the most top left point of the picture. Actual height of the object in image is equal to image width minus object's Y value [33].

Since cameras are aligned to X Axis every point's height is equivalent in both images. So in order to determine depth of the object, we need to calculate precisely X amount, which is the distance of the object from Y -Axis. Furthermore, to facilitate indentifying the correspondence point, we divide the second picture into equivalent horizontal slices. This reduces search area and increases search accuracy.

Another problem that we face is viewing point; there are some objects in the first image that are not visible in the second image. Same applies to the first image in which some apparent objects in the first picture are not visible in the second image. To solve this problem we cut out an invisible area in each image. After finishing these pre-processing on the two images, it is the time to find the points on each slice of the image

Here we use another function from Open CV library. This function will search feature of first image in second image. There are many types of optical flow but optical flow for a sparse feature in pyramids is best choice for this project since the function searches correspondence slice on the second image.

The function *cvCalcOpticalFlowPyrLK* implements sparse iterative version of Lucas-Kanade optical flow in pyramids. It calculates coordinates of the feature points on the current video frame given their coordinates on the previous frame. The function finds the coordinates with sub-pixel accuracy [34].

3.4 3D reconstruction

3.4.1 Introduction

After finding the points with their correspondence it's require to transfer points from a 2-Dimension environment to a 3-Dimension environment. In this section, the steps of 3D reconstruction will discussed.

3.4.2 Dot cloud

After finding the points and their correspondence it's required to transfer the points to a 3-Dimensional environment and calculate depth [35]. The result of this action is called dot cloud *Figure 18*.

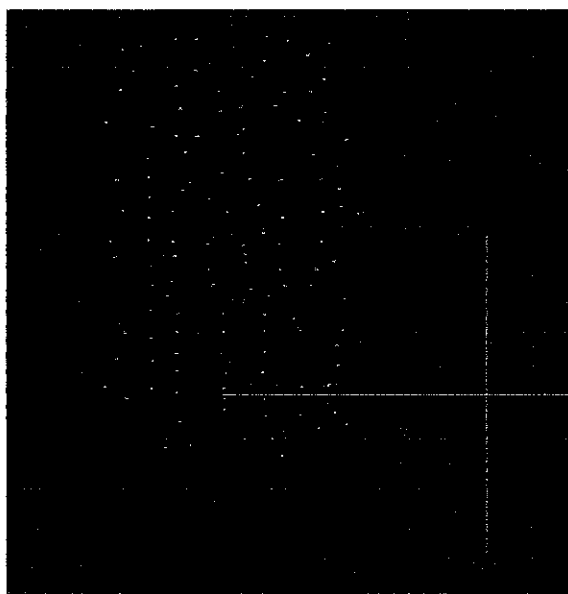


Figure 18: Dot cloud

3.4.3 Wired reconstruction

In 3D reconstruction after location points and construction of dot cloud, it's required to connect the points together. There are many methods and algorithms available such as Delaunay triangulation, shortest distance and etc. shortest distance make a dynamic connection between

points in such a way that it's possible to reduce/increase the connection of one points to other points. The result of this step is called wired structure *Figure 19*.

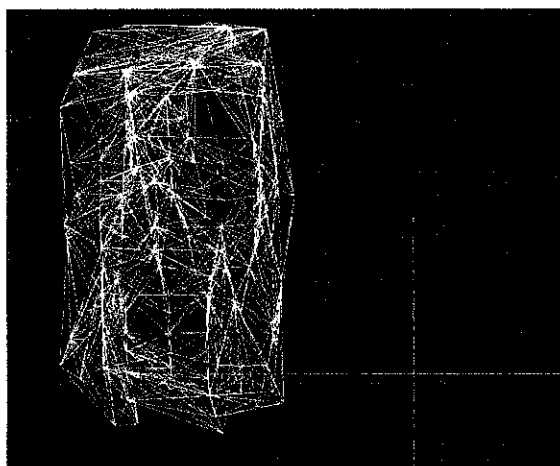


Figure 19: Wired structure

3.4.4 Solid reconstruction

The final steps of 3D reconstruction is to find a path between the connected points in such a way that the result polygons be a planner. To ensure the planarity of the polygons it suggested using triangles since the triangles are always a planner polygons. The result of generating this polygons is a shape with solid color and by adding light effect on that it's possible to see better result on the solid construction *Figure 20*.

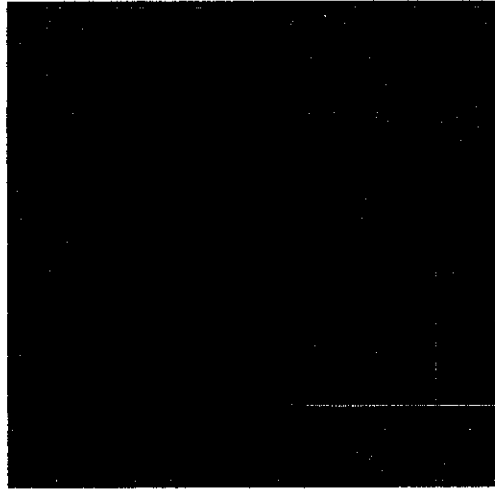


Figure 20: Solid Structure

3.5 Experimental system design

3.5.1 Class Diagram

The system is working on two important libraries. The first one is OpenCV and second one is OpenGL. But unfortunately these two classes have some clashes with each other so we need to separate them as much as possible. So we put each of them in separate classes. And to handle these two classes we put a base class. The base class which is considered as a main program calls the image processing (OpenCV) part first and then converts the variable and transfers to 3D display environment (OpenGL), shown in *Figure 21*.

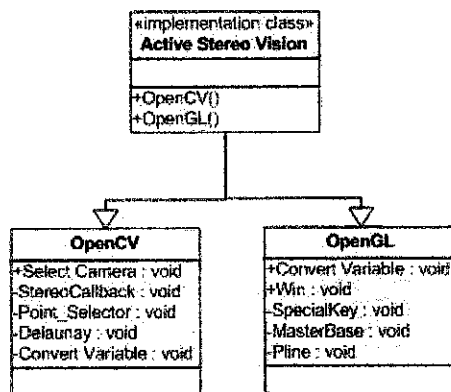


Figure 21: Active Stereo Vision Class Diagram

3.5.2 Flow chart

The flow of the main class is shown in *Figure 22*. Firstly, the image processing class will be called by the system and then data will be converted and transferred to OpenGL class. By displaying the result, the process will end but in each class an infinite loop will continue until user requests for program termination.

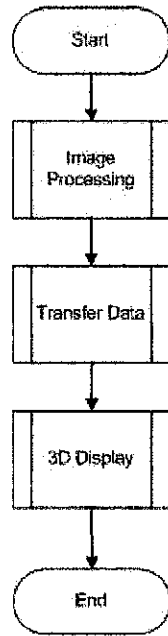


Figure 22: Main Class Flow chart

Now we start analyzing each class. The first class for this process is image processing. In image processing class, the first step is to ask user to select the camera and then send the image to the “stereoCallBack” function, from there this function handles the processes. First is preprocessing where the image will be divided to smaller area and then each area will be sent to the “pointSelector” function. This function first gets the first image and selects the points (objects) on it, then based on the object specification it searches on the second image. After specifying the objects with their correspondence it sends the points to Delaunay triangulation function to locate the points and connects them together. Next, the 2D construction will end as in *Figure 23*.

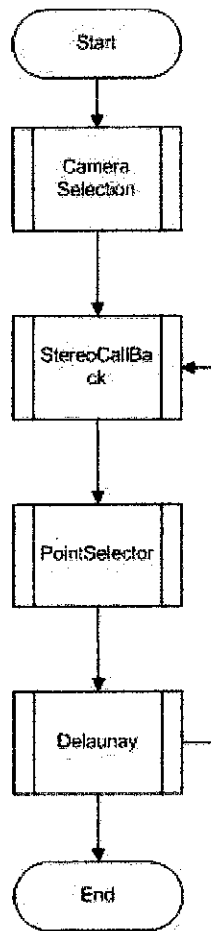


Figure 23: Image processing main flow chart

After processing on the image, it is time for transferring data to OpenGL. This step is very important and it works as translator for both library based on the original C++ class variables. So first the system translates from OpenGL to C++ variable type and then transfers it to graphical class and finally it transfers to OpenGL type, as in *Figure 24*.

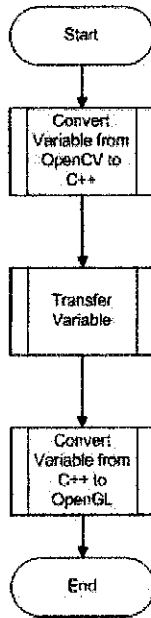


Figure 24: Transfer Flow Chart

Finally, to display the result we first need to create windows and then assign special key for specific tasks like scale the view or move to left or right, etc. After that it calculates the point's location and displays them in the environment as shown in *Figure 25*.

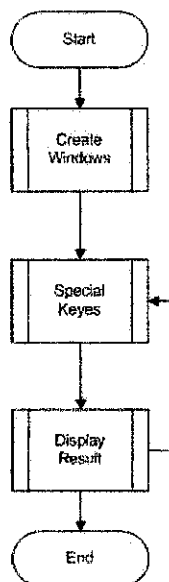


Figure 25: 3D Display

3.5.3 GUI

As it explained before, this system contains three main classes. Each of these classes has their own interface. We tried to simplify the design as much as possible so the user can simply use the system. The main class is basically has four button and a message display area. The functionality of each button will be explained more in the implementation, in *Figure 28*. The next interface is for camera selection. This interface is also very simple as shown in *Figure 29*. The requirement from user is clear and if user makes any mistake the system will inform user about that, shown in *Figure 26*.

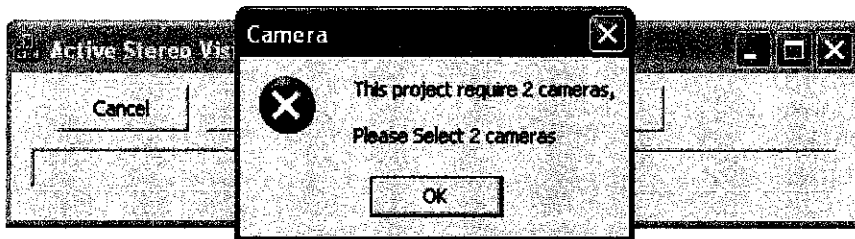


Figure 26: Error Message

Finally, we need a 3D display interface. It is very simple interface where the user can view the 3D result as in *Figure 27*.

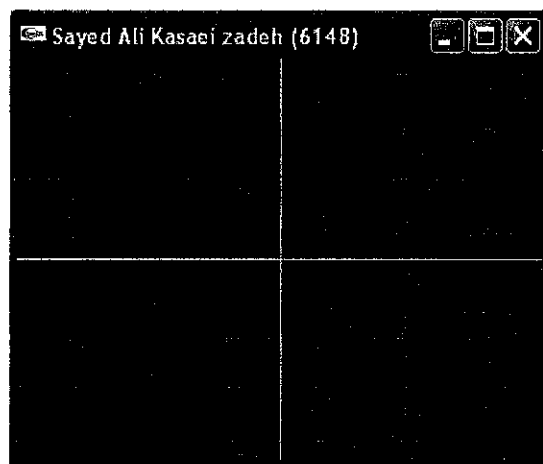


Figure 27: 3D Display

3.6 Experimental system Implementation

In this system there is one main file which handles two classes. The first class is related to image processing using computer vision library OpenCV. The second class is for computer graphic part. Here we explain how this system implemented. First we start the main file and then explain about each class.

