

“3D VISUALIZATION USING DATA RECEIVED FROM
THE PROCESSES OF OBJECT RECOGNITION
AND OBJECT RECONSTRUCTION”

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ÇANKAYA UNIVERSITY

BY

SEHER PELİN GÜVENÇ

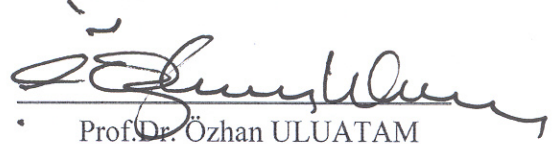
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JUNE, 2008

Title of the Thesis : **3D Visualization Using Data Received from the Processes of Object Recognition and Object Reconstruction.**

Submitted by **Seher Pelin GÜVENÇ**

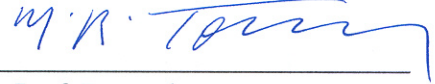
Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.



Prof. Dr. Özhan ULUATAM

Acting Director

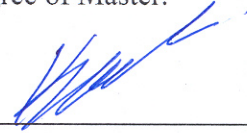
I certify that this thesis satisfies all the requirements as a thesis for the degree of Master.



Prof. Dr. Mehmet R. TOLUN

Head Of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master.



Asst. Prof. Dr. Abdül Kadir GÖRÜR


Supervisor

Examination Date : 20.06.2008

Examining Committee Members :

Asst. Prof. Dr. Reza HASSANPOUR

(Çankaya Univ.)



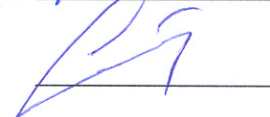
Asst. Prof. Dr. Abdül Kadir GÖRÜR

(Çankaya Univ.)



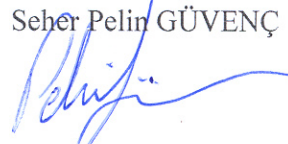
Dr. Ersin ELBAŞI

(TÜBİTAK.)



STATEMENT OF NON-PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Seher Pelin GÜVENÇ
Signature : 
Date : 16.05.2008

ABSTRACT

“3D VISUALIZATION USING DATA RECEIVED FROM THE PROCESSES OF OBJECT RECOGNITION AND OBJECT RECONSTRUCTION”

Güvenç, Seher Pelin

M.S.c., Department of Computer Engineering

Supervisor: Asst. Prof. Dr. Abdül Kadir GÖRÜR

June 2008, 61 pages

This thesis presents the demonstration of what two images or two video sequences can tell us about the situation and model of a third video sequence or image. The method bears ideas from projective geometry as it's basis.

The main purpose of the thesis is to be able to form a base line for tracking an object in a 3D environment not only by using two stereo cameras but also by using other cameras that may be located in various points of the environment. The method visualizes the object and gives the information to a third camera. This way it can be possible to track a moving object, along with it's visualized model, in an environment without losing sight of it and without having to move the other two stereo cameras which we received data from.

Keywords: 3D Visualization, Projective Geometry, Pinhole Camera Model, Hidden Surface Removal.

ÖZ

“CİSİM TANIMLAMA VE CİSİM REKONSTRÜKSİYONUNDAN ALINAN BİLGİ İLE CİSMİ 3-BOYUTLU OLARAK GÖRÜNTÜLEME”

Güvenç, Seher Pelin

Yükseklisans, Bilgisayar Mühendisliği Anabilim Dalı

Tez Yöneticisi: Yrd. Doç. Dr. Abdül Kadir GÖRÜR

Haziran 2008, 61 sayfa

Bu tez çalışması, iki çiftli kameranın bize üçüncü bir kameradaki durum hakkında neler söyleyebileceği ile ilgilidir. Bu görüntüler video veya resim şeklinde de olabilir Uygulanan metod izdüşümsel geometrinin fikirlerinden yola çıkarak gerçekleştirilmiştir.

Tezdeki asıl amaç 3 boyutlu bir ortamda takip edilen cisimi kaybetmemeye çalışmaktır. İki tane sabit çiftli kameradan aldığımız cisim rekonstrüksiyon ve cisimi tanımlama bilgileri ile üçüncü bir kamerada o cisimin nerede olabileceği ile ilgili, görsel bir sembol ile bilgi vermiş oluyoruz. Böylece üç boyutlu ve birden fazla sabit kameranın bulunduğu bir ortamda cisimi kaybetmeden takip edebiliriz.

Anahtar Kelimeler: 3-Boyutlu Görüntüleme, İzdüşümsel Geometri, İğne Deliği Kamera Modeli, Gizli Yüzey Giderme.

ACKNOWLEDGMENTS

I want to thank my co-supervisor Asst. Prof. Dr. Reza HASSANPOUR for sharing his experiences and great knowledge and for his guidance throughout the study of this thesis. I would also like to thank my supervisor Asst. Prof. Dr. Abdülkadir GÖRÜR for his great support and understanding.

I also want to thank my parents; my mom for her support and patience and my dad for his suggestions and comments about the methods of pursuing this experiment.

TABLE OF CONTENTS

STATEMENT OF NON PLAGIARISM.....	iii
ABSTRACT.....	iv
ÖZ.....	v
ACKNOWLEDGMENTS.....	vi
TABLE OF CONTENTS.....	vii
LIST OF FIGURES.....	ix
LIST OF SYMBOLS.....	xi
LIST OF ABBREVIATIONS.....	xii
CHAPTERS:	
1. INTRODUCTION.....	1
2. THE GEOMETRIC BACKGROUND.....	4
2.1 Transformations.....	4
2.1.1 Rotation.....	5
2.1.2 Translation.....	7
3. IMAGE FORMATION.....	10
3.1 Perspective Projection.....	10
3.1.1 The Equations of Perspective Projection.....	13
3.1.2 Properties of Perspective Projection.....	14

3.2	The Pinhole Camera Model.....	17
4.	VISUALIZATION.....	23
4.1	Information Visualization.....	24
5.	HIDDEN SURFACE DETERMINATION.....	27
5.1	Warnock's Algorithm.....	29
5.2	The Painter's Algorithm.....	30
5.3	Binary Space Partitioning (BSP).....	31
5.4	Ray Tracing.....	32
5.5	The Z-Buffer Algorithm (The Depth Buffer Algorithm).....	33
6.	SUPERIMPOSITION.....	40
7.	METHODS AND ALGORITHMS.....	42
7.1	Steps Followed During the Application Process.....	42
7.1.1	Receiving Image Data from Two Stereo Camera's.....	43
7.1.2	Using the Geometric Transformation Methods.....	43
7.1.3	Performing Perspective Projection.....	47
7.1.4	3D Visualization (3D Data Visualization).....	50
7.1.5	Performing the Z-Buffer (Depth Buffer) Algorithm.....	52
7.1.6	Simple Superimposition.....	55
7.2	Experimental Results.....	57
8.	CONCLUSION AND FUTURE WORK.....	59
	REFERENCES.....	R1

LIST OF FIGURES

FIGURES

Figure 2.1	Rotation Around the X-Axis.....	6
Figure 2.2	Rotation Around the Y-Axis.....	7
Figure 2.3	Rotation Around the Z-Axis.....	7
Figure 2.4	2D Translation.....	8
Figure 2.5	3D Translation.....	9
Figure 3.1	Standard Perspective Projection.....	11
Figure 3.2	Alignment of the Camera with respect to the World Coordinate System	12
Figure 3.3	Assumed Alignment of the Image Plane with respect to O	12
Figure 3.4	Geometric Model of Perspective Projection.....	13
Figure 3.5	Model of Object Distance and Size Relationship.....	15
Figure 3.6	Difference between <i>Scaling</i> and <i>Foreshortening</i>	15
Figure 3.7	Example of a Vanishing Point in a Real Life Image.....	16
Figure 3.8	Geometric Representation of Vanishing Point.....	17
Figure 3.9	The Pinhole Camera Model.....	18
Figure 4.1	Visualization of the Human Face using Point Clouds.....	25
Figure 4.2	Visualization of an Architectural Model.....	25
Figure 5.1	Drawing Done Without Performing Hidden Surface Removal.....	27

Figure 5.2	Drawing Done by Performing Hidden Surface Removal.....	27
Figure 5.3	Hidden Line Removal Performed.....	28
Figure 5.4	Hidden Line Removal Not Performed.....	28
Figure 5.5	The Application of Warnock’s Algorithm.....	29
Figure 5.6	The Mountains are Painted First, and then the Meadows.....	31
Figure 5.7	The Few Simple Steps of Binary Space Partitioning.....	32
Figure 5.8	The Ray Tracing Algorithm.....	33
Figure 5.9	Conceptual View of the Depth Buffer.....	34
Figure 5.10	The Z-Buffer Algorithm.....	35
Figure 5.11	Computation of Pseudodepth.....	36
Figure 6.1	Superimposition of a 3D Teapot Model.....	40
Figure 7.1	The Point of View used in the Previous Experiment.....	44
Figure 7.2	The Point of View used in This Experiment.....	44
Figure 7.3	Erroneous Projection.....	48
Figure 7.4	Projection of the 3D Edge Point Coordinates.....	49
Figure 7.5	Projection of the 3D Corner Point Coordinates.....	50
Figure 7.6	3D Visualization of 3D Edge Point Coordinates.....	51
Figure 7.7	Barn Mesh File.....	54
Figure 7.8	Visualization of the Barn Example.....	55
Figure 7.9	Point of View of First Camera.....	56
Figure 7.10	Point of View of Second Camera.....	56
Figure 7.11	Point of View of the Third Camera.....	57

LIST OF SYMBOLS

SYMBOLS

$T(\vec{x})$: Transformation Function

A : Transformation Matrix of T

\vec{x} : Column Vector

R : Rotation Matrix

R^{-1} : Inverse of R

R^T : Transpose of R

T : Translation Matrix

Π : Image Plane

f : Focal Length

P : Point in world coordinate system

p : Point projected to image plane

O : Center of Projection

d : Depth Buffer

p : Pixel Buffer

LIST OF ABBREVIATIONS

ABBREVIATIONS

- 2D : Two Dimensional
- 3D : Three Dimensional
- BSP : Binary Space Partitioning
- lerp* : Linear Interpolation
- AR : Augmented Reality

CHAPTER 1

INTRODUCTION

The purpose of this thesis is to create a program that will visualize a 3D wire frame model of an object that will be tracked in an object tracking system. This visualization will give us an idea of the location of the object being tracked since with a simple use of superimposition we will mark the location of the object while being tracked. Though since this process takes time due to various calculations that are pursued, it is not done in real time.

Various methods like geometric transformations, perspective projection, visualization and superimposition were used in the making of this thesis. The methods are elucidated in the following chapters of this thesis.

In Chapter 2 we describe the basic geometric methods that were used in the basis of the thesis. These geometric methods (rotation and translation) were used to convert the 3D corner and edge point coordinates received from the second or first camera, to the world coordinates and then to the coordinates of the third camera or third view point. This way we obtained the 3D point coordinates as numeric data and stored them into a file to use them in the next process.

In Chapter 3 we describe the basics of image formation by describing the Pinhole Camera Model and perspective projection. Perspective projection plays a big role in mapping 3D image data's to 2D image planes. It was one of the main processes used in the implementation part of the experiment. The mapping of the 3D coordinates, that were transformed using the geometric transforms, rotation and translation, were used in the geometric background of the projective geometry.

The Pinhole Camera model shows us how an image in 3D is mapped onto the a 2D image plane. Here we also define the terms *intrinsic parameters* and *extrinsic parameters* of a camera.

In the next chapter, Chapter 4, the term *visualization* is defined. It explains the types of visualization; though emphasizing on data visualization. It shows the types of data that may be used when visualizing an object. And gives examples of the tools used in graphics visualization. The data in which we used in this experiment was the 3D edge and corner point coordinates which were stored in a file, or in other words numeric data. Another file which included the surface 3D point coordinates, which was used in the process of applying the Z-Buffer algorithm, was used as well.

