Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

Title of Thesis: 3-D Structure Recovery
Using Unified Optical Flow Field Approach.

Anees Ahmad, Master of Science in Electrical Engineering, 1991
Department of Electrical and Computer Engineering

Thesis directed by: Dr. Yun Q. Shi
Assistant Professor
Department of Electrical and Computer Engineering

Recovery of the 3-D structure that is characterized by an N^{th} order factorable and/or non-factorable polynomial equation is studied in this thesis. Analytical tools have been employed to analyze and solve the problem. The solution is based on the direct method, which is derived from the unified optical flow field, which, in turn, is an extension of optical flow to spatial image sequences. Least squares formulation has been employed for optimum estimation. The method used does not require the establishment of point to point correspondence, nor the estimation of the optical flow, as an intermediate step.

The theoretical results are tested using some synthetic images of various structures including cone, parabola and the combination of a cone and a planar surface. Satisfactory results are obtained, thus demonstrating the validity of the newly developed technique for the recovery of a variety of surfaces [14].

3-D Structure Recovery
Using Unified Optical Flow Field Approach

by

Anees Ahmad

Thesis submitted to the Faculty of the Graduate School of
the New Jersey Institute of Technology in partial fulfillment of
the requirements for the degree of
Master of Science in Electrical Engineering
1991

APPROVAL SHEET

Title of Thesis: 3-*D* Structure Recovery
Using Unified Optical Flow Field Approach

Candidate: Anees Ahmad
Master of Science in Electrical Engineering, 1991

Thesis and Abstract Approved by the Examining Committee:

Dr. Y.Q. Shi, Advisor Assistant Professor Department of Electrical and Computer Engineerig	Date
Dr. J. Carpinelli Assistant Professor Department of Electrical and Computer Engineerig	Date
Dr. C.Q. Shu Research Associate Department of Electrical and Computer Engineerig	Date

New Jersey Institute of Technology, Newark, New Jersey.

VITA

Anees Ahmad

75 Belgrove Drive, Apt. 4B

Kearny, NJ 07032

201-998-5152

Date of Birth:

Place of Birth:

Education: *1989-1991*

New Jersey Institute of Technology, NJ: MSEE

1983-1989

N.E.D. University of Engineering & Technology, Karachi, Pakistan: BSEE

ACKNOWLEDEGMENT

The author is deeply indebted to several people for their valuable assistance during the course of this thesis.

In particular, the author wants to thank his thesis supervisor Dr. Yun Q. Shi for his technical guidance and morale support. The author is also thankful to Dr. Chang-Qing Shu, who has always been willing to answer questions and solve difficult problems. The author also wants to express thanks to the other member of the graduate committee, Dr. John Carpinelli, for his assistance and advice in completing this thesis.

Appreciation is also extended to parents and friends, who co-operated and inspired the author greatly during the course of this work.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

The concept of *stereo vision* has emerged as a powerful tool to capture the information about 3-D surface structure. The concept of *stereopsis* is based on essential differences between the images of stereo pair arising out of their different points of view, that is, the same point is viewed from two different locations and angle. It is one of the hot areas of research in the computer vision community and it is expected to find solutions to several problems in the area of machine intelligence and robotics.

The main problem in 3-D surface recovery is to develop algorithms and data structures that extract information from 2-D images (that have gray level character) and manipulate the information to find out the *depth*. The depth or the distance between the object observed and the viewer along the optical axis is of significant importance in various applications of computer vision systems; for example, guidance systems and recognition systems etc.

In the reconstruction of 3-D scene from a digital stereo pair of images, two problems must be solved: (a) Geometrical calculation of 3-D position of the scene point with respect to its stereo projection. (b) The second is rather more

complicated and concerns the problem of correspondence. The correspondence problem needs feature extraction and template matching, which have been found one of the challenging problems in the computer vision community.

The reconstruction of 3-D surface is important, also because the extraction of the information about the surface structure is mainly responsible for the success of further processing of the information. In this thesis the spatial domain domain has been adopted. It has several advantages. For example, it eliminates several constraints imposed by the time-domain analysis, yet the results achieved match very well with any other technique for surface reconstruction.

There are basically two different approaches to recovering the structure of object(s) and the relative motion between object(s) and camera(s): the *optical flow* method and secondly the *feature correspondence* approach.

The *feature correspondence* approach has the disadvantage that it requires some specific features in the image plane to be matched. The feature correspondence problem poses a big challenge in the computer vision world and only the partial solutions have been achieved [1]. On the other hand the *optical flow* method involves large amount of computation [8]. Another drawback of this technique is the fact that with one equation and two unknowns an extra constraint has to be imposed. Usually the smoothness constraint is utilized. This, however, may not be realistic in several cases leading to erroneous results.

The newly developed *direct method* [2][7][9][10] which does not require the computational complexity of intermediate steps (feature detection, correspondence,

and optical flow computation), therefore is desirable.

In the method used in this thesis, the concept of *unified optical flow field* (UOFF) has been exploited. The UOFF is an extension of the fundamental optical flow formulations by Horn and Schunck [8]. Two main aspects of the the UOFF are discussed in the [13]. First of them, is that the *brightness function* of an image, which is not only a function of time but also a function of the sensor's spatial position. The concept of imaging space is presented as an accurate description of the set of all possible brightness functions. Secondly, the *brightness invariance* is recognized not only for the time variation but also for the (sensor's) space variation so that the brightness invariance equations for both time and space domains are established.

The method used in this thesis is based on a new method [14] which is a direct method, based on the UOFF. The UOFF is discussed in more detail in the second chapter. The third chapter discusses the *direct method.* The fourth chapter includes the experimental work on various structures. Several interesting results are also presented there. Finally, the conclusion of the whole work is drawn in the fifth chapter.

Chapter 2

BACKGROUND

This chapter introduces the concepts which provide the framework for the work done in this thesis. Also the mathematical results are presented to be used in the experiments. As already mentioned in the previous chapter, the direct method approach does not include the computation of the optical flow, rather it uses the spatiotemporal derivatives of the image intensity function to estimate the surface.

2.1 Unified Optical Flow Field

Optical flow may be defined as the apparent velocities of movement of brightness patterns in an image, which in turn, is due to the relative motion of the *viewer* and the *object* [8]. However, the motion under consideration is in the time domain. As discussed in the previous chapter, UOFF is the extension of *the temporal optical flow* to spatial domain. All the mathematical formulations, which are parallel to temporal optical flow, are also defined [13].