3.6.1 Main file

This file (*Active Stereo VisionDlg.cpp*) which is created by Microsoft MFC contains the main interface where allows user specified the operations. There are 4 buttons on this interface. The first 2 buttons (Ok and Cancel) *Figure 28* is design to terminate the program. We explain later how this 2 buttons terminate each class base on class terminator. Also these 2 buttons after closing the classes will terminate the main file.

The next button is “Process” button that allows the user to select camera and do the process on the image. Finally the result button where allow user to view the processed result in 3D.



Figure 28: Main file interface

The main file is consisting of many sub functions where most of these function is predefine by Microsoft visual studio. For start it's require to call the include files.

```
#include "stdafx.h"
```

```
#include "Active Stereo Vision.h"
```

```
#include "Active Stereo VisionDlg.h"
```

```
#include "oc.h"
```

```
#include "og.h"
```

The 3 first headers are belonging to Microsoft. The “stdafx.h” is calling MFC class for implementation of interface. The “Active Stereo Vision.h” is for calling resources specified in the “Active Stereo VisionDlg.h”.

Finally the 2 remaining classes are the “oc.h” and “og.h” that we will explain later in detail about them. Here before any process is started is requiring making an instance of each class and calling their constructor. As follow:

```
oc *oci = new oc();  
og *ogi = new og();
```

the next step is to assign each button a class. first we start with the image processing class and assign it to the process button.

```
oci->camselect();
```

This function will allow user to select the cameras and then allow the class to process its required steps. After process finished the user can press the next button to follow the following steps.

```
oci->data_transfer();  
ogi->dc(oci->artoc,oci->cart,oci->vecoc,oci->veccoc);  
ogi->win();
```

The “data_transfer” function will convert all OpenCV variables to be understandable for OpenGL. The “dc” also allocates this data to OpenGL class. And finally by calling “win” function new 3D windows will created and displayed to user. These processed even though looks like simple but play very important role in this application since it translate the OpenCV variables to be understandable for OpenGL.

3.6.2 Image processing

Oc class handling the image processing part of this project. First it requires acquiring two cameras. This part handles by “highgui.h” header. The first step is to ensure the user have 2 cameras and in other case warn the user to apply two cameras for this system. This process done by “cvcamGetCamerasCount” the next step is asking user to select cameras from the list provided to user *Figure 29*. After selecting the cameras and assign them to their own specific

variables now is time to enable the camera property and start rendering. These 2 processes handle by “CVCAM_PROP_ENABLE” and “CVCAM_PROP_RENDER”. The next property need to set is which function needs to call and passes images received by each camera. This property set by “CVCAM_STEREO_CALLBACK” this property calls “void stereocallback(IplImage* Image,IplImage* Image2)” function and this is the main function in the image processing part. Finally by calling “cvcamInit()” the properties will assign to cameras and by “cvcamStart()” cameras will start capturing images. To ensure this process run continuously we apply and infinite loop as “while(1)” and this process will break and go to end by pressing ok or cancel button in main interface.

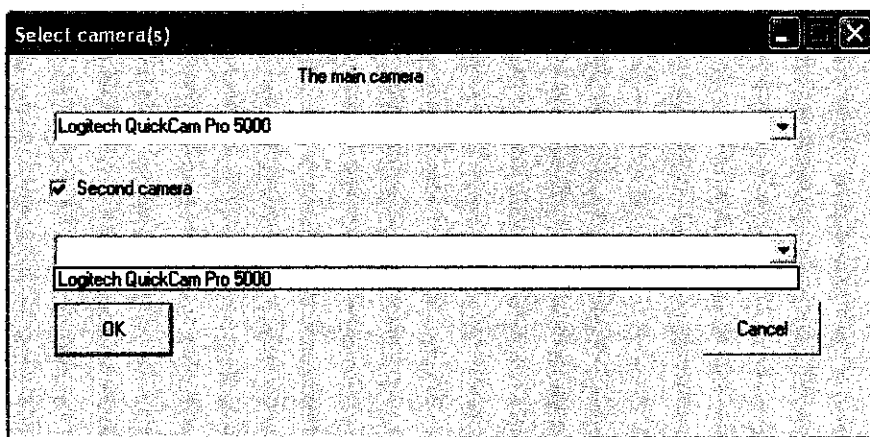


Figure 29: Camera Selection Form

In the “stereocallback” function first we divide the height of image to smaller division and the process the result on each divisions *Figure 30*. The next step is to initial points (objects) in the first image and then finds the objects in second image (find correspondence). This 2 process handle by “void pointselector(IplImage* frame)” as you can see this function get the image and then process on it. In this function set of points with their correspondence will set and displayed. The process of point selection will continue until all image sections processed and the points stores in specific array belong to them.



Figure 30: image division

“Pointselector” function is to process and extract objects and their corespondance. The first step is to initial the memory space and point arrays. Here to allocate point array we use dynamic array allocation to optimum the process. A sample of code will be as follow

```
CvPoint2D32f* points[3][8];  
points[0][0] = (CvPoint2D32f*)cvAlloc(MAX_COUNT*sizeof(points[0][0][0]));
```

this process will perform only one time since its consider as initialization. The function will check if the image is initials or not if its not initial it call the initial function and followed by the object selection. The object selection use a function from OpenCV liberary the sample code will be as follow:

```
cvGoodFeaturesToTrack( grey, eig, temp, points[1][icnt], &count,quality,  
min_distance, 0, 3, 0, 0.04 );  
  
cvFindCornerSubPix( grey, points[1][icnt], count,cvSize(win_size,win_size),  
cvSize(-1,-1),  
  
cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,10,0.03));
```

These functions first allocate the point on the first image and then base on the point’s neighbors it specified the best object to track in second image. After selecting the objects in first image the function will return and second image will send to the function to track the points. Here we use optical fellow (another OpenCV functions) to track the points in second image. A sample code is as follow:

```
cvCalcOpticalFlowPyrLK( prev_grey, grey, prev_pyramid,  
pyramid,points[0][icnt], points[1][icnt], count,  
cvSize(win_size,win_size), 3, status, 0,  
cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03), flags );
```

After selecting point in both image it’s require to remove extra points which help us to eliminate invisible views on either of images and then display result on the images *Figure 31*. After allocation objects in both images the function will return to original function and continue the process.

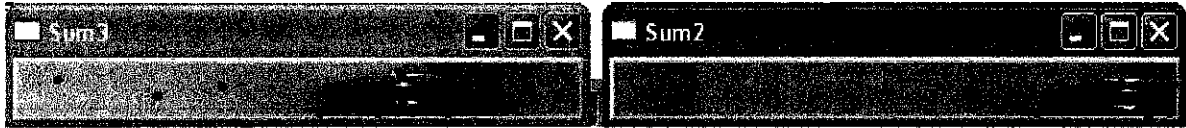


Figure 31: optical fellow and good Features to track result

The next step is connecting the point using Delaunay Triangulation Algorithm. This algorithm also implemented in OpenCV. For start there are few initialization such as memory allocation and also temporary images used in this algorithm. Then it starts reading the point one by one and base on Delaunay algorithm the points will connect to each other. For start we need to allocate the points on their own subdivision using following sample function:

```
cvSubdiv2DLocate( subdiv, fp, &e0, &p );
```

then the points will sent to Delaunay algorithm to connect to its neighbors using another function of OpenCV as following sample:

```
cvSubdivDelaunay2DInsert( subdiv, cvPoint2D32f(art[j][i][0],art[j][i][1]) );
```

the function will return the connection and the system will display the result to user *Figure 32*. This is the end of processing part. The next step is to transfer the calculation data and combine them all to gether and transfer to OpenGL.

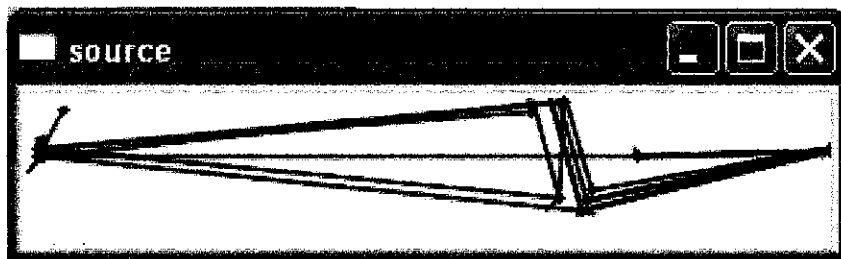


Figure 32: Delaunay Triangulation

After finishing the calculation and make the connection between objects now it's time for pass the process to the graphic library but before that there is one function need to implement to ensure when the user ask for program termination the process class release all the memory space it requires. It does require terminating all the windows created for displaying images and all memory resource it created. Sample of termination code is as follow:

```
cvReleaseImage(&imagep2); //closing image resources
```

```

cvDestroyAllWindows(); //closing all display image windows
cvcamStop( );        //stop the cameras
cvcamExit( );        //terminate camera resources

```

After ensuring all the resources will destroy on termination now its time to transfer data to a data type that OpenGL understand. For example in OpenCV “CvPoint2D32f” is defining points where it store the x and y position of the point so we need to store these type of variable in an array where for example “array[0]” holds “x” and “array[1]” hold the “y” value. There are many other similar data type is required that system need to convert before it use.

After conversion and transferring data to OpenGL environment now is time to construct 3D view. For time being this process only consist of points and connection between them in 3D mode. But before start it's require to specify certain criteria for displaying windows Like the name, size, accepting key for 3D manipulation and ETC. You can see the samples as follow:

```

glutInitWindowSize(320,240);           //size of windows
glutSpecialFunc(SpecialKeys);         //call function special key
glutDisplayFunc(myDisplay);           //call function myDisplay

```

And also special keys for manipulation:

```

if(key == GLUT_KEY_F6)
    xRot += .5f;

if(key == GLUT_KEY_F8)
    scale -= .05f;

if(key == GLUT_KEY_LEFT)
    xp -= .05f;

```

To ensure this variable is affecting the view we need to refresh the windows after each modification

```

glutPostRedisplay();

```


And finally by specifying the location of points we can connect them together in a 3D environment as you can see in the following sample:

```
dline(vecog[j][0][0],vecog[j][0][1],vecog[j][0][2]*10,vecog[j][1][0],vecog[j][1][1],vecog[j][1][2]*10);
```

```
void dline(double x,double y,double z,double x1,double y1,double z1)
```

and finally we can achieve to a result in 3D environment [36].

3.7 TOOLS

The project is developed using:

C++

Camera (webcam)

Open CV

Open GL

Visual Studio

CHAPTER 4

RESULT AND DISCUSSION

4.1 Depth calculation

As it discussed before to calculate depth of an object we can use the location of object in each image. Here we manually select the point to show how this formula can extract 3rd dimension from 2D images.

4.1.1 Simple object

First sample is one cup. The main point here is how to chose the points in first image and track them in the second image. For start we track the shape points on the cup *Figure 33*.