Chapter 5 defines the meaning of removing hidden surfaces and explains the various hidden surface algorithms used to eliminate surfaces that remain behind other surfaces. The algorithm in which we emphasize on is the Z-Buffer (depth buffer) algorithm, because this algorithm was used in the method for performing the hidden surface eliminations.

The reason of removing such surfaces gives us a more realistic view of the object modeled.

The last method used was superimposition, seen in Chapter 6. This section of the thesis gives a brief explanation of the term *superimposing* and explains its applications. It also mentions very briefly about an application in which superimposition is used, called *augmented reality*.

In the end we combine the methods and explain the process done and how the methods were used during the study period of the thesis, along with what could be done in the future with a more advanced version of the methods and system.

CHAPTER 2

THE GEOMETRIC BACKGROUND

In the basis of the experiment, the geometrical methods used are the mathematical and geometrical algorithms and equations used in both the camera model, image processing and the three dimensional graphics operations.

The most commonly used geometric method are the transformations.

2.1 Transformations

There are several transformations in linear algebra; linear transformation, affine transformation and perspective projection. In our case we have used the linear transformation.

A linear transformation can be represented by a matrix. If we have a linear transformation T which maps a column vector \vec{x} that has n entries to a vector with m columns, then we have;

$$T(\vec{x}) = \mathbf{A}\vec{x} \tag{2.1}$$

Where the $m \times n$ matrix \mathbf{A} is called the **transformation matrix of T** .

The following are the most common used linear transformations and are also the ones that were used in the baseline of the experiment.

2.1.1 Rotation

Rotation is a transformation around a plane or space that describes the motion of a solid object around a fixed point. At the end the distance between any two points on the object are unchanged.

R is said to be a rotation matrix if $R^{-1} = R^T$, which means that each rotation matrix is orthogonal.

There are two types of rotation; axis rotation and vector rotation. In our case we used the axis rotation since we had two camera's positioned in different places which means their coordinate axis's were rotated according to one another.

The representation of a 3x3 rotation matrix can be given as follows;

$$\mathbf{R} = \mathbf{R}_\zeta \mathbf{R}_\eta \mathbf{R}_\xi \quad (2.2)$$

Where in matrix notation;

$$\mathbf{R}_\xi = \begin{pmatrix} \cos(\xi) & \sin(\xi) & 0 \\ -\sin(\xi) & \cos(\xi) & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad (2.3)$$

(Rotation done around the z-axis)

$$\mathbf{R}_\eta = \begin{pmatrix} \cos(\eta) & 0 & -\sin(\eta) \\ 0 & 1 & 0 \\ \sin(\eta) & 0 & \cos(\eta) \end{pmatrix} \text{ and} \quad (2.4)$$

(Rotation done around the new y-axis)

$$\mathbf{R}_\zeta = \begin{pmatrix} \cos(\zeta) & \sin(\zeta) & 0 \\ -\sin(\zeta) & \cos(\zeta) & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (2.5)$$

(Rotation done around the new z-axis (x-axis))

When we multiply each individual matrix we get the rotation matrix R ;

$$\mathbf{R} = \begin{pmatrix} \cos(\zeta)\cos(\eta)\cos(\xi) - \sin(\zeta)\sin(\xi) & \cos(\zeta)\cos(\eta)\sin(\xi) + \sin(\zeta)\cos(\xi) & -\cos(\zeta)\sin(\eta) \\ -\sin(\zeta)\cos(\eta)\cos(\xi) - \cos(\zeta)\sin(\xi) & -\sin(\zeta)\cos(\eta)\sin(\xi) + \cos(\zeta)\cos(\xi) & \sin(\zeta)\sin(\eta) \\ \sin(\eta)\cos(\xi) & \sin(\eta)\sin(\xi) & \cos(\eta) \end{pmatrix} \quad (2.6)$$

Figure 2.1 shows the rotation around the x -axis, Figure 2.2 shows the rotation around the y -axis and Figure 2.3 shows the rotation around the z -axis.

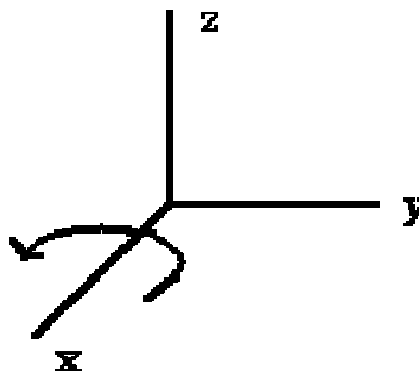


Figure 2.1 : Rotation Around the X-Axis

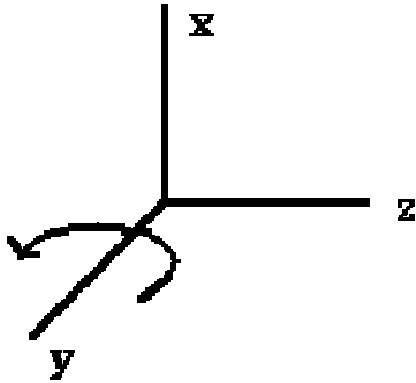


Figure 2.2 : Rotation Around the Y-Axis

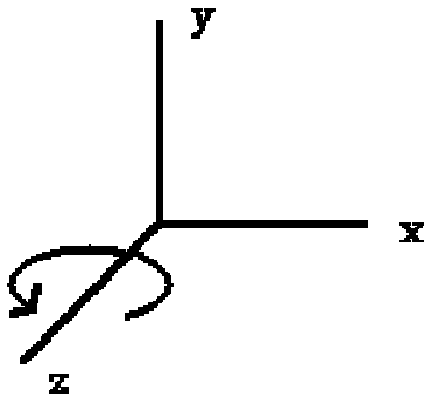


Figure 2.3 : Rotation Around the Z-Axis

2.1.2 Translation

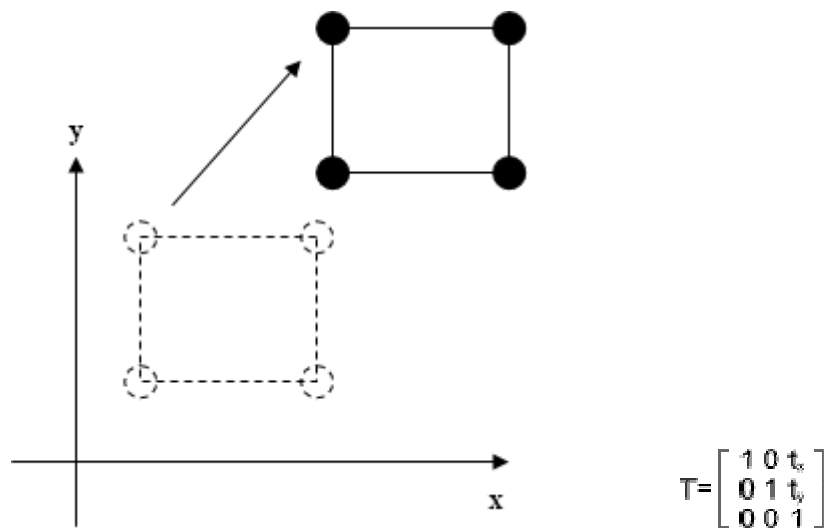
In image processing translation is the movement of every point on the object in a specified direction. Though the points all move a constant distance, meaning that each point on the object moves the same amount towards the specified direction.

The translation matrix, T is a vector that takes 2 or 3 parameters depending on the dimension that the translation will be taking place. In our case we have used three parameters in our translation vector since translation was done in 3D. It is represented as follows;

$$T = (t_x, t_y, t_z) \quad (2.7)$$

Where t_x represents the translation done in the x -axis, t_y represents the translation done in the y -axis and t_z represents the translation done in the z -axis.

Figure 2.4 shows translation done in 2D and Figure 2.5 shows the translation done in 3D.



$$x' = x + t_x$$

$$y' = y + t_y$$

Figure 2.4 : 2D Translation

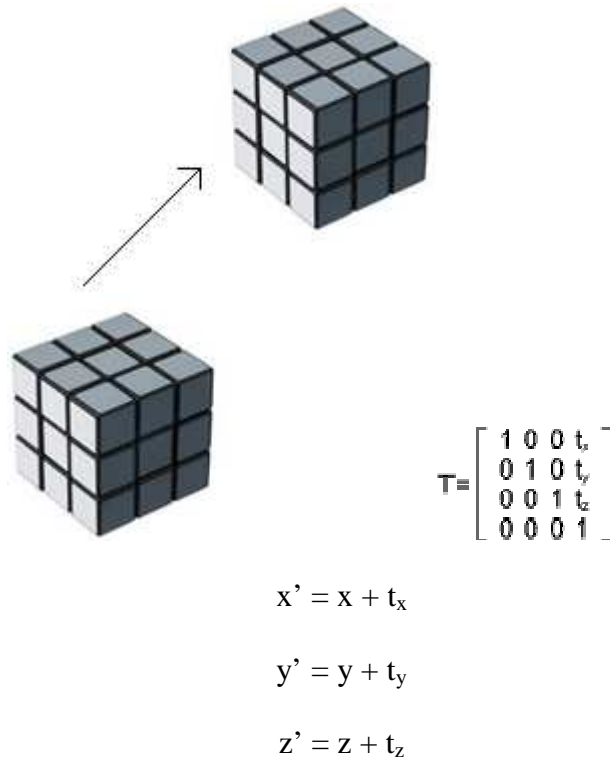


Figure 2.5 : 3D Translation

There are other types of transformations as well such as *Scaling* in which the object either enlarges or diminishes in size. This may happen in 2D or 3D. Another transformation method is the *Normal Transform*. Just as we use the matrix T to transform the geometry of the points or lines we use the normal transform method to transform the normals of these geometries.

CHAPTER 3

IMAGE FORMATION

The commonly used model for capturing images from 3D to 2D is the Pinhole Camera Model. Originally the Pinhole Camera is a camera without a glass lens. It consists of an extremely small hole and is made up of very thin material. This hole can focus the light by confining all rays from a scene through a single point.

In order to understand the Pinhole Camera Model we first need to understand the term; *perspective projection*.

3.1 Perspective Projection

Perspective projection is used to project the 3D world to the 2D image plane. A basic rule of perspective projection is that something that is further away from the viewer at a three-dimensional space is “smaller” in the two-dimensional representation and “larger” if it is closer in the two-dimensional space [1].

Figure 3.1 illustrates how perspective projection works.

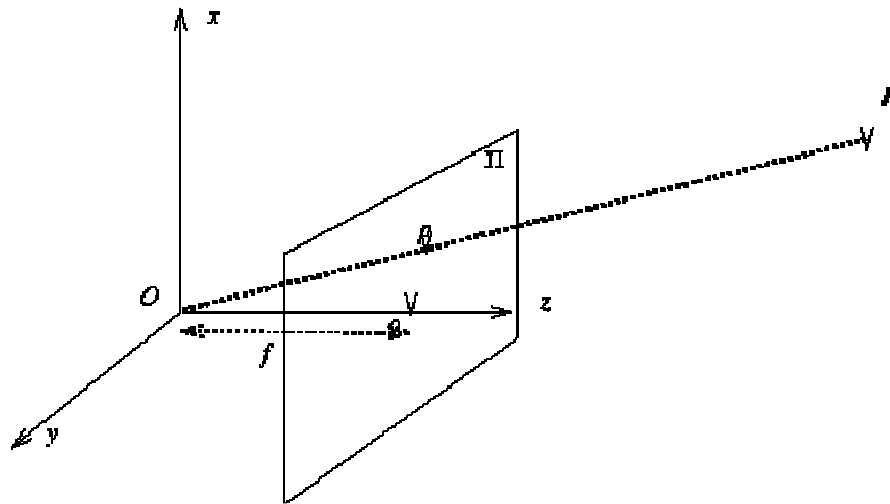


Figure 3.1 : Standard Perspective Projection.[2]

As it is seen the *center of projection* is at the center O which is located at the center of the 3D reference frame, or in other words the *world* coordinate axis. The image plane Π is parallel to the (x, y) plane and is shifted a distance the size of the focal length f along the z -axis from the origin of the 3D reference frame. The 3D point P projects to the image point p .