In 3-D world space, a sensor as a solid article can be translated (which has three degrees of freedom) and rotated (which has two free dimensions). It is noted

that here the rotation of a sensor about its *optical axis* is not counted since the images thus generated will remain unchanged. So we can obtain a variety of images when a sensor is translated to different coordinates and rotated to different angles in 3-D world space. Equivalently, we can imagine that there are infinitely many sensors in 3-D world space which occupy all of possible spatial coordinates and assume all of possible orientations at each coordinate, i.e., they are located on all of possible positions. At one specific moment all of these images form a set of images, called *a spatial sequence of images.* When time varies these sets of images form a much bigger set of images. Clearly, it is impossible to describe such a set of images by using the $g(x, y, t)$ discussed in [13]. Instead, it should be described by a more general brightness function $g(x, y, t, \vec{s})$, where $\vec{s}$ indicates the sensor's position in 3-D world space, i.e. the coordinates of the sensor center and the orientation of the optical axis of the sensor . As mentioned previously $\vec{s}$ is a 5-D vector [13]. That is

$$\vec{s} = (\tilde{x}, \tilde{y}, \tilde{z}, \beta, \gamma) \tag{2.1}$$

where $\tilde{x}$, $\tilde{y}$ and $\tilde{z}$ represent the coordinates of the optical center of the sensor in 3-D world space; β and γ represent the orientation of the optical axis of the sensor in 3-D world space.

In dealing with a *"spatial" sequence of images*, consider the various positions of cameras in space at a specific moment. One way to describe the camera movement in space is fixing the left camera and moving the right camera (see Figure 2.1). The movement of the right camera can be viewed as: the translation of the lens center O^R followed by the rotation of the optical axis $O^R Z^R$. The two optical axes OZ and $O^R Z^R$ are assumed, for simplicity, to be coplanar.

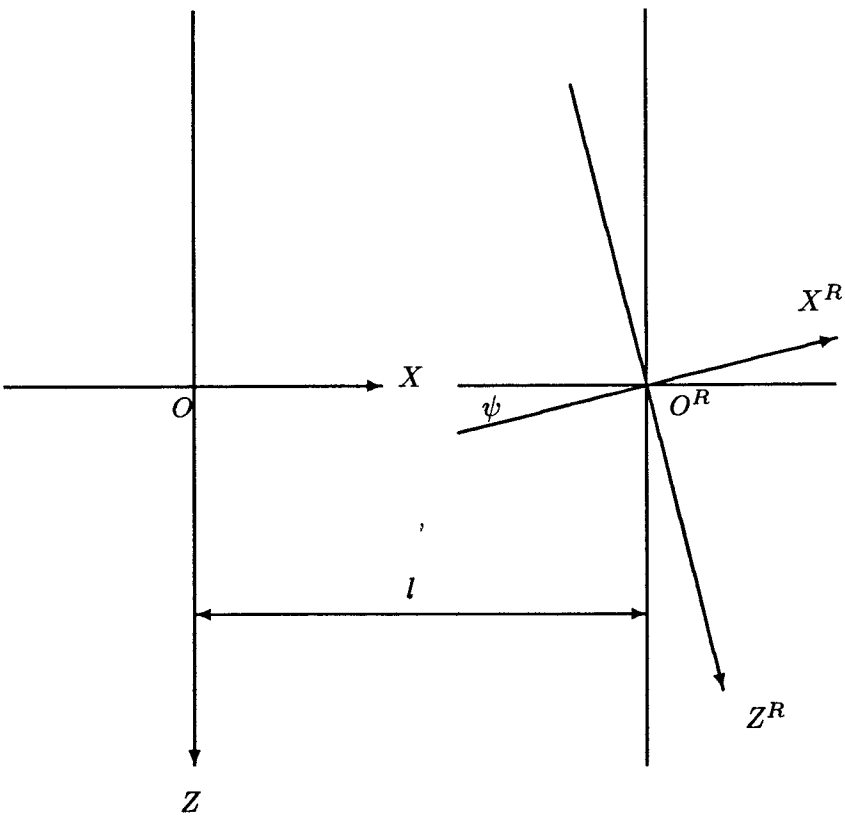


Figure 2.1: Imaging geometry

The lens center O^R can therefore only be translated on the OXZ plane. Hence any translation of the O^R on the OXZ plane can be decomposed as the translation along the direction parallel to the OX axis and the translation along the direction parallel to the OZ axis. The rotation of the optical axis O^RZ^R is about the O^RY^R axis. The displacements of the optical center O^R along the OX and OZ directions are denoted by $\tilde{x}$ and $\tilde{z}$, respectively. The angle displacement of O^RZ^R about O^RY^R is marked by $\tilde{\theta}$. However, the assumption made previously that the O^R lies on the OX implies $\tilde{z} = 0$. Therefore $\tilde{z}$ will not be considered under the assumption made. Define

$$\delta s \triangleq \sqrt{\tilde{x}^2 + \chi^2\tilde{\theta}^2}$$

where χ is a *characteristic length* chosen according to imaging setting. So, δs is a measure of the variation of the right camera position with respect to the left camera position, i.e, the variation of the position of the right lens center O^R with respect to that of the left lens center O and the orientation of the right optical axis O^RZ^R with respect to that of the left optical axis OZ. Let s denote the camera position in space and its superscript denote which camera is considered. For instance, s^L is used to denote the left camera position, s^R the right camera position, and we have $s^R = s^L + \delta s$.

So, when $\tilde{x} = 0$ and $\tilde{\theta} = 0$ (hence $\delta s = 0$), the two cameras are at the same position in space, i.e., $s^L = s^R$. If the camera's moving path is specified on the $\tilde{x}$-$\tilde{\theta}$ plane, different values of $\tilde{x}$ and $\tilde{\theta}$ (hence different values of δs) determine the various values of s^R, i.e, the various positions of the right camera in space.