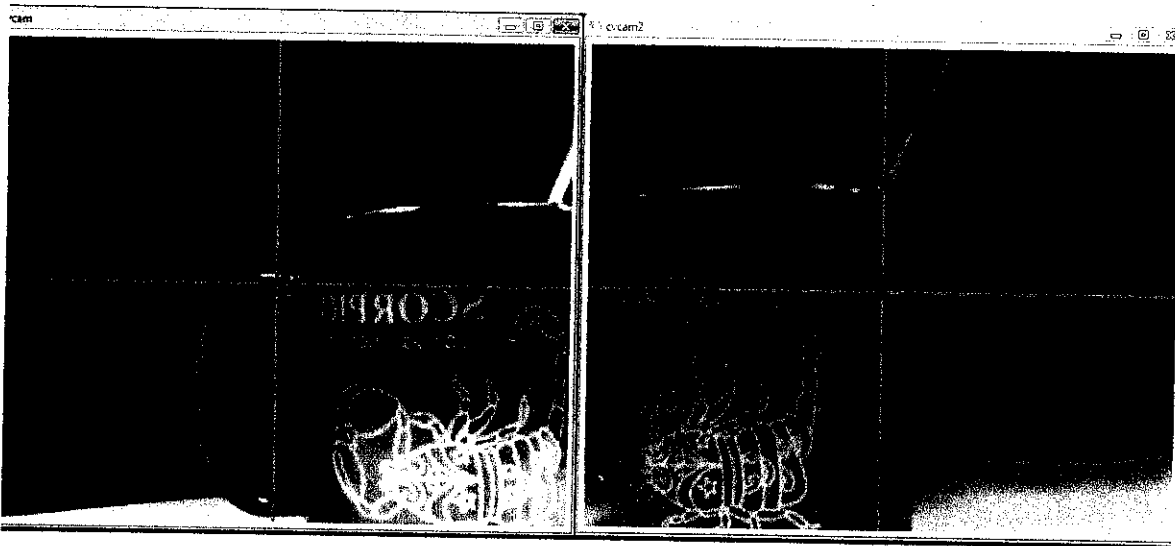


Figure 33: Points on Image

In (*Table 1*: Simple Object) the selected point's value and the result is stated in this part we use.

Table 1: Simple Object

Point No	1	2	3	4	5	6
B1	22.30	20.92	19.62	18.10	16.51	14.92
B2	8.57	7.02	5.47	4.16	2.72	1.80
a=B1-B2	13.73	13.90	14.15	13.94	13.79	13.12
Z	13.34	13.03	12.58	12.96	13.23	14.54

In this part the Q (distance of 2 camera is 28.5 centimeters and the camera value is 12.4.

Base on the result *Figure 34* you can see the point's depth shows some curve which is the cup curve. Even though the heights of the points are different but since the camera are in horizontal line this difference doesn't affect the calculation.

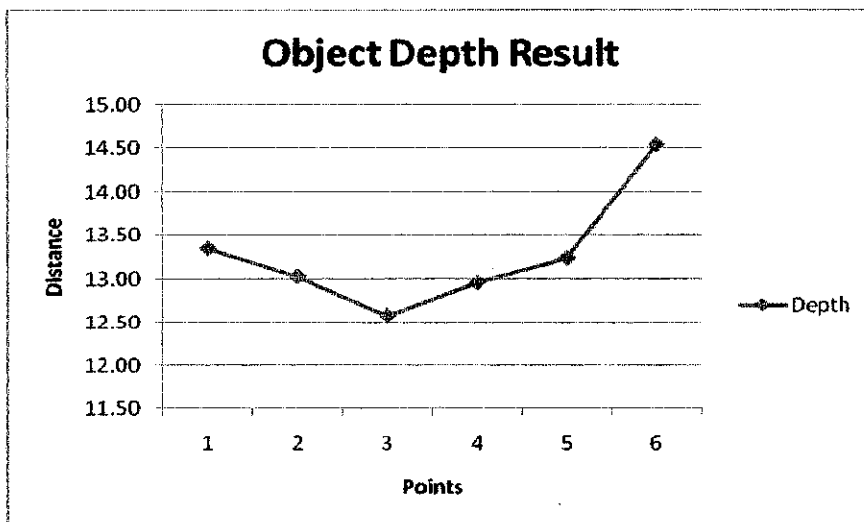


Figure 34: Object Depth Result

4.1.2 Transparent Bottle

In the second image it tries to show even if the selection is in transparency mod but still the result is correct. As you can see *Figure 35* the selection is a transparent bottle. Here tries to show the calculation is independent from object material.

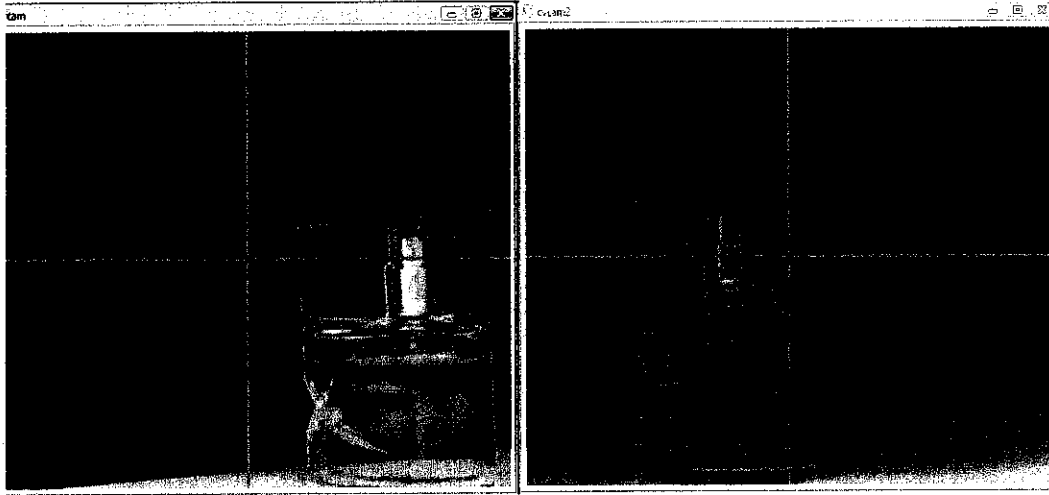


Figure 35: Transparent Bottle

Here also the value in *Table 2: Transparent Object Result* are the point position in first image, the second image, the difference of 2 and the final row shows the depth result.

Table 2: Transparent Object Result

Point No.	1	2	3	4	5	6	7	8
N	21.38	19.02	18.77	18.28	17.89	17.32	13.51	18.45
n'	12.07	9.49	9.17	8.64	8.22	7.73	4.55	8.96
a=n-n'	9.31	9.53	9.60	9.64	9.67	9.59	8.96	9.49
z	25.57	24.69	24.42	24.27	24.15	24.46	27.05	24.85

Here Q as distance of 2 cameras is 28.5 centimeters and the camera value (F) is 12.4 the result is in centimeters. Base on the result *Figure 36* you can see that the depth calculated respectfully to the object regardless of the object material.

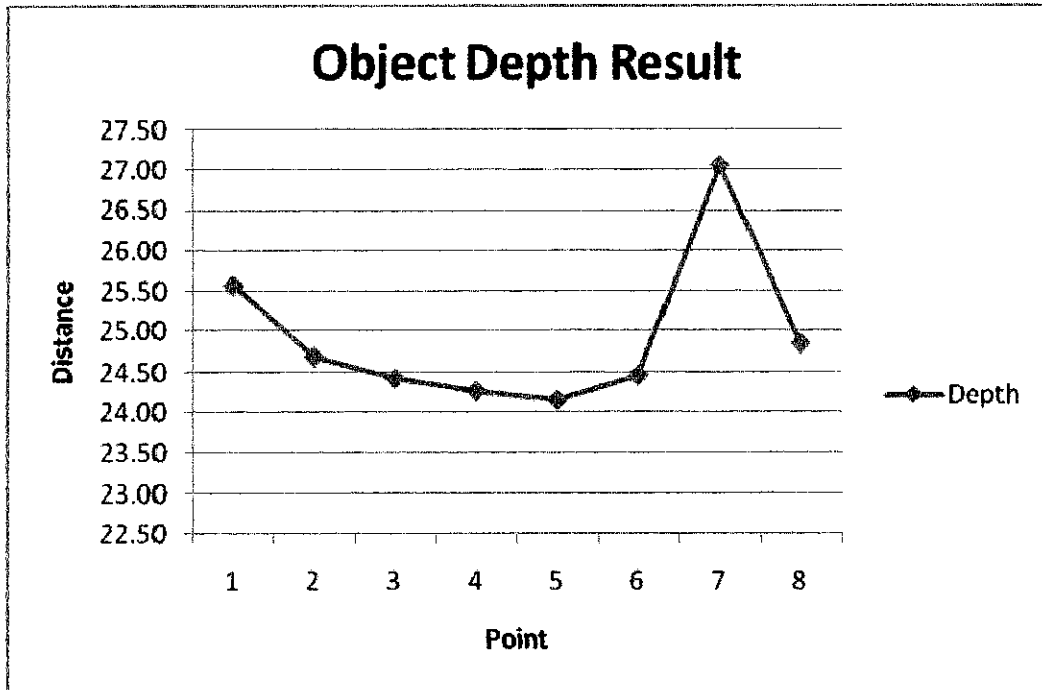


Figure 36: Transparent Object Depth Graph

4.2 Fining correspondence

For start an automated system it's require to select the base points on one image and then track them in second image. *Figure 37* is sample of this function result. Green dot on the image indicate the points found by the function. And the red dots on the second image are the points that corresponded to the first image points. As it's clear the points selection and their correspondence is on the image part that is contain texture.

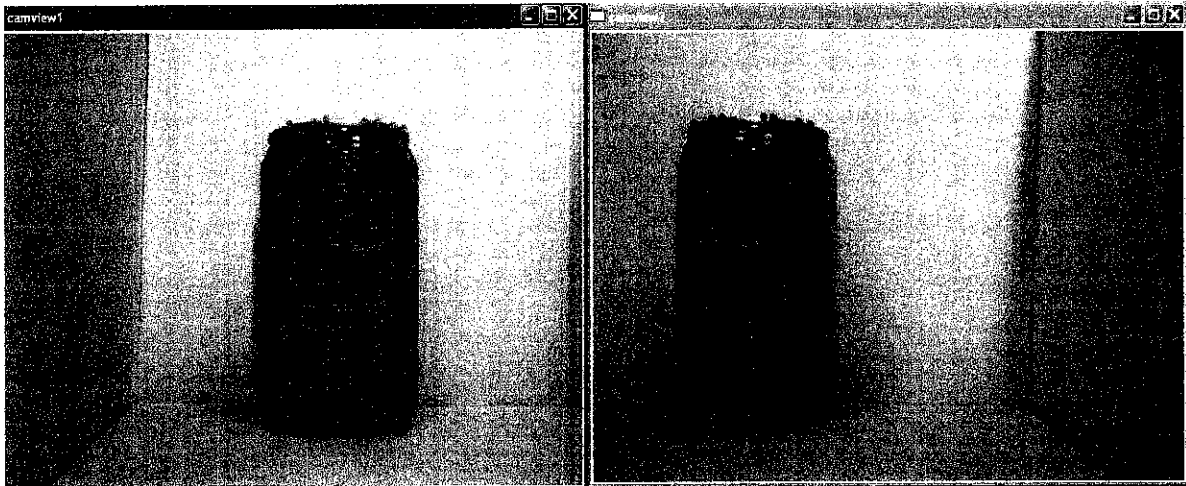


Figure 37: Sample of good feature to track

4.3 3D reconstruction

4.3.1 Dot cloud

The first outcome of this project from the correspondence result is dot cloud. But before that it's required to transfer result in 3D modeling environment and calculate depth. Then display result for farther process. As you can see the shape of the dot cloud is exactly similar to the curve on the can displayed in the image *Figure 38*.