Though in order to perform perspective projection some simple geometric arrangements need to be done. We know that in general the world coordinate system does not align or overlap with the camera coordinate system.

Figure 3.2 shows a simple model of how the camera may be aligned according to the world coordinate system. As it is seen in the figure the z -axis of the world coordinate system is upward when the camera's z -axis faces a different direction. The camera coordinate system is rotated and translated compare to the world coordinate system.

This can make things more difficult when calculating and projecting points on the image plane.

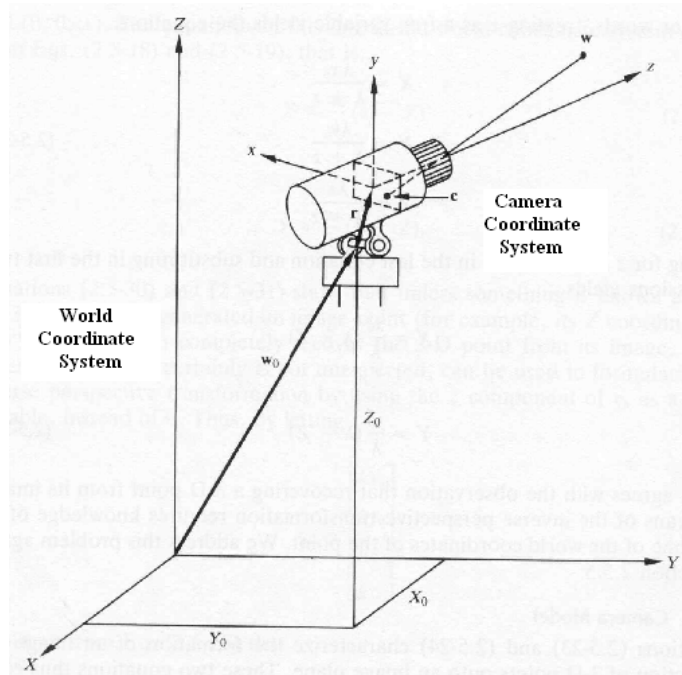


Figure 3.2 : Alignment of the Camera with respect to the World Coordinate System.[2]

For this reason, in order to simplify the derivation of the perspective projection equations, the following assumptions should be made;

1. The center of projection O , described above, overlaps with the origin of the world coordinate system.
2. The camera's axis (optical axis) is aligned with the world's z coordinate axis.
3. And we should avoid image inversion by assuming that the location of the image plane is in front of the center of projection as shown in Figure 3.3.

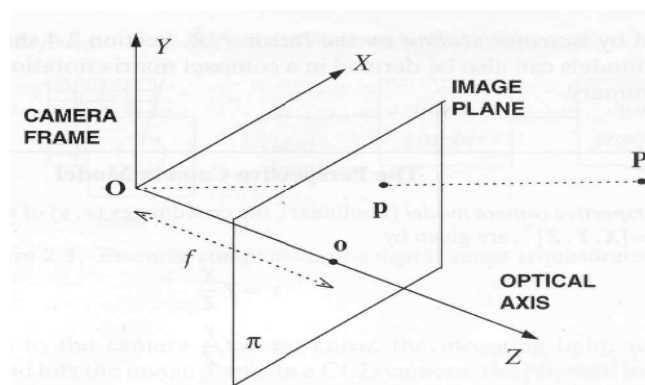


Figure 3.3 : Assumed Alignment of the Image Plane with respect to O . [2]

The line passing through O , and that is perpendicular to the image plane is called the *optical axis*, as seen in the figure above. The intersection of the *optical axis* with the image plane is called the *principal point* or *image center*. (Though the principal point does not always have to be the image center.)

3.1.1 The Equations of Perspective Projection

Figure 3.4 shows the geometric model of perspective projection, according to the figure the following equations are derived.

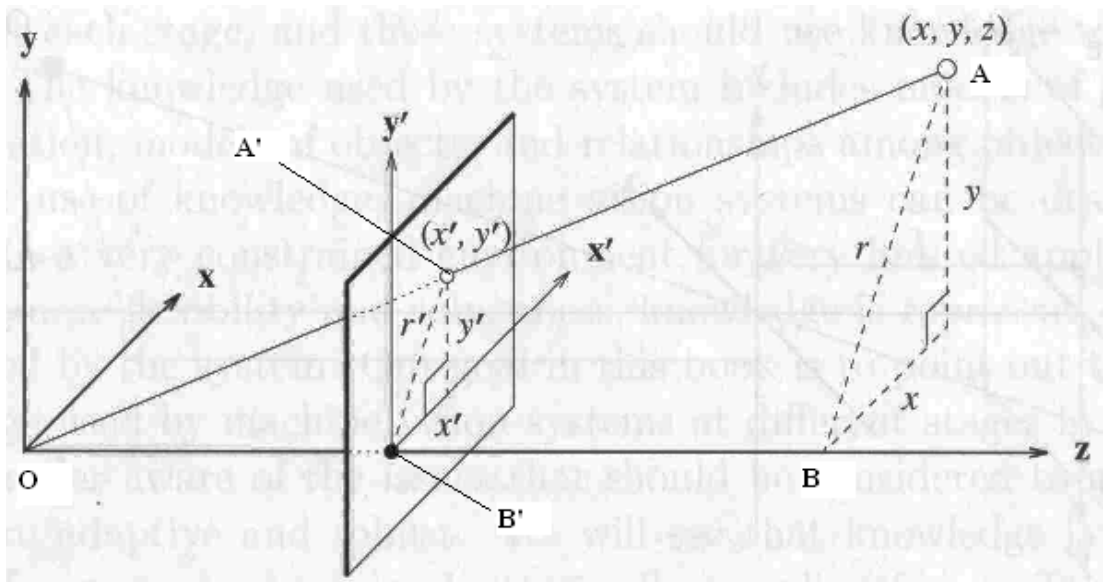


Figure 3.4 : Geometric Model of Perspective Projection. [2]

(notations: $(x, y, z) \rightarrow (X, Y, Z)$, $r \rightarrow R$, $(x', y', z') \rightarrow (x, y, z)$, $r' \rightarrow r$)

By using the simple triangles indicated in the figure above, and with the help of simple geometry and ratio we have;

- From $OA'B'$ and OAB : $f / Z = r / R$ (f is the focal length.)
- From $A'B'C'$ and ABC : $x / X = y / Y = r / R$

From the above equations we obtain the following;

$$x = \frac{Xf}{Z} \quad y = \frac{Yf}{Z} \quad z = f \quad (3.1)$$

By using the matrix notation the perspective projection matrix looks like the following;

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (3.2)$$

To verify whether the matrix above is correct or not we do the following (homogenize the matrix by assuming $w = Z$);

$$x = \frac{x_h}{w} = \frac{fX}{Z} \quad y = \frac{y_h}{w} = \frac{fY}{Z} \quad z = \frac{z_h}{w} = f \quad (3.3)$$

3.1.2 Properties of Perspective Projection [2]

- **Many-to-One Mapping;** The projection of a point is not unique. Any point on the line connecting the *center of projection*, O , and the point P as shown in figure 3.3, has the same projection.
- **Scaling/Foreshortening;** The distance of an object is inversely proportional to its image size. In other words, the farther away the object is the smaller it would look on the image plane. Why

it is like that is explained in Figure 3.5. It shows the geometric basics.

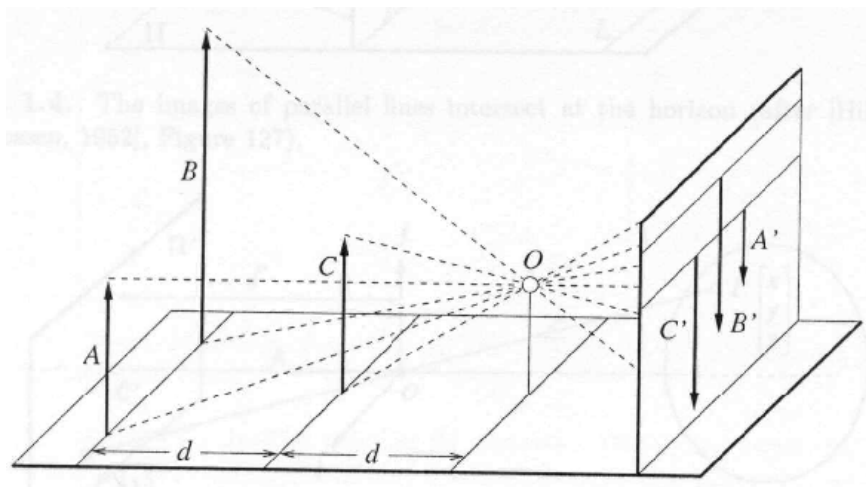


Figure 3.5 : Model of Object Distance and Size Relationship. [2]

When a line (or surface) is parallel to the image plane the effect of perspective projection is *scaling*. In the reverse situation, meaning that if the line or surface is not parallel to the image plane the term *foreshortening* is used to describe the projective distortion. As shown in figure 3.6 the line parallel to the optical axis is compressed relative to the frontal line.

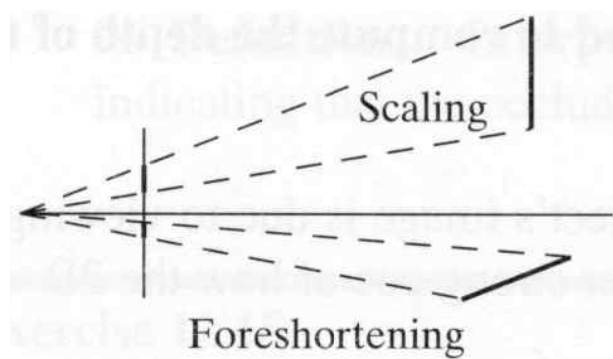


Figure 3.6 : Difference between *Scaling* and *Foreshortening*. [2]

- Effect of the Focal Length; As the focal length f gets smaller the number of points projected on the image plane increase.(example: wide-angle camera). On the contrary, if f gets larger the field of view gets smaller (example: telescopic view.)
- Lines, Distances, Angles; Lines in 3D project to lines in 2D. While doing that the distances and angles are *not* preserved. And parallel lines do not project to parallel lines unless they are parallel to the image plane.
- Vanishing Point; Parallel lines in space project perspectively onto lines that intersect at a single point in the image plane called the *vanishing point* or *point at infinity*. The vanishing point of any given line in space is located at the point in the image where a parallel line through the center of projection intersects the image plane. Figure 3.7 shows an image of a vanishing point.



Figure 3.7 : Example of a Vanishing Point in a Real Life Image. [2]

The following figure shows the geometric representation of the perspective projection of parallel lines in the world coordinate system. See how the parallel lines tend to vanish when projected onto the image plane.