At a specific moment t_1, if the optical radiation of a world point P is isotropical

we then get:

$$g(x_p(t_1,\vec{s}_1),y_p(t_1,\vec{s}_1),t_1,\vec{s}_1) = g(x_p(t_1,\vec{s}_2),y_p(t_1,\vec{s}_2),t_1,\vec{s}_2) \tag{2.2}$$

where x_P, y_P is the coordinate for a world point P. This is *the brightness space-invariance equation.* Applying the similar derivation to that used in [2] for determining the temporal optical flow, the following equation for the spatial optical flow is presented.

$$\frac{\partial g^L}{\partial x}u^s + \frac{\partial g^L}{\partial y}v^s + \frac{\partial g^L}{\partial s} = 0 \tag{2.3}$$

Let us take a close look at each quantity in the above equation. In equation(2.3) the quantities with the superscript L are related to the left sensor. The $\frac{\partial g^L}{\partial s}$ can be estimated from image data as follows:

$$\frac{\partial g^L}{\partial s} \approx \frac{g^R(x^L,y^L,t) - g^L(x^L,y^L,t)}{\delta s} \tag{2.4}$$

This is similar to the estimation of $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial y}$ in [8]. The u^s and v^s are defined as follows. Let

$$\delta x \triangleq x^R - x^L \qquad \delta y \triangleq y^R - y^L \tag{2.5}$$

where (x^R, y^R) and (x^L, y^L) are projections of a same world point on the right and left images, respectively. Therefore, δx and δy are, respectively, the horizontal and vertical coordinate differences of the image points, corresponding to the same world point in 3-D space, on the right and left image planes.

$$u^s \triangleq \lim_{\delta s \to 0} \frac{\delta x}{\delta s} \tag{2.6}$$

$$v^s \triangleq \lim_{\delta s \to 0} \frac{\delta y}{\delta s} \tag{2.7}$$

Hence, the afore-defined u^s and v^s are, respectively, the spatial variation rates of δx and δy with respect to δs. These two quantities generated from the spatial sequence of images can be viewed as the counterpart of u^L and v^L (or u^R and v^R) generated from the temporal sequence of images.

It is seen that Equation (2.3) derived from the spatial sequence of images [13] is very similar, in format, to one derived from the *temporal sequence* of images by Horn and Schunck [8]. It can be seen that this equation will serve as the start point of our approach.

Chapter 3

DIRECT STEREO

As mentioned previously, the objective of the work done in this thesis is *to recover surface structure that can be characterized by an Nth degree polynomial equation from a given pair of stereo images.* This section presents the new approach, which has been used for the structure recovery.

3.1 Surface Structure - Nth Degree Polynomial Equation

In this thesis, we consider a surface that can be characterized by an Nth degree polynomial equation. That is

$$\sum_{j=0}^{K-1} \lambda(j) X^{\alpha(j)} Y^{\beta(j)} Z^{\gamma(j)} = 0 \tag{3.1}$$

where $0 \leq \alpha(j) + \beta(j) + \gamma(j) \leq N$; K is the number of coefficients that are not identically vanishing in the Nth degree polynomial; $j = 0, 1, \cdots, K-1$ is an arbitrary but fixed index sequence by which all K coefficients are arranged in Eq. (3.1). Obviously, there are $K-1$ independent coefficients among the total of K coefficients. Without loss of generality, we can choose a Cartesian coordinate system in 3-D space so that Eq. (3.1) can be rewritten as

$$\sum_{j=1}^{K-1} \left(\lambda(j) X^{\alpha(j)} Y^{\beta(j)} Z^{\gamma(j)}\right) + \lambda(0) = 0 \tag{3.2}$$

3.2 Brightness Invariance Equation

In Section 3.1, the concept of *imaging space* is introduced. At a specific moment, the various sensor's positions in 3-D space are considered. We assume that the left sensor is located at the origin of a Cartesian coordinate system in 3-D space (see figure 2.1). As discussed in the previous chapter, for a specific moment all of the images taken by various sensors in 3-D space form a *spatial sequence of images*. Hence, though an object does not move in 3-D space, it looks as if it would have experienced the certain movement from the various sensors' view. These *pseudo-movements* can be treated in a manner similar the treatment of the relative motion between the sensor and the rigid environment provided in Eq. (2.3). That is, we define

$$\vec{V}_s = (\frac{dX}{ds}, \frac{dY}{ds}, \frac{dZ}{ds})^T \tag{3.3}$$

where the superscript T represents transposition of the concerned vector, the subscript s indicates the s-domain. The various position of sensors in 3-D space with respect to the origin can be considered as sensors after experiencing various movement with respect to the origin. This type of movement is characterized by

$$\vec{T}_s = (U_s, V_s, W_s)^T \tag{3.4}$$

$$\vec{\omega}_s = (A_s, B_s, C_s)^T \tag{3.5}$$

where $\vec{T}_s$ is the translational component and $\vec{\omega}_s$ the rotational component, the

superscript T and the subscript s are the same as defined above. We then have

$$\vec{V}_s = -\vec{T}_s - \vec{\omega}_s \times \vec{r}_s \tag{3.6}$$

or in component form:

$$\begin{aligned}
\frac{dX}{ds} &= -U_s - B_s Z + C_s Y && (3.7)\\
\frac{dY}{ds} &= -V_s - C_s X + A_s Z && (3.8)\\
\frac{dZ}{ds} &= -W_s - A_s Y + B_s X && (3.9)
\end{aligned}$$

In Figure 2.1 the coordinates in image plane are denoted by x and y.

When the *focal length* of a lens equals 1, Eq. (3.11-3.12) can be derived under consideration of *perspective projection.*

$$\begin{aligned}
x &= \frac{X}{Z} && (3.10)\\
y &= \frac{Y}{Z} && (3.11)
\end{aligned}$$

From the definitions of u^s and v^s, i.e., Eq. (2.6-2.7), and the relations in Eq. (3.8-3.12), one can derive the next two equations.

$$\begin{aligned}
u^s &= \frac{\frac{dX}{ds}}{Z} - \frac{X\frac{dX}{ds}}{Z^2}\\
&= (-\frac{U_s}{Z} - B_s + C_s y) - x(-\frac{W_s}{Z} - A_s y + B_s x) && (3.12)\\
v^s &= \frac{\frac{dY}{ds}}{Z} - \frac{Y\frac{dX}{ds}}{Z^2}\\
&= (-\frac{V_s}{Z} - C_s x + A_s) - y(-\frac{W_s}{Z} - A_s y + B_s x) && (3.13)
\end{aligned}$$

Let us rewrite the brightness invariance equation for a spatial sequence of images.