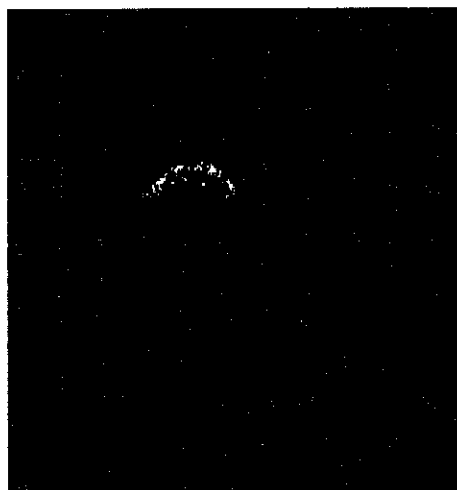


Figure 38: Dot cloud Result

4.3.2 *Wired result*

By connecting the points in dot cloud the wired structure is created. In 2 dimensions there are many algorithms where it's requiring to modify them to fit the 3D environment. Here the shortest distance is implemented *Figure 39*.

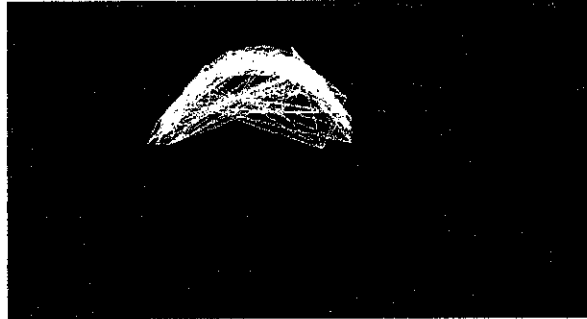


Figure 39: Wired Structure

4.3.3 *Solid result*

After generating the wired structure we can find triangles within the points and by connecting them together and form triangles and with help of lighting the solid result will be generated *Figure 40*.

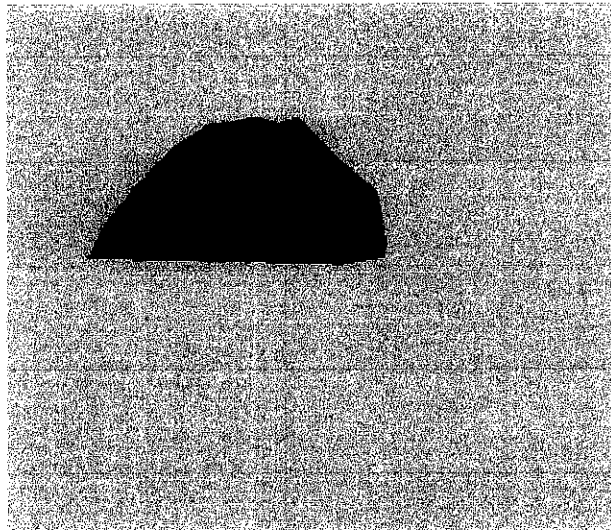


Figure 40: Solid Structure

CHAPTER 5

CONCLUSION

In this report it tries to show how computer can understand depth. There were sequences of tasks required to achieve fully automated system. In this project we use block matching algorithm to find correspondence of object where the result only limited to textured object. For future it suggested to use other algorithm or a mixture of algorithms to achieve better performance and results.

Also for wired structure the shortest path implemented. This algorithm is good because of the dynamic connection of the points but it's better to advance 2D algorithm to be implemented in 3D environment such as Delaunay triangulation or other methods.

The current result is based on only 2 images which is best case can return one side of the object. For future job it suggested a system with ability of generating fully 3D object sample. Also it's suggested to improve the solid structure by adding texture and generate a fully realistic 3D object in the computer.

CHAPTER 6

REFERENCES

- 1- Input Devices and Sensors for Virtual Environments (January 31st 2007)
<[http://wings.buffalo.edu/courses/sp07/mae/574-410/Course Notes/C5 and C6-Sensors and Input Devices.pdf](http://wings.buffalo.edu/courses/sp07/mae/574-410/Course%20Notes/C5%20and%20C6-Sensors%20and%20Input%20Devices.pdf)>
- 2- Govindarajan Srimathveeravalli 1998, "VR Lab"
- 3- Vision Group at SIRS Lab - University of Siena. 8 August 2007
<<http://sirslab.dii.unisi.it/vision/index1.htm>>
- 4- RSCC Volume 1 Module 7 - Stereoscopy and Height Measurement. 8 August 2007
<<http://www.r-s-c-c.org/rsc/v1m7.html>>
- 5- RSCC Volume 1 Module 7 - Stereoscopy and Height Measurement. 8 August 2007
<http://www.r-s-c-c.org/rsc/v1m7images/stereo_viewing_eyes_building.jpg>
- 6- What is Stereo Vision? 21 August 2007 <<http://www.vision3d.com/stereo.html>>
- 7- What is Stereo Vision? 21 August 2007 <<http://www.vision3d.com/images/bb.jpeg>>
- 8- Fundamentals of Stereo Computer 21 August 2007
<http://egweb.mines.edu/faculty/tvincent/Welding/fundamentals_of_stereo_computer.htm>
- 9- D. Scharstein and R. Szeliski, May 2002 "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," Int'l J. Computer Vision, vol. 47, no. 1, pp. 7-42.
- 10- S. Roy, 1999 "Stereo without Epipolar Lines: A Maximum-Flow Formulation," Int'l J. Computer Vision, vol. 34, nos. 2/3, pp. 147- 161.

- 11-H. Ishikawa and D. Geiger, 1998 "Occlusions, Discontinuities, and Epipolar Lines in Stereo," Proc. European Conf. Computer Vision, pp. 232-249.
- 12-Y. Boykov, O. Veksler, and R. Zabih, 2001, "Fast Approximate Energy Minimization Via Graph Cuts," Proc. IEEE Int'l Conf. Computer Vision, vol. 1, pp. 532-539.
- 13-J. Sun, N.-N. Zheng, and H.-Y. Shum, July 2003, "Stereo Matching Using Belief Propagation," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 25, no. 7, pp. 787-800.
- 14-V. Kolmogorov and R. Zabih, 2001, "Computing Visual Correspondence with Occlusions Using Graph Cuts," Proc. IEEE Int'l Conf. Computer Vision.
- 15-H. Tao, H.S. Sawhney, and R. Kumar, 2001, "A Global Matching Framework for Stereo Computation," Proc. IEEE Int'l Conf. Computer Vision, vol. 1, pp. 532-539.
- 16-L. Hong and G. Chen, 2004, "Segment-Based Stereo Matching Using Graph Cuts," Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition, vol. 1.
- 17-Y. Wei and L. Quan, 2004, "Region-Based Progressive Stereo Matching," Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition, vol. 1, pp. 106-113.
- 18-M. Bleyer and M. Gelautz, Jan. 2005, "Graph-Based Surface Reconstruction from Stereo Pairs Using Image Segmentation," Proc. SPIE, vol. 5656.
- 19-Y. Zhang and C. Kambhamettu, 2002, "Stereo Matching with Segmentation-Based Cooperation," Proc. European Conf. Computer Vision, pp. 556-571.
- 20-C. Baillard and H. Mar'stre, Dec. 1999, "3-D Reconstruction of Urban Scenes from Aerial Stereo Imagery: A Focusing Strategy," Computer Vision and Image Understanding, vol. 76, no. 3, pp. 244-258.
- 21-G. Egnal and R.P. Wildes, Aug. 2002, "Detecting Binocular Half-Occlusions: Empirical Comparisons of Five Approaches," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 24, no. 8, pp. 1127-1133.

- 22- C.L. Zitnic and T. Kanade, July 2000, "A Cooperative Algorithm for Stereo Matching and Occlusion Detection," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 22, no. 7, pp. 675-684.
- 23- J. Sun, Y. Li, S.-B. Kang, and H.-Y. Shum, June 2005, "Symmetric Stereo Matching for Occlusion Handling," Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition, vol. 2, pp. 399-406.
- 24- A.S. Ogale and Y. Aloimonos, 2004, "Stereo Correspondence with Slanted Surface: Critical Implication of Horizontal Slant," Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition, vol. 1, pp. 568-573.
- 25- M.Z. Brown, D. Burschka, and G.D. Hager, Aug. 2003, "Advances in Computational Stereo," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 25, no. 8, pp. 993-1008.
- 26- DigitalCamera2, 3rd April 2008,
< <http://earthenterprises1.com/Digital%20Cameras2.htm>>
- 27- Embedded Technology – Camera, 3rd April 2008,
< <http://www.bluewatersys.com/design/technology/camera.php>>
- 28- Digital Photography Essentials #003 @Digital Outback Photo 3rd April 2008,
< http://www.outbackphoto.com/dp_essentials/dp_essentials_03/essay.html>
- 29- Fundamentals of Stereo Computer, 3rd April 2008, <
http://egweb.mines.edu/faculty/tvincent/Welding/fundamentals_of_stereo_computer.htm>
- 30- Open CV reference opencvref_cv.htm cvGoodFeaturesToTrack
- 31- Computational Stereo Vision Using Color, IEEE, 1988
- 32- Gang Xu and Zhengyou Zhang. , 1996, "Epipolar Geometry in Stereo, Motion and Object Recognition", 1st edition, Kluwer Academic Publishers.
- 33- Emanuele Trucco and Alessandro Verri, 1998, "Introductory Techniques for 3-D computer Vision", 1st edition, Prentice-Hall, Inc.

34- Rafael C. Gonzalez and Richard E. Wood, 2002, "Digital Image Processing", 2nd Edition, Prentice-Hall, Inc.

35- Richard S. Wright, Jr and Benjamin Lipchak, 2005, "OpenGL Supper Bible", 3rd edition, Sams Publishing.