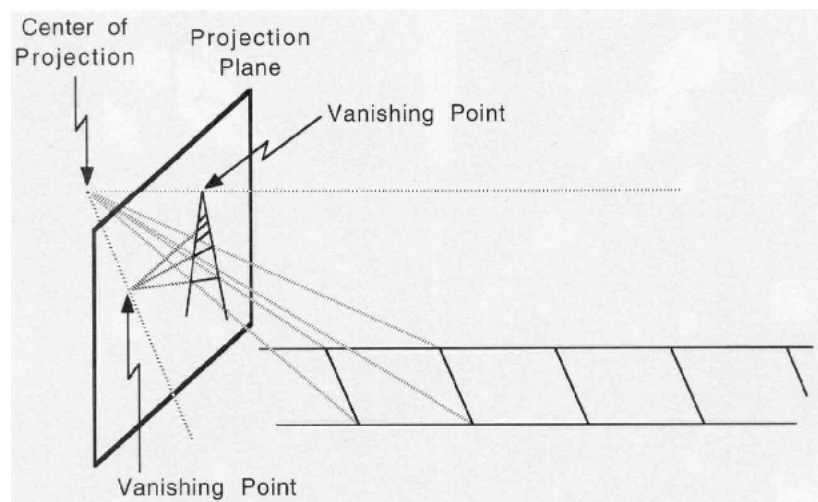


Figure 3.8 : Geometric Representation of Vanishing Point. [2]

- **Vanishing Line;** The vanishing points of all the lines that lie on the same plane form the *vanishing line*.

3.2 The Pinhole Camera Model

As it is stated at the beginning of this chapter; in a Pinhole Camera model a scene view is formed by projecting 3D points into the image plane by using perspective projection.

Figure 3.9 shows a simple drawing of how the Pinhole Camera model looks like. The rays of light which reflect from the top and bottom of the object go through

the tiny hole to form an image in the back of the box, though an upside down image of the object is created. It is similar to the image formation in the human eye.

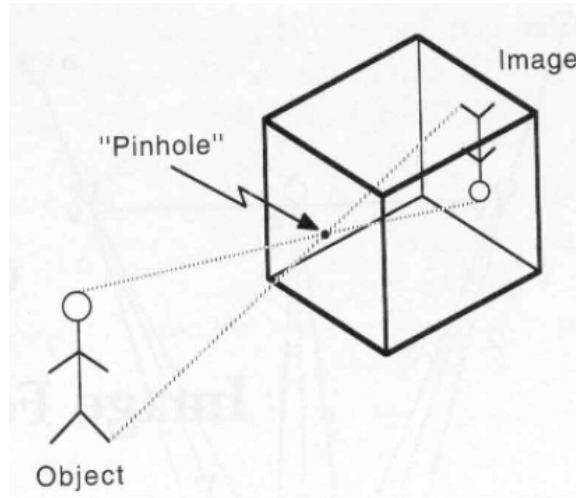


Figure 3.9 : The Pinhole Camera Model. [2]

The geometry of the Pinhole Camera Model is shown in Figure 3.1 where we described the geometry of perspective projection. Since the basis of the Pinhole Camera relies on perspective projection there is no other way then to represent it geometrically.

The projection of a 3D point on an object to the 2D image plane can be represented by the following equation;

$$s * p' = A * [R | t] * P' \quad (3.4)$$

or

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3.5)$$

Here the (x, y, z) coordinates are the coordinates of a point in the 3D world coordinate space. The “1” in the coordinate vector states that the coordinates are homogeneous coordinates. (u, v) represent the coordinates of the point projection. The camera matrix, or matrix of intrinsic parameters is represented by A . It includes the parameters (c_x, c_y) , which is the principal point that is usually considered as the image center and f_x, f_y which are the focal lengths in both the x and the y direction respectively.

If the image maybe scaled by some factor the parameters $(f_x, f_y, c_x$ and $c_y)$ must all be scaled (multiplied or divided depending on the sampling done) by the same factor as well.[3]

The matrix of intrinsic parameters does not depend on the scene viewed, and once it is estimated it can be used again as long as the focal length does not change. Otherwise it may differ in case of zoom lens, which in that case will have to be estimated again with the new focal length values.[3]

The joint rotation-translation matrix $[R | t]$ is called the matrix of extrinsic parameters and is used to describe the motion of a camera around a static scene or a motion of an object in front of a still camera. The joint rotation-translation matrix; $[R | t]$ translates coordinates of a point (x, y, z) to some other coordinate system, fixed with respect to the camera. [3]

The equation above is a detailed version of how the actual estimation is done when converting coordinates from one coordinate system to another coordinate

system or in other words another camera's coordinate system, the following equation is another representation of the equation above when $z \neq 0$;

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \quad (3.6)$$

$$x' = x / z$$

$$y' = y / z$$

$$u = f_x * x' + c_x$$

$$v = f_y * y' + c_y$$

Here R represents the rotation matrix belonging to the camera in which the coordinates (x, y, z) want to be estimated. t is the translation vector of the same camera. The projected points u and v are calculated using the transformed coordinates (x', y') and the intrinsic parameters of the camera matrix.

Though in the real case lens usually have distortion, which means the abnormal rendering of lines in an image. There are two major distortion components which are radial and slightly tangential distortion. With these two components the above equation turns into the following equation;

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$\begin{aligned}x' &= x/z \\y' &= y/z\end{aligned}\tag{3.7}$$

$$\begin{aligned}x'' &= x' * (1 + k_1 r^2 + k_2 r^4) + 2 * p_1 x' * y' + p_2 (r^2 + 2 * x'^2) \\y'' &= y' * (1 + k_1 r^2 + k_2 r^4) + p_1 (r^2 + 2 * y'^2) + 2 * p_2 * x' * y'\end{aligned}$$

Where $r^2 = x'^2 + y'^2$.

And from this the (u, v) parameters become;

$$\begin{aligned}u &= f_x * x'' + c_x \\v &= f_y * y'' + c_y\end{aligned}\tag{3.8}$$

The parameters k_1 and k_2 are the radial distortion coefficients. The parameters p_1 and p_2 are the tangential distortion coefficients. The distortion coefficients also do not depend on the scene viewed since they are also considered as the intrinsic parameters of the camera. And they do not differ according to the resolution of the image taken.

The following are the methods in which the Pinhole camera model described above is used;

- Projecting 3D points to an image plane given the intrinsic and extrinsic parameters of the camera.
- Computing extrinsic parameters given intrinsic parameters of the camera and a few 3D points and there projections.

- Estimating intrinsic and extrinsic camera parameters from several views of a known calibration pattern (i.e. every view is described by several 3D-2D point correspondences). [3]

CHAPTER 4

VISUALIZATION

Visualization can be any technique used for creating images, diagrams and animations to show what the data has to tell us. It has been used since the first times man learned how to draw.

We can visualize 2D or 3D data depending on the numeric values that are present. Because, visualization transforms numeric data into a visual form that enables the users to conceptualize and understand the information. 3D visualization is the ability to display, analyze, manipulate and interact with 3D data in a 3D environment [4].

There are various fields, in computer graphics, in which visualization is used. These fields are; Information Visualization, Knowledge Visualization, Educational Visualization and Product Visualization. We are going to emphasize on Information Visualization since it is the type of visualization method used in the experimental part of this thesis. The information received for applying visualization is numeric data.

4.1 Information Visualization

Information visualization concentrates on the use of computer-supported tools and libraries to examine large amounts of abstract data. The use of visualizing such data helps the user or developer gain knowledge about the information written in a file or another type of structure and helps the user or developer analyze the abstract data in more detail. These structures maybe coordinates of points in 3D space, or point cloud data of an object (which is our case) or could be color data or texture data etc.

There are various visualization techniques which are commonly used; some of them are; constructing isosurfaces, direct volume rendering, parallel coordinates which is a common visualization technique used to visualize high-dimensional geometry, tables, matrixes, Maps etc...

Figure 4.1 shows an example of 3D information visualization. The information received by the supported tool or library is a point cloud, or in other words the 3D coordinates of the points, of the specific curves and edges of the human face. These points are extracted using certain object reconstruction methods and can be stored in a file for later use.



Figure 4.1 : Visualization of the Human Face using point clouds. [5]

Figure 4.2 shows another example of 3D information visualization. This figure represents an example for architectural data visualization. It is clearly seen that the data source used here is much larger and contains more detail such as color, texture and depth information.



Figure 4.2 : Visualization of an Architectural Model. [5]

These types of 3D models are visualized using strong visualization toolkits for architectural development and modeling.

As it is seen from the figures above, computer graphics plays a huge role in the area of visualization. Not only can tools be used to visualize certain data but some libraries such as OpenGL (Open Source Graphics Library) or VTK (The Visualization Toolkit) can be used as well, where the developer writes his/her own code by using special functions to visualize the data given. These are only some examples of visualization libraries used in computer graphics programming.

CHAPTER 5

HIDDEN SURFACE DETERMINATION

When visualizing a source of data, such as polygonal meshes, due to the data being 3 dimensional the surface of an object which needs to be behind another object or obstacle may be seen through the first object. Hidden surface determination, which is also known as hidden surface removal or visible surface determination is used to find a solution to such a problem. It determines which parts of an object in a scene is not visible from a certain point of view.

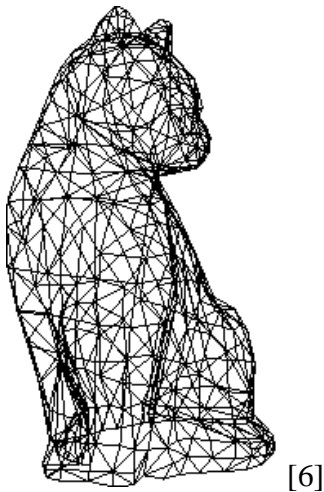


Figure 5.1 : Drawing done without performing hidden surface removal.

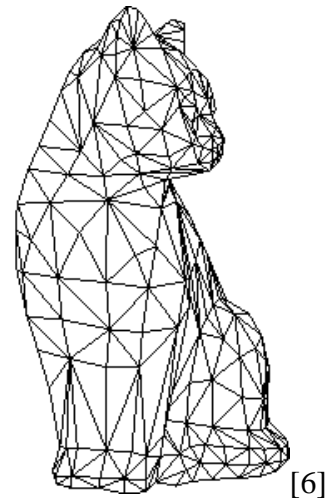


Figure 5.2 : Drawing done by performing hidden surface removal.

Figures 5.1 and 5.2 show a polygonal mesh representation of a cat without performing hidden surface removal and by performing hidden surface removal respectively.

As it is seen in the images, the figure in which hidden surface removal was not performed is complex and uncertain and not close to what the object would look like in real life, on the other hand the figure in which hidden surface removal was performed the front meshes have come forward and the object no longer has a transparent look, which means that the object is rendered in a way in which it would look like in real life.

Another type of removal is the hidden line removal which is used in rendering lines. Figure 5.3 and 5.4 show some polygonal objects drawn with hidden line removal performed and not performed respectively.

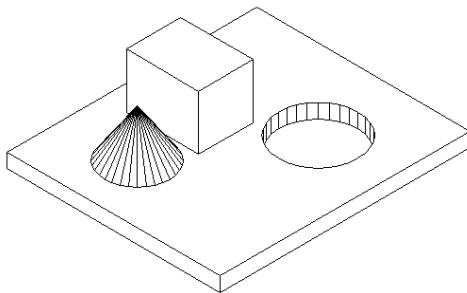


Figure 5.3 : Hidden Line Removal performed. [6]

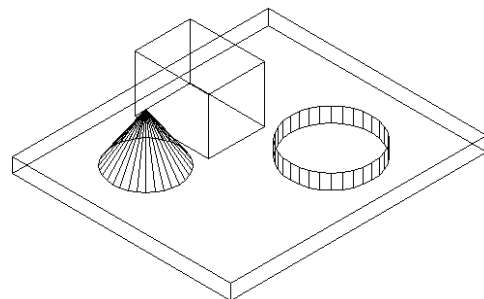


Figure 5.4 : Hidden Line Removal not performed. [6]

There are many algorithms used in order to perform Hidden Surface Removal in the rendering pipeline, the projection, the clipping and the rasterization steps of the the visualization of the scene. All of these steps are handled differently according to the following algorithms;

- Warnock's Algorithm.
- The Painter's Algorithm.
- Binary Space Partitioning.
- Ray Tracing.
- The Z-Buffer Algorithm. (Depth Buffer Algorithm)

5.1 Warnock's Algorithm

Warnock's Algorithm divides the screen into smaller areas and sorts out the triangles within these areas. If there is ambiguity (i.e., polygons overlap in depth extent within these areas), then further subdivision occurs. Subdivision may occur down to the pixel level, which is the limit of the performance of the algorithm [7].