$$\frac{\partial g^L}{\partial x}u^s + \frac{\partial g^L}{\partial y}v^s + \frac{\partial g^L}{\partial s} = 0 \tag{3.14}$$

or,

$$g_x + g_y + g_s = 0 \tag{3.15}$$

where, we have the following definition.

$$g_x = \frac{\partial g^L}{\partial x} \tag{3.16}$$

$$g_y = \frac{\partial g^L}{\partial y} \tag{3.17}$$

$$g_s \approx \frac{g^R(x^L, y^L, t) - g^L(x^L, y^L, t)}{\delta s} \tag{3.18}$$

Substituting (3.13-3.14) into (3.15) one can obtain

$$\begin{aligned} \frac{1}{Z} &= \frac{(-A_s y + B_s x)(x g_x + y g_y) - g_x(-B_s + C_s y) - g_y(-C_s x + A_s) - g_s}{x W_s g_x + y W_s g_y - U_s g_x - V_s g_y} \\ &= Q(x, y, g_x, g_y, g_s, A_s, B_s, C_s, U_s, V_s, W_s) \end{aligned} \tag{3.19}$$

It is noted that $A_s, B_s, C_s, U_s, V_s, W_s$ can be determined once the relative position of the two sensors in stereo imagery is known. g_x, g_y, and g_s can be determined from the given image data [2]. In Eq. (3.19) x and y are coordinates on image plane. Instead of explicitly solving $\frac{1}{Z}$, we apply $Q = \frac{p}{q}$ to the performance function in a minimization formulated below.

3.3 A Least Squares Formulation

Using Eq. (3.10-3.11), one can derive the following equations from Eq. (3.2),

$$\sum_{j=1}^{K-1} \left[\lambda(j)(xZ)^{\alpha(j)}(yZ)^{\beta(j)}Z^{\gamma(j)}\right] + 1 = 0 \tag{3.20}$$

or

$$\sum_{j=1}^{K-1}\left[\lambda(j)x^{\alpha(j)}y^{\beta(j)}Z^{\alpha(j)+\beta(j)+\gamma(j)}\right]+1=0 \tag{3.21}$$

According to Eq, (3.19), the above equation can be converted to

$$\sum_{j=1}^{K-1}\left[\lambda(j)x^{\alpha(j)}y^{\beta(j)}Q^{-(\alpha(j)+\beta(j)+\gamma(j))}\right]+1=0 \tag{3.22}$$

Define a *performance function* as

$$J=\int\int_I\left\{\sum_{j=0}^{K-2}\left[\lambda(j)x^{\alpha(j)}y^{\beta(j)}Q^{-(\alpha(j)+\beta(j)+\gamma(j))}\right]+1\right\}^2 dxdy \tag{3.23}$$

where I is a region on the image plane associated with the concerned surface in 3-D space. The task here is to find a set of coefficients $\lambda(j)$ so that the performance function J is minimized.

It is well known that the following linear equations are necessary conditions for minimization of the J function.

$$\frac{\partial J}{\partial\lambda(i)}=0 \tag{3.24}$$

where $i=1,2,\ldots,K-1$. That is equivalent to

$$\sum_{j=1}^{K-1}\left[\int\int_I(x^{\alpha(j)+\alpha(i)-4}y^{\beta(j)+\beta(i)}Q^{-(\alpha(j)+\beta(j)+\gamma(j)+\alpha(i)+\beta(i)+\gamma(i)-4)})dxdy\right]\lambda(j) = -\int\int_I \mathrm{x}^{\alpha(\mathrm{i})-2}\mathrm{y}^{\beta(\mathrm{i})}\mathrm{Q}^{-(\alpha(\mathrm{i})+\beta(\mathrm{i})+\gamma(\mathrm{i})-2)}\mathrm{dxdy} \tag{3.25}$$

with $i=1,\ldots,K-1$.

In matrix-vector form, one has

$$\begin{bmatrix} M_{1,1} & \cdots & M_{1,K-1} \\ \vdots & \vdots & \vdots \\ M_{K-1,1} & \cdots & M_{K-1,K-1} \end{bmatrix}\begin{bmatrix} \lambda(1) \\ \vdots \\ \lambda(K-1) \end{bmatrix}=\begin{bmatrix} D_1 \\ \vdots \\ D_{K-1} \end{bmatrix} \tag{3.26}$$

$$M_{i,j} = \int\int_I (x^{\alpha(j)+\alpha(i)-4} y^{\beta(j)+\beta(i)} Q^{-(\alpha(i)+\beta(i)+\gamma(i)+\alpha(j)+\beta(j)+\gamma(j)-4)}) dxdy \quad (3.27)$$

$$D_i = -\int\int_I x^{\alpha(i)-2} y^{\beta(i)} Q^{-(\alpha(i)+\beta(i)+\gamma(j))} dxdy \quad (3.28)$$

$i, j = 1, 2, \ldots, K-1$.

In this *set of linear equations*, all of coefficients of the Nth degree polynomial, i.e., $\lambda(1), \cdots, \lambda(K-1)$ are unknown. The all of entries in the matrix, i.e., $M_{i,j}$ and the all of entries in the vector, i.e., D_i can be computed from the given stereo image data and the known imaging setting. The unknown coefficients $\lambda(1), \cdots, \lambda(K-1)$ can thus be solved. In other words, one can recover the surface structure: both the shape of the surface and the position of the surface in 3-D space because the polynomial equation characterizing the surface has been completely determined.

Chapter 4

SIMULATION

This chapter describes *the experimental work* done to solve the problem of 3 -D surface recovery. The work is primarily based on the *theoretical results* presented in the previous chapters. The tests were performed on various surfaces and favorable results were achieved showing the generality of the proposed algorithm. The surface structures used for the experimentation include *a cone, a parabola, a planar surface* and *the combination of the cone and planar surfaces* (i.e., a surface characterized by a 3^{rd} degree *factorable polynomial* equation). However, only the cone, the plane and the combination of the plane and the cone will be discussed in this chapter.