36- GlauCAD: Workplan, 3rd April 2008, < <http://www.msr.uni-bremen.de/glaucad/workplan.html>>

APPENDIX I

1) Schedule FYP I

Table 3: Schedule and milestone FYP I

No.	Detail/week	1	2	3	4	5	6	7		8	9	10	11	12	13	14	
1	Selection of project Topic								Mid Semester Break								
2	Preliminary Research Work																
3	Submission of Preliminary Report				⊗												
4	Seminar 1 (optional)																
5	Project Work																
6	Submission of Progress Report										⊗						
7	Seminar 2 (compulsory)																
8	Project work continues																
9	Submission of Interim Report Final Draft																⊗
10	Oral Presentation																⊗

2) Schedule FYP II

Table 4: Schedule and milestone FYP II

No.	Detail/week	1	2	3	4	5	6	7		8	9	10	11	12	13	14	
1	Project work continue	■	■	■					Mid Semester Break								
2	Submission of progress report 1				●												
3	Project work continue				■	■	■	■									
4	Submission of progress report 2										●						
5	Seminar											■	■	■			
6	Project work continue										■	■	■	■			
7	Pre EDX												●				
8	Dissertation														●		
9	Oral Presentation															●	
10	Hard Bound Project submission																●

APPENDIX II

1) Main class header:

```
1. // Active Stereo VisionDlg.h : header file
2. //
3.
4. #pragma once
5.
6.
7. // CActiveStereoVisionDlg dialog
8. class CActiveStereoVisionDlg : public CDialog
9. {
10. // Construction
11. public:
12.     CActiveStereoVisionDlg(CWnd* pParent = NULL); // standard
        constructor
13.
14. // Dialog Data
15.     enum { IDD = IDD_ACTIVESTEREOVISION_DIALOG };
16.
17.     protected:
18.     virtual void DoDataExchange(CDataExchange* pDX); //
        DDX/DDV support
19.
20.
21. // Implementation
22. protected:
23.     HICON m_hIcon;
24.
25.     // Generated message map functions
26.     virtual BOOL OnInitDialog();
27.     afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
28.     afx_msg void OnPaint();
29.     afx_msg HCURSOR OnQueryDragIcon();
30.     DECLARE_MESSAGE_MAP()
31. public:
32.     afx_msg void OnEnChangeEdit1();
33. public:
34.     afx_msg void OnBnClickedButton1();
35. public:
36.     afx_msg void OnBnClickedOk();
37. public:
38.     afx_msg void OnBnClickedCancel();
39. public:
40.     afx_msg void OnBnClickedButton2();
41.};
```

2) Main Class source:

```
1. // Active Stereo VisionDlg.cpp : implementation file
2. //
3.
4. #include "stdafx.h"
5. #include "Active Stereo Vision.h"
6. #include "Active Stereo VisionDlg.h"
7. #include "oc.h"
8. #include "og.h"
9.
10.     #ifdef _DEBUG
11.     #define new DEBUG_NEW
12.     #endif
13.
14.
15.     // CAboutDlg dialog used for App About
16.
17.     class CAboutDlg : public CDialog
18.     {
19.     public:
20.         CAboutDlg();
21.
22.     // Dialog Data
23.         enum { IDD = IDD_ABOUTBOX };
24.
25.     protected:
26.         virtual void DoDataExchange(CDataExchange* pDX);    //
DDX/DDV support
27.
28.     // Implementation
29.     protected:
30.         DECLARE_MESSAGE_MAP()
31.     };
32.
33.     CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
34.     {
35.     }
36.
37.     void CAboutDlg::DoDataExchange(CDataExchange* pDX)
38.     {
39.         CDialog::DoDataExchange(pDX);
40.     }
41.
42.     BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
43.     END_MESSAGE_MAP()
44.
45.
46.     // CActiveStereoVisionDlg dialog
47.
48.
49.
50.
51.     CActiveStereoVisionDlg::CActiveStereoVisionDlg(CWnd* pParent
/*=NULL*/)
52.         : CDialog(CActiveStereoVisionDlg::IDD, pParent)
```



```

53.     {
54.         m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
55.     }
56.
57. void CActiveStereoVisionDlg::DoDataExchange(CDataExchange* pDX)
58. {
59.     CDialog::DoDataExchange(pDX);
60. }
61.
62. BEGIN_MESSAGE_MAP(CActiveStereoVisionDlg, CDialog)
63.     ON_WM_SYSCOMMAND()
64.     ON_WM_PAINT()
65.     ON_WM_QUERYDRAGICON()
66.     //}AFX_MSG_MAP
67.     ON_EN_CHANGE(IDC_EDIT1,
&CActiveStereoVisionDlg::OnEnChangeEdit1)
68.     ON_BN_CLICKED(IDC_BUTTON1,
&CActiveStereoVisionDlg::OnBnClickedButton1)
69.     ON_BN_CLICKED(IDOK, &CActiveStereoVisionDlg::OnBnClickedOk)
70.     ON_BN_CLICKED(IDCANCEL,
&CActiveStereoVisionDlg::OnBnClickedCancel)
71.     ON_BN_CLICKED(IDC_BUTTON2,
&CActiveStereoVisionDlg::OnBnClickedButton2)
72. END_MESSAGE_MAP()
73.
74.
75. // CActiveStereoVisionDlg message handlers
76.
77. BOOL CActiveStereoVisionDlg::OnInitDialog()
78. {
79.     CDialog::OnInitDialog();
80.
81.     // Add "About..." menu item to system menu.
82.
83.     // IDM_ABOUTBOX must be in the system command range.
84.     ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
85.     ASSERT(IDM_ABOUTBOX < 0xF000);
86.
87.     CMenu* pSysMenu = GetSystemMenu(FALSE);
88.     if (pSysMenu != NULL)
89.     {
90.         CString strAboutMenu;
91.         strAboutMenu.LoadString(IDS_ABOUTBOX);
92.         if (!strAboutMenu.IsEmpty())
93.         {
94.             pSysMenu->AppendMenu(MF_SEPARATOR);
95.             pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
96.         }
97.     }
98.
99.     // Set the icon for this dialog. The framework does this
automatically
100.    // when the application's main window is not a dialog
101.    SetIcon(m_hIcon, TRUE);           // Set big icon
102.    SetIcon(m_hIcon, FALSE);        // Set small icon
103.

```

```

104.         // TODO: Add extra initialization here
105.
106.         return TRUE; // return TRUE unless you set the focus to a
        control
107.     }
108.
109.     void CActiveStereoVisionDlg::OnSysCommand(UINT nID, LPARAM
        lParam)
110.     {
111.         if ((nID & 0xFFFF) == IDM_ABOUTBOX)
112.         {
113.             CAboutDlg dlgAbout;
114.             dlgAbout.DoModal();
115.         }
116.         else
117.         {
118.             CDialog::OnSysCommand(nID, lParam);
119.         }
120.     }
121.
122.     // If you add a minimize button to your dialog, you will need the
        code below
123.     // to draw the icon. For MFC applications using the
        document/view model,
124.     // this is automatically done for you by the framework.
125.
126.     void CActiveStereoVisionDlg::OnPaint()
127.     {
128.         if (IsIconic())
129.         {
130.             CPaintDC dc(this); // device context for painting
131.
132.             SendMessage(WM_ICONERASEBKGND,
                reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);
133.
134.             // Center icon in client rectangle
135.             int cxIcon = GetSystemMetrics(SM_CXICON);
136.             int cyIcon = GetSystemMetrics(SM_CYICON);
137.             CRect rect;
138.             GetClientRect(&rect);
139.             int x = (rect.Width() - cxIcon + 1) / 2;
140.             int y = (rect.Height() - cyIcon + 1) / 2;
141.
142.             // Draw the icon
143.             dc.DrawIcon(x, y, m_hIcon);
144.         }
145.         else
146.         {
147.             CDialog::OnPaint();
148.         }
149.     }
150.
151.     // The system calls this function to obtain the cursor to display
        while the user drags
152.     // the minimized window.
153.     HCURSOR CActiveStereoVisionDlg::OnQueryDragIcon()
154.     {

```

```

155.         return static_cast<HCURSOR>(m_hIcon);
156.     }
157.
158.
159.     void CActiveStereoVisionDlg::OnEnChangeEdit1()
160.     {
161.         // TODO: If this is a RICHEDIT control, the control will
not
162.         // send this notification unless you override the
CDialog::OnInitDialog()
163.         // function and call CRichEditCtrl().SetEventMask()
164.         // with the ENM_CHANGE flag ORed into the mask.
165.
166.         // TODO: Add your control notification handler code here
167.     }
168.     oc *oci = new oc();
169.     og *ogi = new og();
170.     void CActiveStereoVisionDlg::OnBnClickedButton1()
171.     {
172.
173.         oci->camselect();
174.
175.         // TODO: Add your control notification handler code here
176.     }
177.
178.     void CActiveStereoVisionDlg::OnBnClickedOk()
179.     {
180.         // TODO: Add your control notification handler code here
181.         oci->stopit = 1;
182.         oci->final();
183.         ogi->~og();
184.         OnOK();
185.     }
186.
187.     void CActiveStereoVisionDlg::OnBnClickedCancel()
188.     {
189.         // TODO: Add your control notification handler code here
190.         oci->stopit = 1;
191.         oci->final();
192.         ogi->~og();
193.         delete ogi;
194.         OnCancel();
195.     }
196.
197.     void CActiveStereoVisionDlg::OnBnClickedButton2()
198.     {
199.         oci->pntt();
200.         //ogi->dc();
201.         ogi->win(oci->pos,oci->posc);
202.
203.
204.
205.         // TODO: Add your control notification handler code here
206.     }

```

3) Image processing class header:

```
42. #include <stdio.h>
43. #include <math.h>
44. #include <time.h>
45. #include "cv.h" // include core library interface
46. #include "highgui.h" // include GUI library interface
47. #include "cvcam.h"
48. #include "cxcore.h"
49.
50. #pragma once
51.
52. class oc
53. {
54. private:
55.     int ncams,cam1,cam2,nselected;
56.     int* out;
57.     char tstc[300];
58.
59.     IplImage* img2;
60.     char c;
61. public:
62.     int pos[2][500][3];
63.     int posc;
64.     oc(void);
65.     int stopit;
66.     void pntt();
67.     void final();
68.     //void houghlines(IplImage* src2);
69.     void camselect();
70. public:
71.     virtual ~oc(void);
72. };
```

4) Image processing class source:

```
1. #include "StdAfx.h"
2. #include "oc.h"
3. IplImage* Im, * Im1,*imagep,*imagep2;
4. IplImage *image = 0, *grey = 0, *prev_grey = 0, *pyramid = 0,
   *prev_pyramid = 0, *swap_temp;
5. CvPoint2D32f* points[3][8] = {0,0}, *swap_points;
6. const int MAX_COUNT = 500;
7. int ilo2 =1,flags = 0,count = 0,win_size = 10,need_to_init = 2,i, k,
   c,add_remove_pt = 0,pcount = 0;
8. char* status = 0;
9. int sy,icnt,inj;
10. CvPoint pt;
11. char ctv[100];
12. int setv=0,omk=0,omk2=0;
13. int omx[500][3],omx2[500][3];
14. FILE *ri,*li;
```

```

15.  errno_t err;
16.  void save_point(char rin[],char lin[]){
17.      int i;
18.      if( (err = fopen_s( &ri, rin, "w" )) !=0 )
19.          MessageBox(NULL,"Warning!\nCannot open the right
image file","File Manager",MB_ICONERROR);
20.      if( (err = fopen_s( &li, lin, "w" )) !=0 )
21.          MessageBox(NULL,"Warning!\nCannot open the left image
file","File Manager",MB_ICONERROR);
22.      if(rin){
23.          for(i=0;i<omk;i++)
24.              fprintf(ri,"%d\t%d\t%d\n",omx[i][0],omx[i][1],omx[i][2]);
                //(pns,1024,pn);
25.          }
26.      if(lin){
27.          for(i=0;i<omk2;i++)
28.              fprintf(li,"%d\t%d\t%d\n",omx2[i][0],omx2[i][1],omx2[i][2]);
                //(pns,1024,pn);
29.          }
30.      if(ri)
31.          {
32.              if ( fclose( ri ) )
33.                  {
34.                      MessageBox(NULL,"Warning!\nCannot close the right image
file","File Manager",MB_ICONERROR);
35.                  }
36.          }
37.      if(li)
38.          {
39.              if ( fclose( li ) )
40.                  {
41.                      MessageBox(NULL,"Warning!\nCannot close the left image
file","File Manager",MB_ICONERROR);
42.                  }
43.          }
44.      }
45.  void Text_On_Image(IplImage* img,char text[],CvPoint
Start_Pnt,CvScalar color)
46.  {
47.      CvFont font1;
48.      cvInitFont(&font1,CV_FONT_HERSHEY_COMPLEX,0.4,0.4,0,
1,CV_AA);
49.      cvPutText(img,text,Start_Pnt,&font1,color);
50.  }
51.  void copy2img(char rin[],char lin[]){
52.      int i=1;
53.      if( (err = fopen_s( &ri, rin, "r" )) !=0 )
54.          MessageBox(NULL,"Warning!\nCannot open the right
image file","File Manager",MB_ICONERROR);
55.      if( (err = fopen_s( &li, lin, "r" )) !=0 )
56.          MessageBox(NULL,"Warning!\nCannot open the left image
file","File Manager",MB_ICONERROR);
57.      if(rin){
58.          i=0;
59.          while(! feof(ri)){