Figure 5.5 shows an example of the application of the Warnock Algorithm.

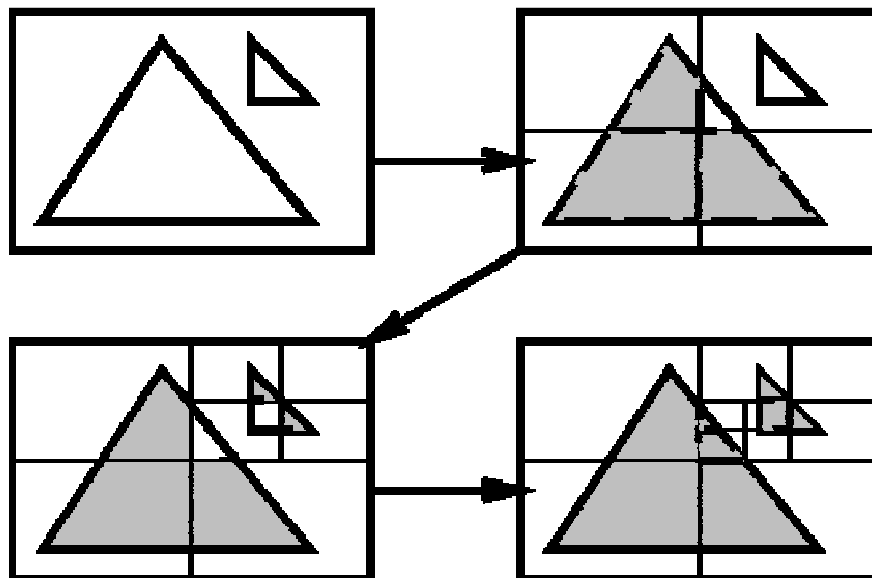


Figure 5.5 : The Application of Warnock's Algorithm. [8]

A simple implementation of this algorithm can be given by the following steps [8];

1. Take a given section of the screen. (In the first run it is the entire screen.)
2. Check to see that it is "simple enough". The meaning of simple enough is; no more than one polygon in the viewport [9].
3. If it is simple enough, display it.
4. If it isn't then subdivide the screen into four sections and begin from the first step.

You can see the above algorithm applied in figure 5.5.

5.2 The Painter's Algorithm

The painter's algorithm, also known as a priority fill, is one of the simplest solutions to the visibility problem. When projecting a 3D scene onto a 2D plane, it is necessary at some point to decide which polygons are visible, and which are hidden [7].

The name "painter's algorithm" refers to a simple-minded painter who paints the distant parts of a scene at first and then covers them by those parts which are closer. The painter's algorithm sorts all the polygons in a scene by their depth and then paints them in this order. It will paint over the parts that are normally not visible -- thus solving the visibility problem -- at the cost of having painted unnecessary areas of distant objects [7].

Figure 5.6 shows the simple steps of how the Painter's Algorithm works.

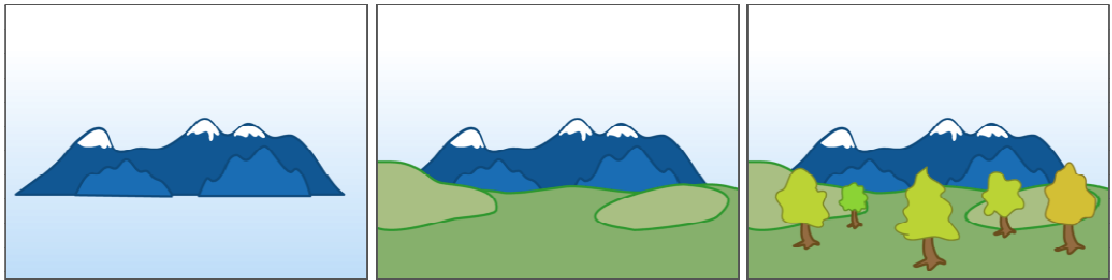


Figure 5.6 : The mountains are painted first, and then the meadows and finally the trees up close are painted last. (Simple example of the Painter's Algorithm) [7]

5.3 Binary Space Partitioning (BSP)

Binary space partitioning is a generic process of recursively dividing a scene into two until the partitioning satisfies one or more requirements. The specific method of division varies depending on its final purpose. For instance, in a BSP tree used for collision detection, the original object would be partitioned until each part becomes simple enough to be individually tested, and in rendering it is desirable that each part be convex so that the painter's algorithm can be used. [7]

It divides a scene along planes corresponding to polygonal boundaries. The subdivision is constructed in such a way to provide an unambiguous depth ordering from any point in the scene when the BSP tree is traversed. The disadvantage here is that the BSP tree is created with an expensive pre-process. This means that it is less suitable for scenes consisting of dynamic geometry. The advantage is that the data is pre-sorted and error-free [7].

The following figure shows an example of how Binary Space Partitioning works, in a simple way.

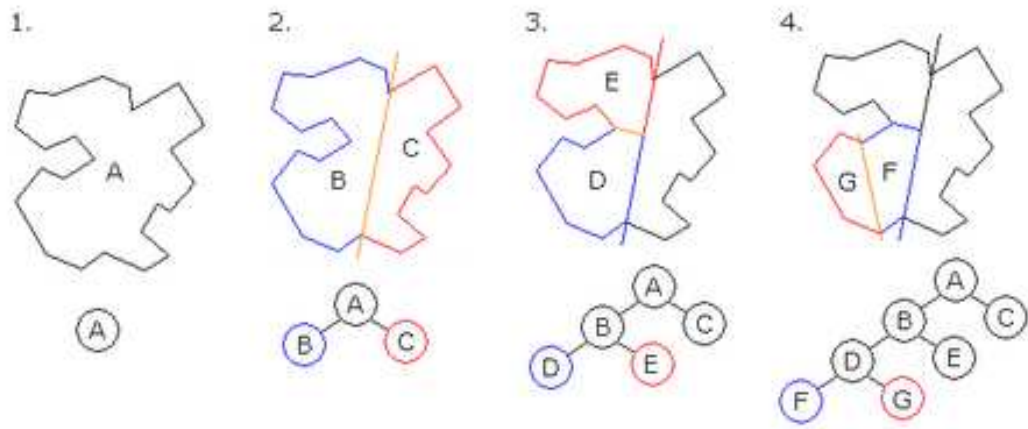


Figure 5.7 : The few simple steps of Binary Space Partitioning. [7]

As it is seen, A is the root of the tree and the entire polygon. A is split into B and C. Then B is split into D and E. And in the last part D is split into G and F which are convex and hence become leaves on the tree. [7]

5.4 Ray Tracing

Ray tracing; attempts to model the path of light rays to a viewpoint by tracing rays from the viewpoint into the scene. Although it may not be considered as a hidden surface removal algorithm, it implicitly solves the hidden surface removal problem by finding the nearest surface along each view-ray. Effectively this is equivalent to sorting all the geometry on a per pixel basis [7].

The following figure shows an example of how ray tracing works.

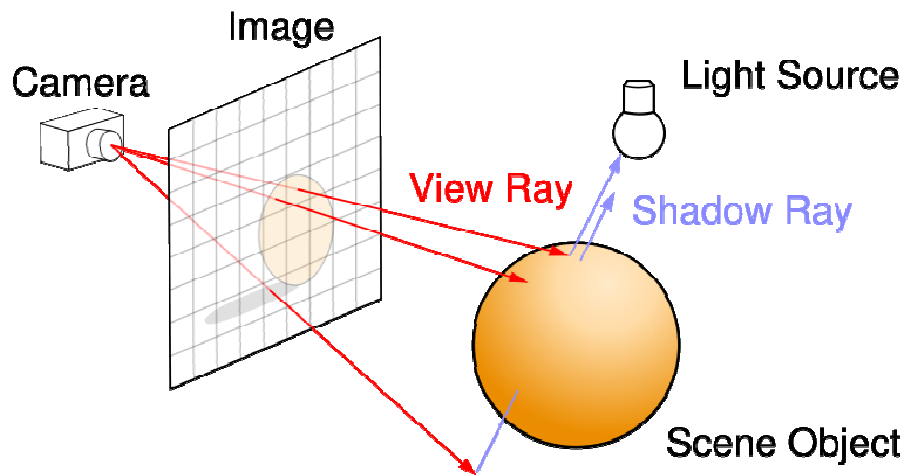


Figure 5.8 : The Ray Tracing Algorithm. [7]

5.5 The Z-Buffer Algorithm (The Depth Buffer Algorithm)

The final algorithm and the algorithm which was used in this experiment is the Z-Buffer Algorithm. It is one of the simplest and most easily implemented methods for removing hidden surfaces. Though there are some limitations to this algorithm such as large amount of memory usage and it often renders an object that is later on neglected by an object that is rendered which is nearer, which means that the amount of time spent for the first object is wasted.

The reason this algorithm is called the Z-Buffer algorithm is because the z -coordinate that represents depth value is used.

Figure 5.9 shows a frame buffer along with its depth buffer.

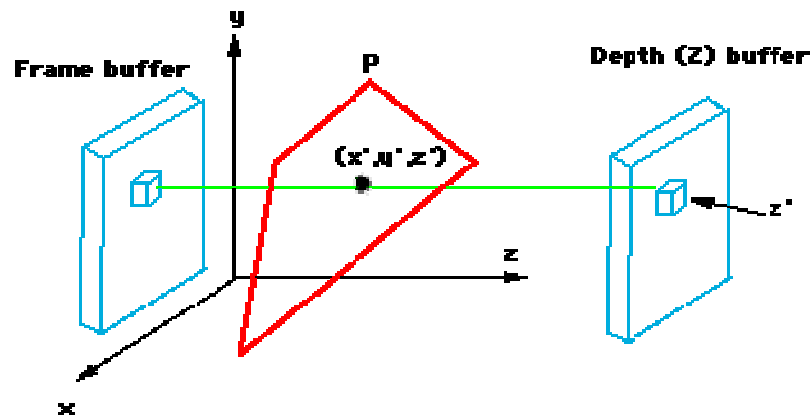


Figure 5.9 : Conceptual view of the depth buffer. [10]

For every pixel $p[x][y]$ the depth buffer stores a b-bit (that is usually in the range from 12 to 30) quantity $d[x][y]$. During the rendering process the depth buffer, $d[x][y]$ contains the pseudodepth (provides an adequate measure for pixel p) of the closest object encountered so far at the pixel $p[x][y]$. As the algorithm proceeds, “tile” by “tile” (we can consider each pixel as a tile), it checks and compares the pseudodepth (z value) of the current tile with the depth $d[x][y]$ stored in the depth buffer. If it is less than the value stored in the depth buffer then the color of the closer “tile” or surface replaces the color stored in pixel $p[x][y]$, and the smaller pseudodepth value replaces the old value in the depth buffer $d[x][y]$.

The faces can be drawn in any order, though as mentioned above if the surface that is far away is drawn first then the surface that is near will be drawn on top of it which yields into a waste of time spent on the drawing of the far surface.

This algorithm can be used on any surface and works for any type of object shape, including curved shapes, since it finds the closest surface based on a point-by-point (pixel-by-pixel) testing.

Figure 5.10 shows a simple example of how the Z-Buffer Algorithm works. A tile sample is used to visually show how each tile is drawn step by step.