4.1 Cone

The cone used for the simulation is characterized by the following mathematical representation (see Fig. 4.1 and Fig. 4.2),

$$K^2X^2 - Y^2 + K^2(Z - D)^2 = 0 \tag{4.1}$$

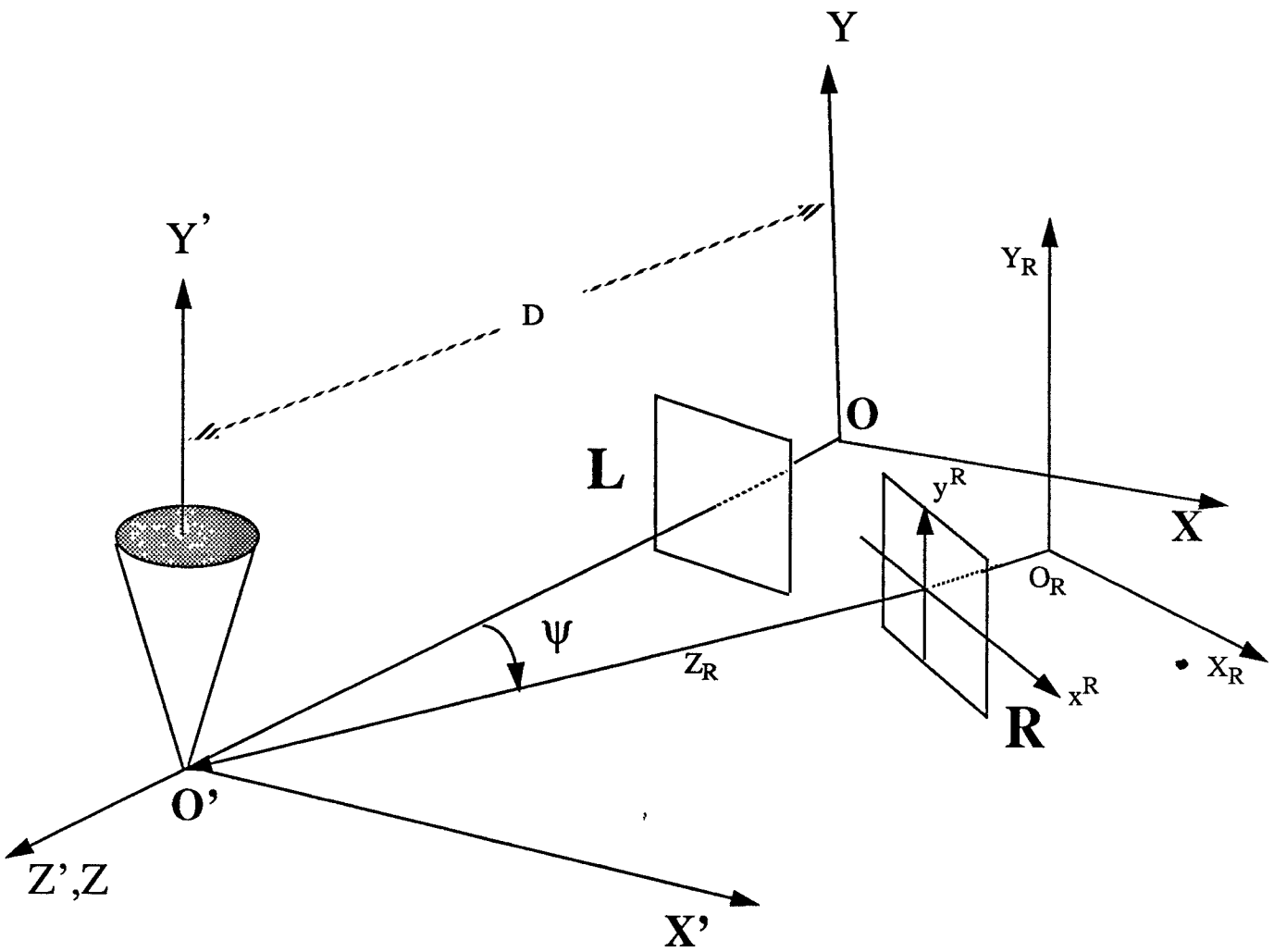


Fig. 4.1 Stereo image setting

where K is defined as:

$$K = tan(\frac{\pi}{2} - \phi) \tag{4.2}$$

and ϕ is the angle as shown in the Fig. 4.2.

The typical value of ϕ used in the simulation is 15°. Therefore, the equation of the cone may be rewritten as:

$$13.92820X^2 - Y^2 + 13.92820(Z - D)^2 = 0$$

or

$$X^2 - 0.0717967Y^2 + (Z - D)^2 = 0 \tag{4.3}$$

Since, the cone represented by the equation given above, is of infinite size, we select an arbitrary height of the cone for simulation purposes. The typical values of the height of the cone, used in the simulation are 2.0 and 5.0.

4.1.1 Imaging geometry

The center of the cone lies on OZ axis and its base is located at $(0, 0, D)$ with respect to the reference Cartesian coordinate system as shown in the Fig. 4.1. The distance between the center of the lens and the center of the cone is denoted by D. However, the distance has to be large enough to satisfy the far field assumptions, which is generally the case in practice. The optical axis of the left lens is aligned along the OZ axis. The Cartesian coordinate system of the right camera is achieved by rotating the left camera by an angle ψ in the clockwise