```

```

60.     fscanf(ri, "%d\t%d\t%d\n", &omx[i][0], &omx[i][1], &omx[i][2]);
61.     cvCircle(imagep, cvPoint(omx[i][0], omx[i][1]), 3, CV_RGB(0, 255, 0), -
1, CV_AA, 0);
62.         i++; //(pns, 1024, pn);
63.     }
64.     omk = i-1;
65. }
66.     if(lin){
67.         //for(i=0;i<10;i++)
68.         i=0;
69.         while(! feof(li)){
70.
71.             fscanf(li, "%d\t%d\t%d\n", &omx2[i][0], &omx2[i][1], &omx2[i][2]);
72.             cvCircle(imagep2, cvPoint(omx2[i][0], omx2[i][1]), 3, CV_RGB(255, 0, 0)
, -1, CV_AA, 0);
73.                 i++; //(pns, 1024, pn);
74.             }
75.             omk2 = i-1;
76.         }
77.         cvShowImage("camview1", imagep);
78.         cvShowImage("camview2", imagep2);
79.         if(ri)
80.         {
81.             if ( fclose( ri ) )
82.             {
83.                 MessageBox(NULL, "Warning!\nCannot close the right image
file", "File Manager", MB_ICONERROR);
84.             }
85.             if(li)
86.             {
87.                 if ( fclose( li ) )
88.                 {
89.                     MessageBox(NULL, "Warning!\nCannot close the left image
file", "File Manager", MB_ICONERROR);
90.                 }
91.             }
92.         }
93.     void oc::pntt(){
94.         int i, j;
95.         if(omk>omk2)
96.             j = omk2;
97.         else
98.             j = omk;
99.         for (i=0;i<j;i++){
100.             pos[0][i][0] = omx[i+1][0];
101.             pos[0][i][1] = imagep->height - omx[i+1][1];
102.             pos[0][i][2] = omx[i+1][2]*4;
103.             pos[1][i][0] = omx2[i+1][0];
104.             pos[1][i][1] = imagep->height - omx2[i+1][1];
105.             pos[1][i][2] = omx2[i+1][2]*4;
106.         }
107.         posc = j;
108.     }

```

```

109.
110. void on_mouse( int event, int x, int y, int flags, void* param )
111. {
112.     //MessageBox(NULL,"onmouse","Camera",MB_OK);
113.     if (flags == 2){
114.         if(setv == 0){
115.             omk++;
116.             omx[omk][0]=x;
117.             omx[omk][1]=y;
118.             if(omk<=omk2){
119.                 omx[omk][2]=omx[omk][0]-omx2[omk][0];
120.                 omx2[omk][2]=omx[omk][0]-omx2[omk][0];
121.
122.                 sprintf(ctv,"%4.2f",3023.622047244/(float)omx[omk][2]);
123.                 Text_On_Image(imagep,ctv,cvPoint(x+1,y+1),CV_RGB(0,255,0));
124.                 Text_On_Image(imagep2,ctv,cvPoint(omx2[omk][0]+1,omx2[omk][1]+1),
125.                 CV_RGB(255,0,0));
126.                 cvShowImage("camview2",imagep2);
127.             }
128.             cvCircle(imagep,cvPoint(x,y),3,CV_RGB(0,255,0),-1,CV_AA,0);
129.             cvShowImage("camview1",imagep);
130.             setv = 1;
131.         }
132.     }else
133.         setv = 0;
134. }
135. void undo(){
136.     int i;
137.     if(omk2>omk){
138.         omk2--;
139.         omk--;
140.         for(i=omk;i>=0;i--){
141.             cvCircle(imagep,cvPoint(omx[i][0],omx[i][1]),3,CV_RGB(0,255,0),-
142.             1,CV_AA,0);
143.             cvCircle(imagep2,cvPoint(omx2[i][0],omx2[i][1]),3,CV_RGB(255,0,0)
144.             ,-1,CV_AA,0);
145.             sprintf(ctv,"%4.2f",3023.622047244/(float)omx2[i][2]);
146.             Text_On_Image(imagep2,ctv,cvPoint(omx2[i][0]+1,omx2[i][1]+1),CV_R
147.             GB(255,0,0));
148.             Text_On_Image(imagep,ctv,cvPoint(omx[i][0]+1,omx[i][1]+1),CV_RGB(
149.             0,255,0));
150.             cvShowImage("camview1",imagep);
151.             cvShowImage("camview2",imagep2);
152.         }
153.     }
154.     for(i=omk+1;i<omk2+1;i++){
155.         omx2[i][0] = omx2[i+1][0];
156.         omx2[i][1] = omx2[i+1][1];

```

```

150.     cvCircle(imagep2,cvPoint(omx2[i][0],omx2[i][1]),3,CV_RGB(255,0,0)
    ,-1,CV_AA,0);
151.         cvShowImage("camview1",imagep);
152.         cvShowImage("camview2",imagep2);
153.     }
154.
155.     }else{
156.         omk--;
157.         omk2--;
158.         for(i=omk2;i>=0;i--){
159.             cvCircle(imagep,cvPoint(omx[i][0],omx[i][1]),3,CV_RGB(0,255,0),-
    1,CV_AA,0);
160.             cvCircle(imagep2,cvPoint(omx2[i][0],omx2[i][1]),3,CV_RGB(255,0,0)
    ,-1,CV_AA,0);
161.             sprintf(ctv,"%4.2f",3023.622047244/(float)omx2[i][2]);
162.             Text_On_Image(imagep2,ctv,cvPoint(omx2[i][0]+1,omx2[i][1]+1),CV_R
    GB(255,0,0));
163.             Text_On_Image(imagep,ctv,cvPoint(omx[i][0]+1,omx[i][1]+1),CV_RGB(
    0,255,0));
164.             cvShowImage("camview1",imagep);
165.             cvShowImage("camview2",imagep2);
166.         }
167.         for(i=omk2+1;i<omk+1;i++){
168.             omx[i][0] = omx[i+1][0];
169.             omx[i][1] = omx[i+1][1];
170.             cvCircle(imagep,cvPoint(omx[i][0],omx[i][1]),3,CV_RGB(0,255,0),-
    1,CV_AA,0);
171.             cvShowImage("camview1",imagep);
172.             cvShowImage("camview2",imagep2);
173.         }
174.     }
175. }
176. void on_mouse2( int event, int x, int y, int flags, void* param )
177. {
178.     if (flags == 2){
179.         if(setv == 0){
180.             omk2++;
181.             omx2[omk2][0]=x;
182.             omx2[omk2][1]=y;
183.             if(omk2<=omk){
184.                 omx[omk2][2]=omx[omk2][0]-omx2[omk2][0];
185.                 omx2[omk2][2]=omx[omk2][0]-omx2[omk2][0];
186.                 sprintf(ctv,"%4.2f",3023.622047244/(float)omx2[omk2][2]);
187.                 Text_On_Image(imagep2,ctv,cvPoint(x+1,y+1),CV_RGB(255,0,0));
188.                 Text_On_Image(imagep,ctv,cvPoint(omx[omk2][0]+1,omx[omk2][1]+1),C
    V_RGB(0,255,0));
189.                 cvShowImage("camview1",imagep);