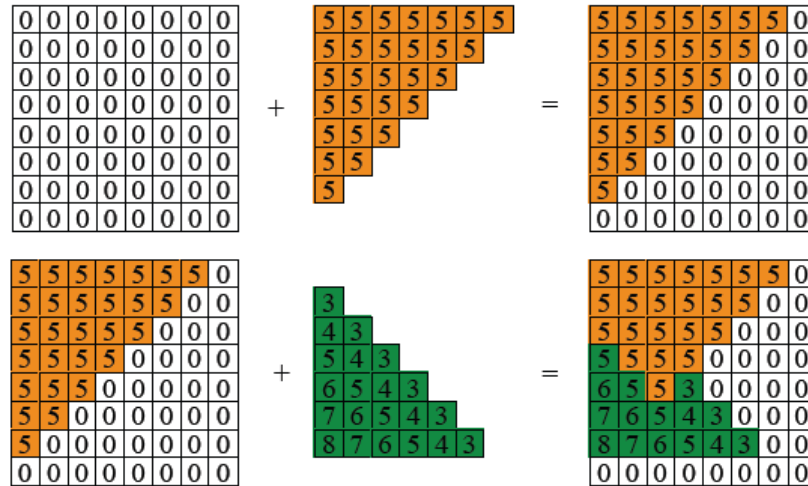


Figure 5.10 : The Z-Buffer Algorithm [10]

As seen in the figure the array $d[x][y]$ is initially loaded with the value “0”. Though usually the depth buffer $d[x][y]$ is initially loaded with the value 1.0 since it is the greatest pseudodepth value possible. The frame buffer on the other hand is initially loaded with the value of the background color.

Now that we have given the information about the Z-Buffer let’s find out how we can find the pseudodepth of each pixel.

In order to compute the pseudodepth we need a fast method. Recall that each vertex $P = (P_x, P_y, P_z)$ of a face is sent down the graphics pipeline and goes through various transformations. The information of each vertex after these transformations is the scaled and shifted version of the following equation;

$$(x, y, z) = \left(\frac{P_x}{-P_z}, \frac{P_y}{-P_z}, \frac{aP_z + b}{-P_z} \right) \quad (5.1)$$

The third component is the pseudodepth. The constants a and b are used so that the third component equals zero if P lies in the near plane and equals to one if P lies in the far plane.

For more efficiency the pseudodepth at each pixel is computed along a scan line incrementally as it is done for the color component of each pixel.

Figure 5.11 shows a face being filled along the scan line y_s . The pseudodepth values at certain points are marked.

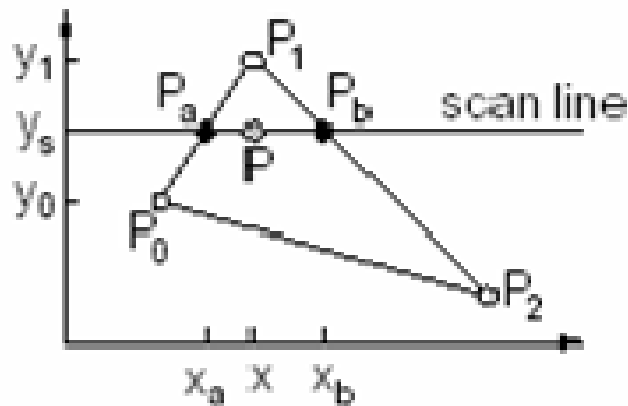


Figure 5.11 : Computation of pseudodepth. [10]

Let's say the pseudodepth values at the vertices P_0 , P_1 and P_2 are known, our aim is to calculate the pseudodepth value at point P_a , on the scan line y_s . So we have the following;

$$P_{a(z)} = \text{lerp}(P_0, P_1, f) = d_{\text{left}}$$

$$\text{Where } f = (y_s - y_0) / (y_1 - y_0) \text{ and} \quad (5.2)$$

$$\text{lerp}(a, b, t) = a + (b - a) * t$$

To find the pseudodepth value at point P_b , on the scan line y_s , we use the following;

$$P_{b(z)} = \text{lerp}(P_1, P_2, h) = d_{\text{right}}$$

$$\text{Where } h = (y_s - y_2) / (y_1 - y_2) \text{ and} \quad (5.3)$$

$$\text{lerp}(a, b, t) = a + (b - a) * t$$

So in general; to find the pseudodepth value d at each pixel (x, y) along the scan line we can use the following;

$$d(P_x) = \text{lerp}(d_{\text{left}}, d_{\text{right}}, k)$$

$$\text{Where } k = (x - x_a) / (x_a - x_b) \text{ and} \quad (5.4)$$

$$\text{lerp}(a, b, t) = a + (b - a) * t$$

(Not: **Lerp** is a quasi-acronym for *linear interpolation*.)

Now that we have defined the Z-Buffer, the question is; why is the Z-Buffer algorithm so popular and why is it the most commonly used algorithm among the hidden surface removal algorithms? Let's list the advantages of the Z-Buffer algorithm to give us an idea of why it is used so commonly.

- It is simple to implement in hardware.

- It supports non-polygonal primitives.
- It does not have any limit in scene complexity.
- The depth values calculated can be saved for later use or for other uses.

Along with the advantages there are also some disadvantages of the Z-Buffer algorithm.

- It uses up extra memory (one storage cell for one pixel) and bandwidth.
- It wastes time drawing objects that may turn out to be hidden afterwards. So it may draw the same pixel more than once.
- Certain errors that may be done with the Z precisions lead to depth aliasing.

If we should make a brief comparison of the three algorithms; Painter's Algorithm, Warnock's Algorithm and Z-Buffer Algorithm, it shall look like the following table [6];

Painter's Algorithm	Warnock's Algorithm	Z-Buffer Algorithm
<ul style="list-style-type: none"> - Device independent. - Details are tough. - Algorithm is slow. 	<ul style="list-style-type: none"> - Semi-device dependent. - Easy to implement. - Not very fast. 	<ul style="list-style-type: none"> - Device dependent. - Easy to implement. - Fast algorithm. - Memory intensive. - Algorithm of choice for hardware.

The reason we are not including the other two algorithms into the table is because the Ray Tracing algorithm is actually not considered as a hidden surface removal algorithm and the Binary Space Partitioning algorithm is not a convenient algorithm. The three algorithms compared above are the most widely used algorithms for hidden surface detection and removal.

Another algorithm, Backface Culling, which we haven't mentioned above is another algorithm that is used for hidden surface removal. Even though this algorithm is fast it is insufficient when used by itself.

CHAPTER 6

SUPERIMPOSITION

In computer graphics, superimposition is the placement of an image or video on top of an already existing image or video. This is usually done to add effects to the video or image or to just conceal the original image (such as superimposition of a face on the original face in an image).

Figure 6.1 shows an example of 3D superimposition into a video sequence. A model of a teapot has been placed onto a certain area on the table and on the floor. Though determining that location is another step which need to be taken in order to place the modeled object on that specific location.

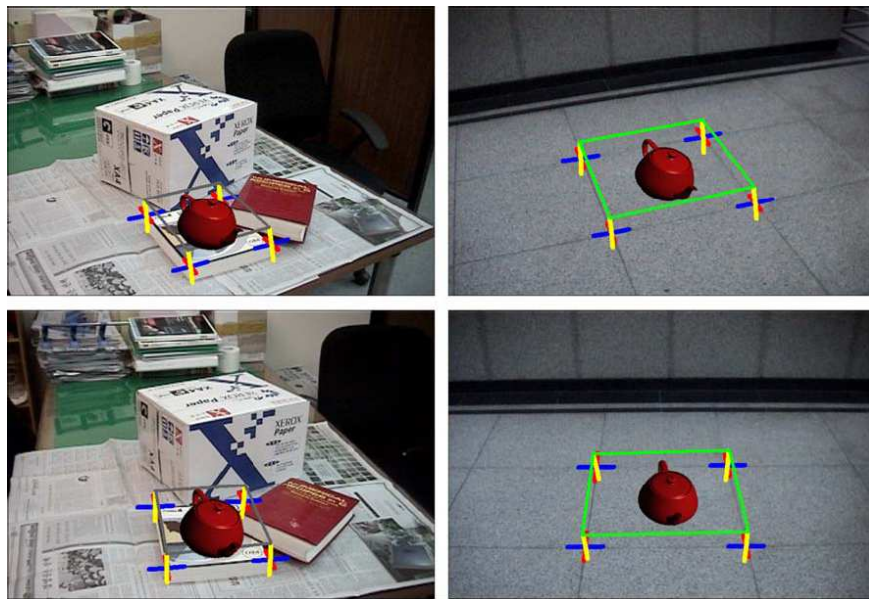


Figure 6.1 : Superimposition of a 3D teapot model. [11]

Superimposition can be done with 3D or 2D objects. You can superimpose lines or dots to emphasize a certain region in an image as well as place an entirely different modeled object onto another object in the image. These processes can both be done in video sequences or images.

A common application area of superimposition is augmented reality. The image above is an example of augmented reality.

If we shall give a brief definition of augmented reality (AR); it combines a virtual environment with the real world, in order to help people to understand the real world more easily by providing additional information about interesting objects in the real environment. An AR system should be able to [11];

1. Combine real environments and computer-generated virtual objects,
2. Operate virtual objects interactively with the change in the real world,
3. Align virtual graphic objects onto real environments.

CHAPTER 7

METHODS AND ALGORITHMS

The application consists of certain steps to perform the visualization of an object that is being tracked in a tracking system. Though the system first goes through certain calculations and observations such as object recognition and object reconstruction. This application of the thesis is the part of the system in which it receives the data from those two methods and then performs its task.

The experiment includes many inputs taken from other experiments, which is why it requires a lot of data which was calculated before.

7.1 Steps Followed During the Application Process

The whole application process went through a series of steps to achieve the goal set for starting this experiment. The steps include the methods and theory mentioned in the previous chapters. We explain how these methods were applied in the following headlines.

7.1.1 Receiving Image Data from Two Stereo Camera's

Before the process started, in another application camera calibration was done using two stereo camera's and with the use of that, object reconstruction and object recognition was applied to the images received. From this application the 3D edge points and 3D corner points were found, and the objects' other features were found and stored in a file.

In the experiment these files were used as a basis for beginning the process of implementation which would lead on to the main idea in the purpose of this thesis.

7.1.2 Using the Geometric Transformation Methods

The data stored in the files mentioned above were in numeric format including the 3D edge points and corner points coordinates and the calibration information of the camera's (intrinsic parameters and extrinsic parameters).

Since we assumed that the third camera we were looking through was the same, we used the same calibration data that belonged to either the first or second camera to perform the geometric methods.

We applied this to still images since the data received was not from a video sequence. Each view was a separate image taken from different view points. We considered each view point to be a view point from a fixed camera.

Figure 7.1 shows the view points used in the previous experiment in which the image data was received from and Figure 7.2 shows the view point used in the basis of this experiment.



Figure 7.1 : The point of views used in the previous experiment.



Figure 7.2 : The point of view used in this experiment.

As mentioned above we used the same calibration data of the first two camera view points for the third view point as well. So the image size was the same, the camera's intrinsic parameters were the same and the distortion coefficients were considered the same.

The rotation matrix and translation vectors were also received from the calibration method of the previous experiment.

With all the data in hand we had to calculate the world coordinates first using the data we had received. We used the following equation for calculating the world coordinates;

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = R_2 * \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + t_2 \quad (7.1)$$

Here the vector containing (x_2, y_2, z_2) are the 3D coordinates of the edge points received in the data file (the same process was done for the 3D coordinates of the corner points as well). R_2 is the rotation matrix belonging to the second camera. The vector (x_w, y_w, z_w) are the world coordinates that is going to be calculated and t_2 is the translation vector of the second camera.