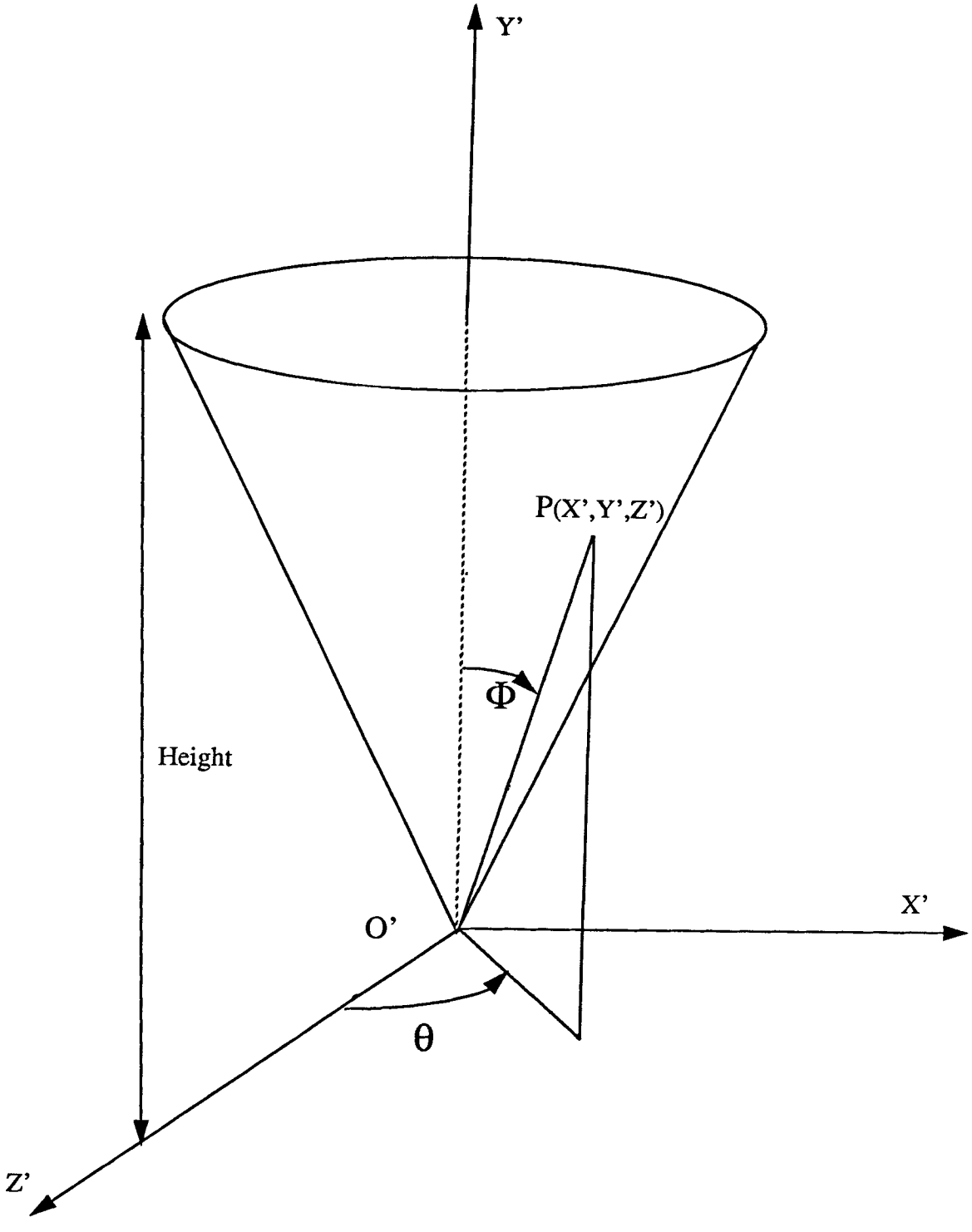


Fig. 4.2. Cone used in the simulation

direction about $O'Y'$ axis (Fig. 4.2), when we are looking towards the origin from positive Y'-*axis* direction.

4.1.2 Simulation data generation

Rewriting the equation of the cone under consideration,

$$K^2X^2 - Y^2 + K^2(Z - D)^2 = 0 \tag{4.4}$$

It may be noted that the gray level assignment is done in the *spherical coordinate system*, which has been found more suitable for this purpose. We can write the coordinate system conversion formula as folows:

$$X = \rho sin(\theta)sin(\phi) \tag{4.5}$$

$$Y = \rho cos(\phi) \tag{4.6}$$

$$Z = D + \rho sin(\theta)cos(\phi) \tag{4.7}$$

We divide the image plane into 128 $\times$ 128 pixels. We see that for each point on the image plane there is a corresponding point in the 3-D space. However, only a subset of those points correspond to the points on the cone. Rest of the points constitute the *background*, which is assigned an arbitrary but fixed gray level.

We apply the concepts of *perspective projection* to find the relation between the points on the image plane and the points in the 3-D space.

We know that a specific point on the image plane and the coordinates of its corresponding point in 3-D space, satisfy Eqs. (3.10-3.11) which are rewritten below.

$$x = \frac{X}{Z} \tag{4.8}$$

$$y = \frac{Y}{Z} \tag{4.9}$$

where the value of f (the focal length of the camera) has been assumed to be equal to 1, for simplicity in computation. It, however, maintains the generality of the problem under consideration.

Substitution of X and Y from Eqs. $(4.8 - 4.9)$ into the equation of the cone, Eq. (4.1), yields one equation with one unknown. Hence, the value of Z can be determined, which is given by,

$$Z = \frac{DK(K - \sqrt{y^2 - K^2x^2})}{x^2K^2 - y^2 + K^2} \tag{4.10}$$

where only the real solution is allowed due to the obvious physical meaning, i.e.,

$$y^2 - K^2x^2 > 0 \tag{4.11}$$

Also, while solving the quadratic equation, we select the solution with the negative sign preceding the square root, because it gives the correct value of Z, i.e., smaller of the two values of Z is selected. Thus, we actually choose the surface of the cone which is facing the camera. Using Eqs. $(4.8 - 4.9)$, the values of X and Y may be found out.

Similarly, the value of θ may be determined by the following formula:

$$\pi - \theta = arcsin(\frac{X}{\rho sin(\phi)}) \tag{4.12}$$

where θ is the angle between the projection of the point on the cone onto the $X'O'Z'$ plane and the positive $O'Z'$ axis (Fig. 4.2). The angle ϕ is constant here.

The brightness function used is as follows:

$$g^L(x^L, y^L) = K_1 sin(K_Y Y) cos(K_\theta \theta) \tag{4.13}$$

where g^L is the gray level assigned to the point (x^L, y^L) with the superscript L indicating the left image plane, and θ is the same as defined above. The values of the parameters (K_1 and K_2) used are:

$$K_1 = 1000$$
$$K_Y = K_\theta' = 8$$

Also, the point (x^L, y^L) is the perspective projection of (X, Y, Z) in 3-D space onto the left image plane, obtained by the perspective projection of the point (X, Y, Z) in the 3-D space. The resulting image of the cone is included in Appendix D.