```



```

190.         }
191.
192.     cvCircle(imagep2,cvPoint(x,y),3,CV_RGB(255,0,0),-1,CV_AA,0);
193.         cvShowImage("camview2",imagep2);
194.         setv = 1;
195.     }
196.     }else
197.         setv = 0;
198. }
199. void presetup(IplImage* frame){
200.     /* allocate all the buffers */
201.     image = cvCreateImage( cvGetSize(frame), 8, 3 );
202.     image->origin = frame->origin;
203.     grey = cvCreateImage( cvGetSize(frame), 8, 1 );
204.     prev_grey = cvCreateImage( cvGetSize(frame), 8, 1 );
205.     pyramid = cvCreateImage( cvGetSize(frame), 8, 1 );
206.     prev_pyramid = cvCreateImage( cvGetSize(frame), 8, 1 );
207.     int jk=0;
208.     for(jk=0;jk<8;jk++){
209.         points[0][jk] =
210.             (CvPoint2D32f*)cvAlloc(MAX_COUNT*sizeof(points[0][0][jk]));
211.         points[1][jk] =
212.             (CvPoint2D32f*)cvAlloc(MAX_COUNT*sizeof(points[0][0][jk]));
213.         points[2][jk] =
214.             (CvPoint2D32f*)cvAlloc(MAX_COUNT*sizeof(points[0][0][jk]));
215.     }
216.     status = (char*)cvAlloc(MAX_COUNT);
217.     flags = 0;
218. }
219. void initpoints(){
220.     /* automatic initialization */
221.     IplImage* eig = cvCreateImage( cvGetSize(grey), 32, 1 );
222.     IplImage* temp = cvCreateImage( cvGetSize(grey), 32, 1 );
223.     double quality = 0.0001;
224.     double min_distance = 1;
225.     count = MAX_COUNT;
226.     cvGoodFeaturesToTrack( grey, eig, temp, points[1][icnt],
227.         &count,quality, min_distance, 0, 3, 0, 0.04 );
228.     cvFindCornerSubPix( grey, points[1][icnt],
229.         count,cvSize(win_size,win_size), cvSize(-1,-1),
230.
231.
232.
233.
234.
235.
236.
237.
238.
239.
240.
241.
242.
243.
244.
245.
246.
247.
248.
249.
250.
251.
252.
253.
254.
255.
256.
257.
258.
259.
260.
261.
262.
263.
264.
265.
266.
267.
268.
269.
270.
271.
272.
273.
274.
275.
276.
277.
278.
279.
280.
281.
282.
283.
284.
285.
286.
287.
288.
289.
290.
291.
292.
293.
294.
295.
296.
297.
298.
299.
300.
301.
302.
303.
304.
305.
306.
307.
308.
309.
310.
311.
312.
313.
314.
315.
316.
317.
318.
319.
320.
321.
322.
323.
324.
325.
326.
327.
328.
329.
330.
331.
332.
333.
334.
335.
336.
337.
338.
339.
340.
341.
342.
343.
344.
345.
346.
347.
348.
349.
350.
351.
352.
353.
354.
355.
356.
357.
358.
359.
360.
361.
362.
363.
364.
365.
366.
367.
368.
369.
370.
371.
372.
373.
374.
375.
376.
377.
378.
379.
380.
381.
382.
383.
384.
385.
386.
387.
388.
389.
390.
391.
392.
393.
394.
395.
396.
397.
398.
399.
400.
401.
402.
403.
404.
405.
406.
407.
408.
409.
410.
411.
412.
413.
414.
415.
416.
417.
418.
419.
420.
421.
422.
423.
424.
425.
426.
427.
428.
429.
430.
431.
432.
433.
434.
435.
436.
437.
438.
439.
440.
441.
442.
443.
444.
445.
446.
447.
448.
449.
450.
451.
452.
453.
454.
455.
456.
457.
458.
459.
460.
461.
462.
463.
464.
465.
466.
467.
468.
469.
470.
471.
472.
473.
474.
475.
476.
477.
478.
479.
480.
481.
482.
483.
484.
485.
486.
487.
488.
489.
490.
491.
492.
493.
494.
495.
496.
497.
498.
499.
500.
501.
502.
503.
504.
505.
506.
507.
508.
509.
510.
511.
512.
513.
514.
515.
516.
517.
518.
519.
520.
521.
522.
523.
524.
525.
526.
527.
528.
529.
530.
531.
532.
533.
534.
535.
536.
537.
538.
539.
540.
541.
542.
543.
544.
545.
546.
547.
548.
549.
550.
551.
552.
553.
554.
555.
556.
557.
558.
559.
560.
561.
562.
563.
564.
565.
566.
567.
568.
569.
570.
571.
572.
573.
574.
575.
576.
577.
578.
579.
580.
581.
582.
583.
584.
585.
586.
587.
588.
589.
590.
591.
592.
593.
594.
595.
596.
597.
598.
599.
600.
601.
602.
603.
604.
605.
606.
607.
608.
609.
610.
611.
612.
613.
614.
615.
616.
617.
618.
619.
620.
621.
622.
623.
624.
625.
626.
627.
628.
629.
630.
631.
632.
633.
634.
635.
636.
637.
638.
639.
640.
641.
642.
643.
644.
645.
646.
647.
648.
649.
650.
651.
652.
653.
654.
655.
656.
657.
658.
659.
660.
661.
662.
663.
664.
665.
666.
667.
668.
669.
670.
671.
672.
673.
674.
675.
676.
677.
678.
679.
680.
681.
682.
683.
684.
685.
686.
687.
688.
689.
690.
691.
692.
693.
694.
695.
696.
697.
698.
699.
700.
701.
702.
703.
704.
705.
706.
707.
708.
709.
710.
711.
712.
713.
714.
715.
716.
717.
718.
719.
720.
721.
722.
723.
724.
725.
726.
727.
728.
729.
730.
731.
732.
733.
734.
735.
736.
737.
738.
739.
740.
741.
742.
743.
744.
745.
746.
747.
748.
749.
750.
751.
752.
753.
754.
755.
756.
757.
758.
759.
760.
761.
762.
763.
764.
765.
766.
767.
768.
769.
770.
771.
772.
773.
774.
775.
776.
777.
778.
779.
780.
781.
782.
783.
784.
785.
786.
787.
788.
789.
790.
791.
792.
793.
794.
795.
796.
797.
798.
799.
800.
801.
802.
803.
804.
805.
806.
807.
808.
809.
810.
811.
812.
813.
814.
815.
816.
817.
818.
819.
820.
821.
822.
823.
824.
825.
826.
827.
828.
829.
830.
831.
832.
833.
834.
835.
836.
837.
838.
839.
840.
841.
842.
843.
844.
845.
846.
847.
848.
849.
850.
851.
852.
853.
854.
855.
856.
857.
858.
859.
860.
861.
862.
863.
864.
865.
866.
867.
868.
869.
870.
871.
872.
873.
874.
875.
876.
877.
878.
879.
880.
881.
882.
883.
884.
885.
886.
887.
888.
889.
890.
891.
892.
893.
894.
895.
896.
897.
898.
899.
900.
901.
902.
903.
904.
905.
906.
907.
908.
909.
910.
911.
912.
913.
914.
915.
916.
917.
918.
919.
920.
921.
922.
923.
924.
925.
926.
927.
928.
929.
930.
931.
932.
933.
934.
935.
936.
937.
938.
939.
940.
941.
942.
943.
944.
945.
946.
947.
948.
949.
950.
951.
952.
953.
954.
955.
956.
957.
958.
959.
960.
961.
962.
963.
964.
965.
966.
967.
968.
969.
970.
971.
972.
973.
974.
975.
976.
977.
978.
979.
980.
981.
982.
983.
984.
985.
986.
987.
988.
989.
990.
991.
992.
993.
994.
995.
996.
997.
998.
999.

```

```

240.         cvCalcOpticalFlowPyrLK( prev_grey, grey,
    prev_pyramid, pyramid, points[0][icnt], points[1][icnt], count,
241.             cvSize(win_size, win_size), 3, status, 0,
242.             cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 20, 0.03), flags
    );
243.         flags |= CV_LKFLOW_PYR_A_READY;
244.         for( i = k = 0; i < count; i++ )
245.         {
246.             if( add_remove_pt )
247.             {
248.                 double dx = pt.x - points[1][icnt][i].x;
249.                 double dy = pt.y - points[1][icnt][i].y;
250.                 if( dx*dx + dy*dy <= 25 )
251.                 {
252.                     add_remove_pt = 0;
253.                     continue;
254.                 }
255.             }
256.             if( !status[i] )
257.                 continue;
258.             points[1][icnt][k] = points[1][icnt][i];
259.             points[2][icnt][k++] = points[2][icnt][i];
260.         }
261.         count = k;
262.     }
263.     for( i = 0; i < count; i++ )
264.     {
265.         if( inj == 1 ) {
266.             points[2][icnt][i] = points[1][icnt][i];
267.         } else {
268.             cvCircle( imagep, cvPoint( (int)points[2][icnt][i].x, (int)points[2][
    icnt][i].y+sy ), 2, CV_RGB(255, 0, 0), -1, CV_AA, 0 );
269.             cvCircle( image, cvPoint( (int)points[1][icnt][i].x, (int)points[1][i
    cnt][i].y ), 3, CV_RGB(255, 0, 0), -1, CV_AA, 0 );
270.             cvShowImage( "Sum4", imagep );
271.             cvCircle( imagep2, cvPoint( (int)points[1][icnt][i].x, (int)points[1]
    [icnt][i].y+sy ), 2, CV_RGB(255, 0, 0), -1, CV_AA, 0 );
272.             cvShowImage( "Sum", imagep2 );
273.         }
274.     }
275.     if( add_remove_pt && count < MAX_COUNT )
276.     {
277.         points[1][icnt][count++] = cvPointTo32f(pt);
278.         cvFindCornerSubPix( grey, points[1][icnt] + count -
    1, 1, cvSize(win_size, win_size), cvSize(-1, -1),
279.             cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 20, 0.03));
280.         add_remove_pt = 0;
281.     }
282.     CV_SWAP( prev_grey, grey, swap_temp );
283.     CV_SWAP( prev_pyramid, pyramid, swap_temp );
284.     CV_SWAP( points[0][icnt], points[1][icnt], swap_points );
285.     need_to_init = 0;

```

```

286. }
287.
288. ///////////////////////////////////////////////////////////////////
289. ///////////////////////////////////////////////////////////////////
290. ///////////////////////////////////////////////////////////////////
291. ///////////////////////////////////////////////////////////////////
292. ///////////////////////////////////////////////////////////////////
293. ///////////////////////////////////////////////////////////////////
294. ///////////////////////////////////////////////////////////////////
295. void stereocallback(IplImage* Image,IplImage* Image2){
296.
297.     Im = cvCreateImage( cvGetSize(Image), 8, 3 );
298.     cvShowImage( "camview1", Image);
299.     cvShowImage( "camview2", Image2);
300.     if(!imagep){
301.
302.         imagep = cvCreateImage( cvSize(Image->width,Image-
>height),Image->depth,Image->nChannels );
303.         cvZero(imagep);
304.     }
305.     if(!imagep2){
306.         imagep2 = cvCreateImage( cvSize(Image->width,Image-
>height),Image->depth,Image->nChannels );
307.         cvZero(imagep2);
308.     }
309.     if(need_to_init == 2){
310.         cvCopy(Image,imagep);
311.         cvCopy(Image2,imagep2);
312.         int jk=0;
313.         for(jk=0;jk<8;jk++){
314.             need_to_init = 1;
315.             inj=1;
316.             sy = jk*Image2->height/8;
317.             cvSetImageROI (Image,cvRect(0,sy,imagep->width,Image2-
>height/8));
318.             icnt = jk;
319.             pointselector(Image);
320.             need_to_init = 0;
321.             inj=0;
322.             cvSetImageROI (Image2,cvRect(0,sy,imagep-
>width,Image2->height/8));
323.             pointselector(Image2);
324.         }
325.         cvFlip(imagep,imagep,-1);
326.         cvFlip(imagep2,imagep2,-1);
327.
328.     }
329.     cvSetImageROI (Image2,cvRect(0,0,Image->width,Image2-
>height));
330.     cvShowImage( "Sum2", imagep);
331.     cvShowImage( "Sum3", imagep2);
332. }
333. void oc::final(){
334.     cvDestroyAllWindows();
335.     cvcamStop( );
336.     cvcamExit( );
337. }

```

```

338. ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
339. ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
340. ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
341. //////////////////////////////////////////////////////////////////          CamSelect
342. //////////////////////////////////////////////////////////////////          //////////////////////////////////////////////////////////////////
343. ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
344. ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
345. void oc::camselect(){
346.     IplImage* img, * img2;
347.     char strFilter[] = { "jpg Files (*.jpg)|*.jpg|All Files
(*.*)|*.*||" };
348.     char rif[100],lif[100];
349.     CFileDialog FileDlg(FALSE, ".jpg", NULL, 0, strFilter);
350.
351.     if( FileDlg.DoModal() == IDOK )
352.     {
353.         sprintf(rif,"%s.svi",FileDlg.GetFileName());
354.         img = cvLoadImage(FileDlg.GetFileName());
355.         cvNamedWindow( "camview1", 1 );
356.         cvSetMouseCallback( "camview1", on_mouse, 0 );
357.         cvShowImage("camview1",img);
358.     }
359.     else
360.         return;
361.     if( FileDlg.DoModal() == IDOK )
362.     {
363.         sprintf(lif,"%s.svi",FileDlg.GetFileName());
364.         img2 = cvLoadImage(FileDlg.GetFileName());
365.         cvNamedWindow( "camview2", 1 );
366.         cvSetMouseCallback( "camview2", on_mouse2, 0 );
367.         cvShowImage("camview2",img2);
368.     }
369.     else
370.         return;
371.     if(!imagep){
372.         imagep = cvCreateImage( cvSize(img->width,img-
>height),img->depth,img->nChannels );
373.         cvZero(imagep);
374.         cvCopy(img,imagep);
375.     }
376.     if(!imagep2){
377.         imagep2 = cvCreateImage( cvSize(img->width,img-
>height),img->depth,img->nChannels );
378.         cvZero(imagep2);
379.         cvCopy(img2,imagep2);
380.     }
381.     cvShowImage("Sum",imagep);
382.     cvShowImage("Sum2",imagep2);
383.     while(1){
384.         c = cvWaitKey(10);
385.         if(c=='i'){
386.             need_to_init = 2;
387.             stereocallback(img,img2);
388.         }
389.         if(c=='s'){
390.             save_point(rif,lif);
391.         }

```

```

392.         if(c=='z'){
393.             if(omk>0 && omk2>0){
394.                 cvZero(imagep2);
395.                 cvCopy(img2,imagep2);
396.                 cvZero(imagep);
397.                 cvCopy(img,imagep);
398.                 undo();
399.             }
400.         }
401.         if(c=='c'){
402.             copy2img(rif,lif);
403.         }
404.         if(stopit == 1)
405.             break;
406.     }
407. }
408. oc::oc(void)
409. {
410. }
411.
412. oc::~~oc(void)
413. {
414. }