So in this equation we have one unknown which is the 3D world coordinates that we need to calculate to move on to the next step. So with basic arithmetic we extracted the world coordinates as follows;

(Let's replace the second camera coordinate vector with; X_2 and the world coordinate vector with; X_W .) By using the equation 7.1 we can have the following;

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = R_2 * \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + t_2 \quad \rightarrow \quad X_2 = R_2 * X_W + t_2$$

Now let's extract the world coordinates;

$$X_2 - t_2 = R_2 * X_w$$

$$(X_2 - T_2) * R_2^{-1} = R_2 * X_W * R_2^{-1}$$

(R_2^{-1} is the transpose of the matrix R_2)

$$(X_2 - T_2) * R_2 = X_W$$

(calculated using equation 7.1)

With the equation above we have calculated the 3D world coordinates. By finding the world coordinates or in other words the representation of the 3D coordinates in the world space, we can convert the coordinates to any type of coordinate system. Our aim was to find the 3D coordinates of the points in the third camera view point. So we used the following equation (taken by using equation 7.1) again to find the 3D coordinates of camera three;

$$X_3 = R_3 * X_W + t_3$$

Where X_3 represents the coordinate vector (x_3, y_3, z_3) of camera three and R_3 and t_3 are the rotation matrix and translation vector respectively of the third camera. Though before calculating the world coordinates the rotation data received from the previous experiment needed to be transformed. Because after calibration the rotation data was in the form of a vector, though to use it in the equation we first needed to transfer the rotation vector of the third camera to a 3x3 rotation matrix. That way we could calculate the X_3 coordinates using the above equation. The transfer of the rotation vector to a rotation matrix was done by using a special function in OpenCV.

The implementation of this part was done by using the open source library, OpenCV. Microsoft Visual Studio 6.0 was used as the platform for implementation. The programming language used for the implementation of the code was C.

7.1.3 Performing Perspective Projection

After calculating the 3D coordinates it was time to use perspective projection to project the points calculated onto the image to test whether the points have been transformed correctly according to the third camera view point.

The 3D point coordinates were read from an input file and then given to a special function used in OpenCV. The function receives the object points, the translation and rotation vectors of the coordinate system belonging to the third camera and the third camera's intrinsic parameters and distortion coefficients. As a result it gives us the matrix of image points on the 2D image plane.

In order to test whether the projection of the points was done right, dots were drawn on the image where the projected points belonged. At first by giving the rotation and translation vectors the projected points did not align with the points in which they should have. This is shown in figure 7.3.



Figure 7.3 : Erroneous Projection

As it is faintly seen in the image above only one dot was drawn on an axis of the image yet the other points are nowhere to be found. From this solution it was understood that the function used in this method translated and rotated the already translated and rotated points. In fact the points in which were given as an input were already rotated and translated according to the following equation;

$$X_3 = R_3 * X_W + t_3 \quad (\text{see equation 7.1})$$

After realizing the reason for the erroneous output, instead of giving the rotation and translation vectors as inputs zero vectors were given, along with the other parameters mentioned above, so that the points would be projected directly without an change. And with having done this the output image turned out to look like the image predicted at the beginning. The result of the projection of the 3D edge point coordinates are shown in figure 7.4.



Figure 7.4 : Projection of the 3D edge point coordinates.

The same method was used to project and draw the 3D corner point coordinates as well. That application is shown in Figure 7.5.

It is seen in both of the figures that the projected edge point coordinates and corner point coordinates all overlap on the points in which they were suppose to. This shows that the geometric methods used were correct and the calculationns done were correct as well.



Figure 7.5 : Projection of the 3D corner point coordinates.

7.1.4 3D Visualization (3D Data Visualization)

In the process of 3D Visualization the open source graphics library, OpenGL, was used.

As an input the calculated 3D edge point coordinates were used. The aim was to draw the coordinates in the 3D format, in order to visualize the 3D edge points in a discrete environment. For this method a function in the OpenGL library was used. Though due to the lack of surface point coordinates and other types of features such as lighting effects and texture we needed to use another mesh file in order to test the algorithm. The output of this test will be explained and shown in section 7.1.5 Performing the Z-Buffer (Depth Buffer) Algorithm.

The following figure shows the output of the visualization of the 3D edge point coordinates of the third point of view.

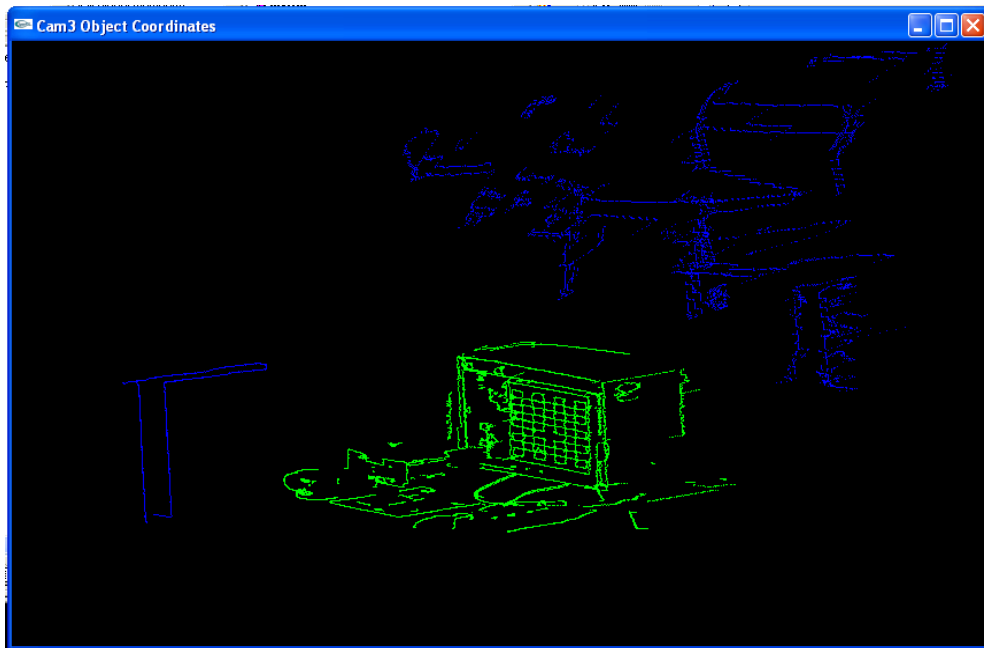


Figure 7.6 : 3D Visualization of 3D edge point coordinates.

As it is seen in the image, there is something wrong in the way that the points have been drawn. The image above does not look exactly like the original image. The reason for this is the way that the points have been projected to the 2D screen. The function used actually projects the points orthogonally, meaning directly without giving any perspective. The perspective comes from the points itself. The edges which are blue are the ones that are far away, and the green edges are the edges that are closer to the viewer.

This image is not enough though. As an input there must also be texture or surface information (for example 3D surface point coordinates) at least in order to perform the next task. These informations are received from object recognition.

From that method we will receive a mesh file including every data, such as vertices, faces, and normals of the faces or vertices. By using these data's we form a 3D triangular mesh of the object being tracked.

7.1.5 Performing the Z-Buffer (Depth Buffer) Algorithm

The Z-Buffer algorithm was applied to another data file to test the algorithm. Since the data file calculated did not include the surface data. In order to perform the Z-Buffer algorithm the data that will be drawn needs to include 3D point coordinates of the surface of the object as well, otherwise, by using only edge point coordinates, the elimination of the hidden parts can not be done, which means that the Z-Buffer algorithm will not work properly.

As mentioned before the Z-Buffer algorithm eliminates the hidden surfaces, meaning it only shows what is suppose to be seen not the objects that remain behind of other objects.

By using the sample test data to test the Z-Buffer algorithm it was seen that it worked. The test data file included the vertex's of the polygonal surfaces in the image and the point coordinates of the surfaces. With this data the following algorithm was used in order to apply the hidden surface method;

```
For each polygon P
  For each pixel (x, y) in P
    Compute z_depth at x, y
    If z_depth < z_buffer (x, y) then
      set_pixel (x, y, color)
      z_buffer (x, y) = z_depth
```

The algorithm shown above is done during the visualization period of the program. Since the Z-Buffer algorithm works in scan line format it checks the Z value of the pixel and then draws it on the screen.

The Z-Buffer defined was first initialized to 1.0, since 1.0 is the largest value stated for the Z-Buffer, because the range of the buffer was taken from 0.0 to 1.0. So the Z values in which we are dealing with are actually $1/Z$. By doing this it helped us to linearly interpolate the Z values along the polygon edges. This also means that Z (Z^{-1}) has been inverted, so points that are far away will have small numbers, and points that are close to the viewer will have larger numbers. The calculation was implemented as follows; as the 3D points were projected onto the screen, the $1/Z$ value at that x, y point was stored.

This continued on until every polygonal edge was drawn onto the image screen. The surfaces and edges that were in the back did not appear. So the Z-Buffer algorithm worked successfully. Instead of having the wire frame, transparent view, as shown before in the visualization of the edge points we have a non-transparent though again wire frame view due to the data. If we include the texture as well then we will have a complete solid model view of the environment and object.

Normally the data file will contain the surface point data along with the edge data, of the scene in which we are viewing, as well. These data's will be received from the results of object recognition and segmentation (accomplished in other experiments).

Though in order to test the algorithm that had been written another sample mesh file was used. This mesh file was compiled using the Z-Buffer algorithm, which is why this example is being explained in this part of the thesis; to show how the algorithm worked as a whole. Since a proper visualization can not be done without the use of the Z-Buffer algorithm.

Our example mesh file was the simple Barn example. This example is usually used in Computer Graphics text books.

The following figure shows how the file looked like.

```
10 7 7
0 0 0 1 0 0 1 1 0 0 0.5 1.5 0 1 0
0 0 1 1 0 1 1 1 1 1 0.5 1.5 0 1 1
-1 0 0 0 -0.707107 0.707107 0 0.707107 0.707107
1 0 0 0 -1 0 0 1 0 0 0 -1
4 0 5 9 4 0 1 2 3
4 3 4 9 8 0 1 2 3
4 2 3 8 7 0 1 2 3
4 1 2 7 6 0 1 2 3
4 0 1 6 5 0 1 2 3
5 5 6 7 8 9 0 1 2 3 4
5 0 4 3 2 1 0 1 2 3 4
```

Figure 7.7: Barn Mesh File

The format of the file was like the following; The numbers at the beginning of the file indicate the number of vertices, the number of normals and number of faces in the mesh. Each vertex is listed below it and following it are the normals to those vertices. Next, each face is listed containing the number of vertices in the face, the vertex list and the normal list for the vertices.

By using this very simple file we managed to test our algorithm and we obtained the solution seen in figure 7.8. The solution does not only contain the edges and vertices it also contains the shading and light information as well.