So far, we have discussed the generation of the image, which has arbitrarily been chosen as left image by the choice of the coordinated system. To simulate another image (right), we use the following procedure. Since, the surface structure of the cone is *symmetrical* with respect to Y'-*axis*, which is also the

axis of rotation of the optical axis of the right camera, the geometrical shape of the cone as viewed from the right camera remains same. This is also verified by the use of the rotation transformation matrix. However, care has to be taken for the change in the brightness pattern. Instead of moving the camera to the new position, we can rotate the object (cone) by the same angle, ψ, but in the opposite direction. This can be achieved by adding an angle $-\psi$ to the angle θ, where θ and ϕ are defined as shown in Fig. 4.2. This maintains the *brightness invariance constraint.* The gray level function for the right camera in terms of the coordinates of the already defined Cartesian coordinate system is,

$$g^R(x^R, y^R) = K_1 sin(K_Y Y) cos(K_\theta(\theta - \psi)) \tag{4.14}$$

A more generalized way of handling this issue is considered in the case of the non-symmetrical surface structure, to be discussed in the next section.

Once the gray level assignment is done, the values of the spatial derivatives, i.e., g_x and g_y are calculated using the formula given in Appendix A. The value of g_s is calculated, using Eq. (3.18).

4.1.3 Surface estimation

Since any Nth order surface structure may be represented by a general equation as follows:

$$\sum_{j=0}^{K-1} \lambda(j) X'^{\alpha(j)} Y'^{\beta(j)} Z'^{\gamma(j)} = 0 \tag{4.15}$$

where $0 \leq \alpha(j) + \beta(j) + \gamma(j) \leq N$; K is the number of coefficients that are not identically vanishing in the Nth degree polynomial; $j = 0, 1, \cdots, K-1$ is an

arbitrary but fixed index sequence by which all K coefficients are arranged.

Obviously, there are $K-1$ independent coefficients among the total of K coefficients.

For the case of second order polynomial, the above equation can be, possibly, expanded as follows:

$$\lambda'(0)X'^2 + \lambda'(1)Y'^2 + \lambda'(2)Z'^2 + \lambda'(3)X'Y' + \lambda'(4)X'Z' + \lambda'(5)Y'Z' + \lambda'(6)X' + \lambda'(7)Y' + \lambda'(8)Z' + \lambda'(9) = 0 \quad (4.16)$$

It is noted that for the computational purposes, we have shifted the coordinate system as see in Fig. 4.1. i.e., $Z' = Z - \overline{Z}$ is used. The main reason for doing so is to avoid the computational error arising due to the numerical differences between the values of Z and other variables. This issue is discussed in greater detail, later in this section.

Eq. (4.16) may be written as follows by using Eqs. $(4.8 - 4.9)$.

$$\lambda'(0)x^2Z^2 + \lambda'(1)y^2Z^2 + \lambda'(2)(Z-\overline{Z})^2 + \lambda'(3)xyZ^2 + \lambda'(4)xZ(Z-\overline{Z}) + \lambda'(5)yZ(Z-\overline{Z}) + \lambda'(6)xZ + \lambda'(7)yZ + \lambda'(8)(Z-\overline{Z}) + \lambda'(9) = 0 \quad (4.17)$$

It is, generally, simpler to normalize with respect to the constant term, but in our case, the constant term is missing from the equation of the cone, Eq. (4.1), and hence some other term has to be chosen. Here we choose the term $\lambda'(0)X'^2$ for normalization, thus converting Eq. (4.17) to the following equation.

$$1 + \lambda(1)y^2x^{-2} + \lambda(2)x^{-2}Z^{-2}(Z-\overline{Z})^{-2} + \lambda(3)x^{-1}y$$
$$+\lambda(4)x^{-1}Z^{-1}(Z-\overline{Z}) + \lambda(5)yx^{-2}Z^{-1}(Z-\overline{Z}) + \lambda(6)x^{-1}Z^{-1}$$
$$+\lambda(7)x^{-2}yZ^{-1} + \lambda(8)x^{-2}Z^{-2}(Z-\overline{Z}) + \lambda(9)x^{-2}Z^{-2} = 0 \qquad (4.18)$$

Replacing $\frac{1}{Z}$ by Q according to Eq. (3.19), we get,

$$1 + \lambda(1)y^2x^{-2} + \lambda(2)x^{-2}Q^2(Z-\overline{Z})^{-2} + \lambda(3)x^{-1}y + \lambda(4)x^{-1}Q(Z-\overline{Z})$$
$$+\lambda(5)yx^{-2}Q(Z-\overline{Z}) + \lambda(6)x^{-1}Q + \lambda(7)x^{-2}yQ + \lambda(8)x^{-2}Q^2(Z-\overline{Z}) +$$
$$\lambda(9)x^{-2}Q^2 = 0 (4.19)$$

Hence, *the performance function* J, discussed in Section 3.3, in this experiment becomes,

$$J = \int\int_I \left[\sum_{j=1}^{9} \lambda(j)G(j) + G(0)\right]^2 dxdy \qquad (4.20)$$

where

$$\begin{aligned}
G(0) &= 1 \\
G(1) &= y^2x^{-2} \\
G(2) &= x^{-2}Q^2(Z-\overline{Z})^{-2} \\
G(3) &= x^{-1}y \\
G(4) &= x^{-1}Q(Z-\overline{Z}) \\
G(5) &= x^{-2}yQ(Z-\overline{Z}) \\
G(6) &= x^{-1}Q \\
G(7) &= x^{-2}yQ \\
G(8) &= x^{-2}Q^2(Z-\overline{Z}) \\
G(9) &= x^{-2}Q^2
\end{aligned} \qquad (4.21)$$

A set of necessary conditions for the minimization becomes

$$\frac{\partial J}{\partial \lambda(i)} = \int\int_I 2\left[\sum_{j=1}^{9}\lambda(j)G(j) + G(0)\right]G(i)dxdy = 0 \qquad (4.22)$$

This is equivalent to

$$\sum_{j=1}^{9}\left(\int\int_I G(i)G(j)dxdy\right)\lambda(j) = -\int\int_I G(0)G(i)dxdy \qquad (4.23)$$

with $i = 1, 2, \ldots, 9$.