```

5) Graphic class header:

```

73. #include <stdio.h>
74. #include <math.h>
75. #include <time.h>
76. #include "gl/gl.h" // include core library interface
77. #include "gl/glu.h" // include GUI library interface
78. #include "gl/glut.h"
79.
80. #pragma once
81.
82. class og
83. {
84. private:
85.     int winid;
86.     int lop;
87. public:
88.     //void dc(int ogp[2][500][3],int cnt);
89.     og(void);
90.     void win(int ogp[2][500][3],int cnt);
91. public:
92.     virtual ~og(void);
93. };

```

6) Graphic class source:

```
1. #include "StdAfx.h"
2. #include "og.h"
3. int cnog,mod=0,initial=0;
4. double yRot=0,zRot=0,xRot=0,scale=1.0,xp=-1.0,yp=-1.0;
5. og ogt;
6. #define PI 3.1415
7. float xxRot = 0.0;
8. float x=0,y=3,z=0;
9. int artt[2][500][3];
10.  bool conb[500][500];
11.  const int mxmt = 40;
12.  int mxm = mxmt/4;
13.  int minloc[mxmt];
14.  double mint[mxmt];
15.  bool cand = false;
16.  void dc(int ogp[2][500][3],int cnt){
17.      int i,j,k;
18.      for(i=0;i<2;i++){
19.          for(j=0;j<cnt;j++){
20.              for(k=0;k<3;k++){
21.                  artt[i][j][k] = ogp[i][j][k];
22.              }
23.          }
24.      }
25.  void fc(int cnt){
26.      int i,j,k,l;
27.      double conn[1][500];
28.      for(i=1;i<cnt;i++){
29.          k=0;
30.          if(i!= 1)
31.              mint[0] = sqrt(pow((float)(artt[0][i][0]-
artt[0][1][0]),2)+pow((float)(artt[0][i][1]-
artt[0][1][1]),2)+pow((float)(artt[0][i][2]-artt[0][1][2]),2));
32.          else
33.              mint[0] = sqrt(pow((float)(artt[0][i][0]-
artt[0][2][0]),2)+pow((float)(artt[0][i][1]-
artt[0][2][1]),2)+pow((float)(artt[0][i][2]-artt[0][2][2]),2));
34.          for(j=2;j<cnt;j++){
35.              conn[0][j] = sqrt(pow((float)(artt[0][i][0]-
artt[0][j][0]),2)+pow((float)(artt[0][i][1]-
artt[0][j][1]),2)+pow((float)(artt[0][i][2]-artt[0][j][2]),2));
36.              if(conn[0][j] <= mint[k && i != j]){
37.
38.                  l=k;
39.                  while(conn[0][j] <= mint[l]){
40.                      mint[l+1] = mint[l];
41.                      l--;
42.                  }
43.                  mint[l+1] = conn[0][j];
44.                  if(k<(mxm-2))
45.                      k++;
46.              }else if(k<(mxm-1) && i != j){
47.                  mint[k+1]=conn[0][j];
48.                  if(k<(mxm-2))
```

```

49.                                     k++;
50.                                     }
51.                                 }
52.                                 for(j=2;j<cnt;j++){
53.                                     conb[i][j] = false;
54.                                     for(l=0;l<=k;l++){
55.                                         if(conn[0][j] == mint[l])
56.                                             conb[i][j] = true;
57.                                     }
58.                                 }
59.                             }
60.     }
61.
62. void pointd(double x,double y,double z)
63. {
64.     glBegin(GL_LINES);
65.     glVertex3f(1+x, 1+y, 1+z);
66.     glVertex3f(-1+x, -1+y, -1+z);
67.     glEnd();
68.     glBegin(GL_LINES);
69.     glVertex3f(-1+x, 1+y, 1+z);
70.     glVertex3f(1+x, -1+y, -1+z);
71.     glEnd();
72.     glBegin(GL_LINES);
73.     glVertex3f(1+x, y, z+1);
74.     glVertex3f(-1+x, y, -1+z);
75.     glEnd();
76. }
77. void dline(double x,double y,double z,double x1,double y1,double
78. z1){
79.     glBegin(GL_LINES);
80.     glVertex3f(x, y, z);
81.     glVertex3f(x1, y1, z1);
82.     glEnd();
83. }
84. float red(float af[], float bf[], float cf[])
85. {
86.     float v1[3],v2[3],l[3],n[3],normn,norml,ia,id,ndotl=0.0;
87.     int i;
88.     for(i= 0;i<3;i++)
89.     {
90.         v1[i] = cf[i]-bf[i];
91.         v2[i] = af[i]-bf[i];
92.     }
93.     n[0] = (v1[1]*v2[2])-(v1[2]*v2[1]);
94.     n[1] = (v1[2]*v2[0])-(v1[0]*v2[2]);
95.     n[2] = (v1[0]*v2[1])-(v1[1]*v2[0]);
96.     normn =
97.     pow((pow(n[0],2)+pow(n[1],2)+pow(n[2],2)),(float)0.5);
98.     l[0] = x-bf[0];
99.     l[1] = y-bf[1];
100.    l[2] = z-bf[2];
101.    norml =
102.    pow((pow(l[0],2)+pow(l[1],2)+pow(l[2],2)),(float)0.5);
103.    for(i=0;i<3;i++)
104.    {
105.        l[i] = l[i] / norml;

```

```

103.         n[i] = n[i] / normn;
104.         ndot1 += l[i]*n[i];
105.     }
106.     ia = 0.5;
107.     id = 0.5*ndot1;
108.     return (ia+id);
109.
110. }
111.
112. void dtri(float af[],float bf[],float cf[]){
113.     glColor3f(red(af,bf,cf),0,0);
114.     glBegin(GL_TRIANGLES);
115.         glVertex3f(af[0], af[1], af[2]);
116.         glVertex3f(bf[0], bf[1], bf[2]);
117.         glVertex3f(cf[0], cf[1], cf[2]);
118.     glEnd();
119. }
120. void dotcloud(void){
121.     int i;
122.     for(i=0;i<cnog;i++)
123.         pointd(artt[0][i][0],artt[0][i][1],artt[0][i][2]);
124. }
125. void wiremod(void){
126.     int i,j;
127.     for(i=1;i<cnog;i++)
128.         for(j=1;j<cnog;j++){
129.             if(conb[i][j])
130.
131.                 dline(artt[0][i][0],artt[0][i][1],artt[0][i][2],artt[0][j][0],art
t[0][j][1],artt[0][j][2]);
132.         }
133. void solid(void){
134.     int i,j,k,l,m,n,o,p;
135.     float af[3],bf[3],cf[3];
136.     for(i=1;i<cnog;i++){
137.         k = 0;
138.         for(j=1;j<cnog;j++){
139.             if(conb[i][j] && i!=j){
140.                 minloc[k++] = j;
141.             }
142.         }
143.         for(m=0;m<k;m++){
144.             for(l=0;l<k;l++){
145.                 if(conb[minloc[m]][minloc[l]] &&
minloc[l]!=minloc[m] && minloc[l]!=i){
146.                     af[0] = artt[0][i][0];
147.                     af[1] = artt[0][i][1];
148.                     af[2] = artt[0][i][2];
149.                     bf[0] =
150.                         artt[0][minloc[l]][0];
151.                     bf[1] =
152.                         artt[0][minloc[l]][1];
153.                     bf[2] =
154.                         artt[0][minloc[l]][2];
155.                     cf[0] =
156.                         artt[0][minloc[m]][0];

```



```

208.             default:
209.                 solid();
210.                 break;
211.         }
212.
213.
214. }
215. void lights()
216. {
217.     glPushMatrix();
218.     x=390;
219.     y=300*cos(xxRot)+320;
220.     z =150*sin(xxRot)+900;
221.     glTranslatef(x,y,z);
222.     glColor3f(1,1,1);
223.     glutSolidSphere(10,10,10);
224.     glPopMatrix();
225. }
226. void myDisplay(void)
227. {
228.     glEnable(GL_DEPTH_TEST);
229.     glEnable(GL_CULL_FACE);
230.     glCullFace(GL_BACK);
231.     // Save matrix state and do the rotation
232.     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
233.     glPushMatrix();
234.     glTranslatef(xp,yp, 0.0f);
235.     glScalef(scale,scale,scale);
236.     glScalef(2.0/500.0,2.0/400.0,2.0/5000.0);
237.     glRotatef(xRot, 1.0, 0.0, 0.0);
238.     glRotatef(yRot, 0.0, 1.0, 0.0);
239.     glRotatef(zRot, 0.0, 0.0, 1.0);
240.     glBegin(GL_LINES);
241.         glVertex3f(0, 120, 0);
242.         glVertex3f(320, 120, 0);
243.     glEnd();
244.     glBegin(GL_LINES);
245.         glVertex3f(160, 240, 0);
246.         glVertex3f(160, 0, 0);
247.     glEnd();
248.     lights();
249.     MB();
250.     glPopMatrix();
251.     glutSwapBuffers();
252. }
253.
254.
255. void processNormalKeys(unsigned char key, int x, int y) {
256.     if (key=='1')
257.         mod = 0;
258.     if (key=='2')
259.         mod = 1;
260.     if (key=='3')
261.         mod = 2;
262.     if (key=='u')
263.         xxRot-= 0.05f;
264.     if (key=='y')

```

```

265.         xxRot+= 0.05f;
266.     if (key=='i')
267.         if(mxm > 3){
268.             mxm --;
269.             fc(cnog);
270.         }
271.     if (key=='o')
272.         if(mxm < mxmt){
273.             mxm ++;
274.             fc(cnog);
275.         }
276.
277.     glutPostRedisplay();
278. }
279.
280. void SpecialKeys(int key, int x, int y)
281. {
282.     if(key == GLUT_KEY_F1)
283.         yRot -= 0.5f;
284.     if(key == GLUT_KEY_F2)
285.         yRot += 0.5f;
286.     if(key == GLUT_KEY_F3)
287.         zRot -= 0.5f;
288.     if(key == GLUT_KEY_F4)
289.         zRot += 0.5f;
290.     if(key == GLUT_KEY_F5)
291.         xRot -= .5f;
292.     if(key == GLUT_KEY_F6)
293.         xRot += .5f;
294.     if(key == GLUT_KEY_F7)
295.         scale += .05f;
296.     if(key == GLUT_KEY_F8)
297.         scale -= .05f;
298.     if(key == GLUT_KEY_LEFT)
299.         xp -= .05f;
300.     if(key == GLUT_KEY_RIGHT)
301.         xp += .05f;
302.     if(key == GLUT_KEY_UP)
303.         yp += .05f;
304.     if(key == GLUT_KEY_DOWN)
305.         yp -= .05f;
306.     glutPostRedisplay();
307.
308. }
309. void og::win(int ogp[2][500][3],int cnt){
310.     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
311.     glutInitWindowSize(320,240);
312.     winid = glutCreateWindow("Sayed Ali Kasaei zadeh (6148)");
313.     glutKeyboardFunc(processNormalKeys);
314.     glutSpecialFunc(SpecialKeys);
315.     dc(ogp,cnt);
316.     cnog = cnt;
317.     glutDisplayFunc(myDisplay);
318.     glClearColor(0.3,0.3,0.7,0.0);
319.     if(lop==0)
320.         glutMainLoop();
321. }

```

```
322. og::og(void)
323. {
324.     lop=0;
325. }
326.
327. og::~~og(void)
328. {
329.     lop=1;}
```