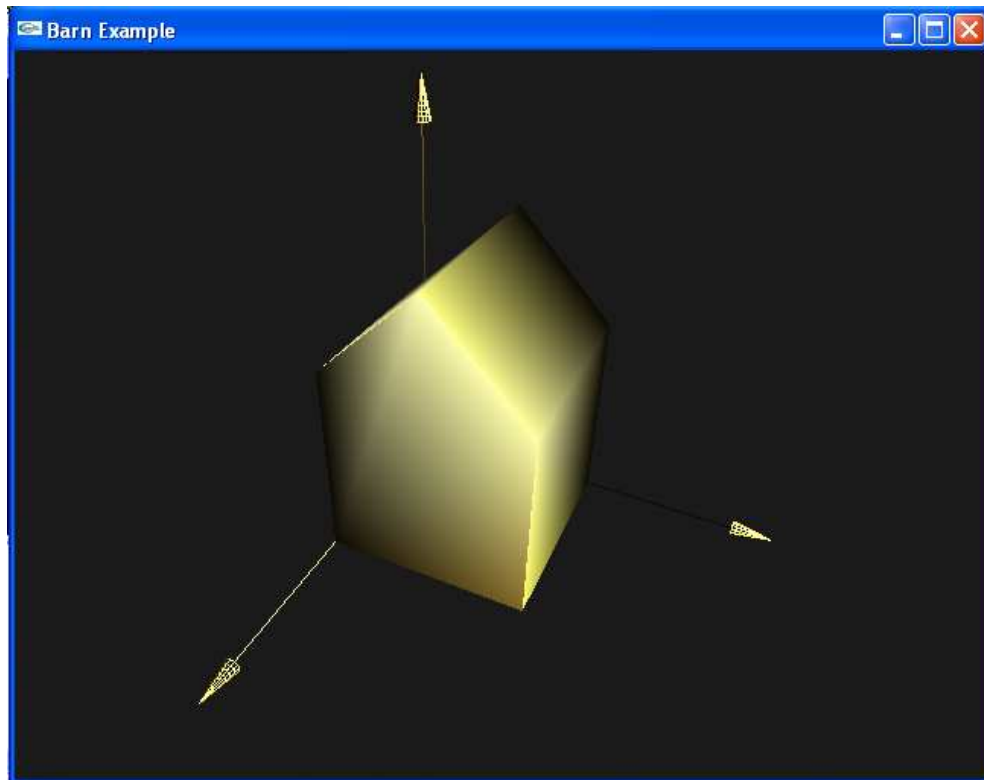


Figure 7.8: Visualization of the Barn Example.

As it is seen in the figure the part of the barn that remains in the back is not seen, this shows us that the Z-Buffer algorithm has worked. Our visualization algorithm has worked also.

7.1.6 Simple Superimposition

After having modeled the object our intentions was to apply a very simple superimposition method to locate the whereabouts of the object, since we are considering of tracking a certain object in a 3D environment. Though since we did not have enough feature points to define the object, such as we mentioned in the

previous sections, we only considered about drawing a circle or rectangle that shows us the objects whereabouts.

If we can consider the boxes in the following two figures;

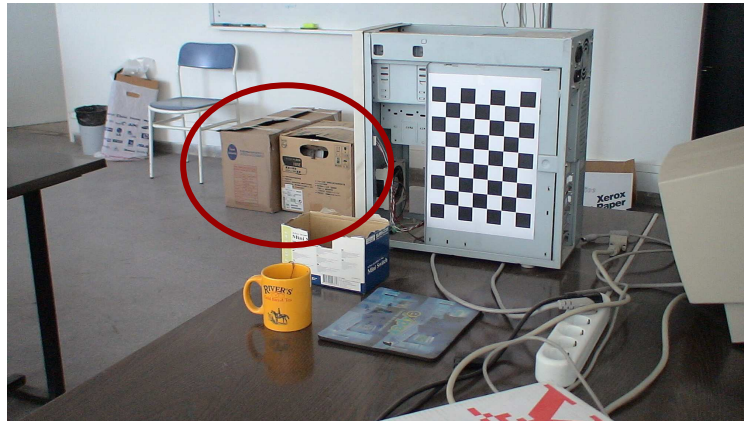


Figure 7.9: Point of view of first camera.



Figure 7.10: Point of view of second camera.

As it is seen in the figures the boxes are visible. These were the point of views from the two cameras' we had taken data from. In our case we can not see the box, it is somewhere behind the computer.

The following figure shows us the point of view of the third camera in which we used to apply the algorithms.

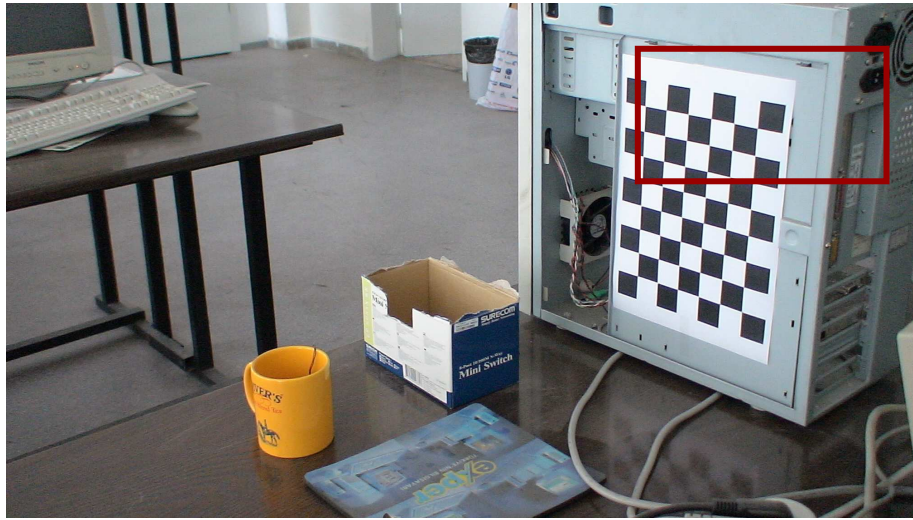


Figure 7.11: Point of view of the third camera.

The red rectangle was superimposed to show us approximately where the boxes in the figure may be.

Superimposition was used in this way. A more advanced use of superimposition would be to superimpose a virtual 3D model of the object which we explain in the next chapter.

7.2 Experimental Results

As a result the algorithms used for this part of the system all worked though were tested with different testing data due to the lack of data belonging to the same system.

If the data used in the object tracking experiment and the object recognition and reconstruction experiments were all the same sample data it would have been

better to show how the system would've worked as a whole. Though we only got to test the algorithms using different testing data, which led to different views and solutions. But the algorithms can work on any type of data received as long as the data is in the suitable format.

This experiment requires various inputs received from other experiments. Which makes it difficult to achieve if the system is not well set.

CHAPTER 8

CONCLUSION AND FUTURE WORK

As a conclusion of this thesis we managed to give the viewer from the third camera an idea of where the object, in which the system it was being tracked, is.

The results are not precise since we have not used any statistical computations, but the results still give us an idea of the whereabouts of the object do to the visalization done. The program has a restriction on the input data given to pursue the process. The data needs to be the 3D point coordinates of the vertices and edges of the object along with the 3D point coordinates of the polygonal surfaces that were calculated using the segmentation method in another experiment. With these data's the program will work. In order to expand the program, so it can work for texture data and other details certain additions must be made to the algorithm.

As mentioned above the program provides a view of the object in the third camera. This object may be behind an obstacle where it can no be seen at all or it can remain behind another object but can be partly seen, meaning a certain portion of the object can no be viewed. Even in these two cases the program is capable of telling us the whereabouts of the object.

Since we tend to use this program in a an object tracking system, the amount of data given as an input to the program will be quite large. The processing of this data will slow down the run time of the system. The reason for the data being so large is because of the amount of detail that will be added for the determination and modeling of the object. In our case we only have 3D corner and edge coordinates and 3D surface coordinates, though in the more complex case we will have texture data, distance data and intensity data. These data's will help us render the object in a more realistic way. Though with this data the Z-Buffer algorithm will work much slower since it already has a tendency of working slow due to the amount it spend on drawing points that may be eliminated later on.

To reduce the amount of time spent optimization can be applied to the calculations. For example not all data may be used for certain processes in the program, such as the Z-Buffer algorithm, since surface data is sufficient for the Z-Buffer algorithm to work.

As future work this thesis can be developed into a system that shows us a 3D virtual model of the object superimposed on the real object itself during the tracking process. The third camera or other camera's can then tell us the exact location of the object along with it's features. This way the camera will act sort of like a human eye focusing only on one object.

In order for this to take place a 3D model of the object must be rendered using a graphics library. The model does not need to be perfectly rendered, meaning that it is not necessary for the object to look realistic, as long as the features are clearly pointed out. Later on this 3D model can be superimposed onto the original view of the object in the viewing screen of the third camera. The 3D model can be

opaque or transparent depending on the level of detail in which the object wants to be tracked. By pursuing such an application the object in which we intend to track will not be lost by the tracking system. Each camera in the environment, will be able to show the exact location of the object along with its features. So as long as the object moves around in the boundaries of the tracking system we will not lose sight of it. The camera's used in the system may be fixed or rotatable. They will be sending each other the data they calculate once one of them loses sight of the object, the other one will continue on with tracking the object until that one has lost sight as well. This process will continue on until the object is totally lost.

Though this application has certain problems. Since we want to track the object in full detail the rendering process will take up most of the run-time of the program. So until the object is superimposed on its original view it may have already moved to another location or even have left the tracking systems boundaries, which will give us false information of the location of the object. Another time consuming process will be the transfer of such large amount of data from one camera to the other. This will effect the system in a negative way since the tracking system will be working in real-time. So as the process time of calculation and drawing increases the system will be far away from becoming a real-time application. These problems can be over come with certain optimizations and better hardware, though further studies are also required.

REFERENCES

- [1] http://faculty.cs.tamu.edu/jchai/CPSC641_fall07/PerspectiveProjection.pdf

- [2] <http://www.roysac.com/blog/2007/10/perspective-projection-on-computer.html>

- [3] <http://opencvlibrary.sourceforge.net/>

- [4] **MCCASLIN, P.T., MCDONALD, P.A., and SZOKE, E.J.** (2000) 3D Visualization Development at NOAA Forecast Systems Laboratory, *COLUMN: Contributions: Focus: New Visualization Techniques*, 41-42. Vol. 34.

- [5] http://www.yuhui-fu.net/Images/JPEG/Project%20Pictures/Point_Cloud.jpg

- [6] <http://medialab.di.unipi.it/web/IUM/Waterloo/node70.html>

- [7] <http://en.wikipedia.org>

- [8] <http://medialab.di.unipi.it/web/IUM/Waterloo/node68.html>

- [9] <http://medialab.di.unipi.it/web/IUM/Waterloo/node69.html>

- [10] **JANOWSKI, A., SAWICKI, P., SZULWIC, J.** (2005) Advanced 3D Visualization of an Architectural Object in the OpenGL Standard, *PanoPhot05*, Berlin
- [11] **PARK, S.C., LEE, S.W. and LEE, S.W.** (2006) Superimposing 3D Virtual Objects using Markerless Tracking, *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, 897-900. Vol. 3.
- [12] **FAUGERAS, O., LUONG, Q.T.**, (2001), *The Geometry of Multiple Images: The Laws that Govern the Formation of Multiple Images of a Scene and Some of Their Applications*, The MIT Press, Massachusetts
- [13] **HARTLEY, R., ZISSERMAN, A.**, (2000), *Multiple View Geometry in Computer Vision*, Cambridge University Press, Cambridge.
- [14] **EICK, S.G** (2000) Visualizing Multi-Dimensional Data, *COLUMN: Contributions: Focus: New Visualization Techniques*, 61-67
- [15] <http://www.cs.helsinki.fi/~piilopinnat/warnock.html>
- [16] <http://www.dgp.toronto.edu/~patrick/csc418/notes/pseudodepth.pdf>
- [17] <http://medialab.di.unipi.it/web/IUM/Waterloo/node70.html>
- [18] **HILL, F.S., Jr.**, (2001) , *Computer Graphics Using OpenGL*, second edition, Prentice Hall, New Jersey.
- [19] **GONZALEZ, R.C., WOODS, R.E.**, (2002), *Digital Image Processing*, second edition, Prentice Hall, New Jersey.
- [20] <http://genex.hgu.mrc.ac.uk/Software/paint/paint/node5.html>
- [21] <http://www.gamespp.com/graphicsprogramming/ZBufferAlgorithm>

- [22] **STASKO, J.T.** (1993) Three-Dimensional Computation Visualization, *Visual Languages, 1993., Proceedings 1993 IEEE Symposium on*, 100-107.

- [23] **RAMAMOORTI, R.** Creating Generative Models from Range Images, Masters Thesis

- [24] **FORSYTH, D.A., PONCE, J.,** (2003), *Computer Vision – A Modern Approach*, Prentice Hall, New Jersey.

- [25] <http://www.siggraph.org/education/materials/HyperGraph/modeling>