In matrix format, these necessary conditions become

$$\int\int_I \begin{bmatrix} G(1)G(1) & \ldots & G(1)G(9) \\ \vdots & \vdots & \vdots \\ G(9)G(1) & \ldots & G(9)G(9) \end{bmatrix} dxdy \begin{bmatrix} \lambda(1) \\ \vdots \\ \lambda(9) \end{bmatrix}$$

$$= \int\int_I \begin{bmatrix} -G(0)G(1) \\ \vdots \\ -G(0)G(9) \end{bmatrix} dxdy \qquad (4.24)$$

In implementing the above computation, one thing needs to be emphasized. That is, the *depth* Z is very large compared with the other quantities involved, causing numerical and computational problem. In order to deal with this problem, $Z = \overline{Z} + \tilde{Z}$ is used where $\overline{Z}$ can be viewed as sort of "steady" or average component while $\tilde{Z}$ "fluctuation" component component in Z. The average value of Z might be used as $\overline{Z}$. In other words, we can percieve of another coordinate system ($O" - X"Y"Z"$), originated at $(0, 0, \overline{Z})$ and with the axes $O"X"$, $O"Y"$, $O"Z"$ parallel to the axes OX, OY, OZ, respectively. In the simulation $\overline{Z}$ is taken as the calculated average value of Z, from the experiment, which equals 99.720230 ($\phi = 15.0°$). By doing this, the values of $X", Y"$ and $Z"$, thus obtained are numerically comparable. The estimated coefficients $\lambda(j), j = 1, \cdots, 8$, returned from the program are as follows.

$\lambda(1)$: 0.078168

$\lambda(2)$: -0.071835

$\lambda(3)$: 0.999335

$\lambda(4)$: -0.000282

$\lambda(5)$: 0.000603

$\lambda(6)$: -0.000325

$\lambda(7)$: 0.000127

$\lambda(8)$: 0.000128

$\lambda(9)$: -0.559010

It is noted that these returned coefficinets are with respect to O"-X"Y"Z" system. The parameter ψ in the experiment is 0.05° and the $\overline{Z} = 99.720230$. Noticing that $Z = \overline{Z} + \tilde{Z}$, one obtains the recovered surface equation approximately as follows.

$$1.000000X^2 - 0.071835Y^2 + 0.999335(Z - 100)^2 = 0 \qquad (4.25)$$

The expected coefficients and the recovered coefficients of the cone are listed in Table 4.1 (see Section 4.4).

Compared with the assumed cone described in the beginning of this section, i.e., a cone of height 5 and the angle with the $Y - axis$ equal to 15° and the distance between its center and the origin of the Cartesian system being D (already defined as 100), the percentage of error is less than 0.1, thus, it is obvious that satisfactory results have been achieved.

4.2 Plane

The plane used in the structure can be described as:

$$AX + BY + CZ + D = 0 \tag{4.26}$$

where A, B, C and D are the coefficients in the equation.

The *imaging geometry*, used for the simulation of the plane surface is the same as described in the previous section (Fig. 4.1).

4.2.1 Simulation data generation

In Eq. (4.26):

$$AX + BY + CZ + D = 0 \tag{4.27}$$

where, the values of the coefficients are as follows:

Substituting the perspective projection Eqs. (4.8-4.9) into the equation of the plane i.e., Eq. (4.26), we can solve for Z,

$$Z = \frac{-D}{Ax + By + C} \tag{4.28}$$

where, x and y represent the coordinates on the image plane.

After determining Z, we now determine the values of X and Y.

The brightness function used for the planar surface is as follows:

$$g^L(x^L, y^L) = MAG \times sin(k_1 X) sin(k_2 Y) sin(k_3 Z) \tag{4.29}$$

where the values of the parameters used, are as follows:

$$\text{MAG}=1000$$

$$k_1 = k_2 = k_3 = 8$$

As described in the case of cone, the point (x^L, y^L) is obtained by the perspective projection of a point (X, Y, Z) in 3-D space onto the left image plane.

The data generated, so far, applies to the left image. To generate the image as viewed by the right camera, we first, translate the camera and then rotate it clockwise, when viewed from the positive Y'-axis towards the origin by an angle ψ. The same effect may be generated by keeping the camera position as that of the left camera and rotating the object (cone) by negative ψ, which is found more convenient for simulation purposes. This is the same as discussed in the previous section.

The coefficients of the surface of the plane as viewed by the right camera. By the rotation and the translation transformation, we may write,

$$X = cos(\psi) \times X_R - sin(\psi) \times Z_R + D\psi \tag{4.30}$$

$$Y = Y_R \tag{4.31}$$

$$Z = sin(\psi) \times X_R + cos(\psi) \times Z_R \tag{4.32}$$

Substituting the values of X, Y and Z from the above equations into Eq. (4.26), we have

$$A_R = Acos(\phi) + Csin(\phi) \tag{4.33}$$

$$B_R = B \tag{4.34}$$

$$C_R = -Asin(\phi) + Ccos(\phi) \tag{4.35}$$

$$D_R = D + AD\psi \tag{4.36}$$

where, the A, B, C and D are the coefficients of the plane described before, i.e., the plane corresponding to the 3-D Cartesian coordinate system associated with the left camera image; A_R, B_R, C_R and D_R are the corresponding coefficients of the plane as seen by the right camera. The same set of Eqs. (4.8-4.9, 4.28) may

be used for the right image. However, to satisfy the brightness invariance constraint, one condition has to be established, i.e., for any pixel on the right image plane, there is a corresponding point in the 3-D space, whose brightness value must be same as the point, when viewed by the left camera (Section 3.2). For this we use the transformations given by Eqs. (4.30-4.32). Thus the brightness invariance condition is satisfied.

The *reconstruction* of the plane is discussed in detail, in the section on composite surface. A table is also included in the last section to analyze the performance for the reconstruction of the plane.

4.3 Composite Surface

This section discusses the simulation and reconstruction of a curved surface that is characterized by a 3^{rd} degree polynomial equation, in which the polynomial can be factored into a 2^{nd} degree polynomial (cone) and a 1^{st} degree polynomial (plane). The procedure to implement is essentially the same as discussed in the previous section. Complexity is, however, increased due to obvious reasons. Special care is taken in the decision of assignment of any image point to the corresponding surface in the 3-D space. This section also focuses on the criterion used to discriminate the surfaces.

4.3.1 Composite surface data generation

The composite surface simulated is obtained by superimposing the plane Eq. (4.26) over the cone Eq. (4.1). It is characterized by,

$$(K^2X^2 - Y^2 + K^2(Z - 100)^2)(AX + BY + CZ + D) = 0 \qquad (4.37)$$

Refer to the pair of images in Appendix 6.4 for the composite surface structure under consideration.

One of the major issues in the simulation of the composite structure has been the establishment of the valid criterion to discriminate the points corresponding to the respective surfaces. The criterion used for this purpose is as follows. A particular point on the image plane is the perspective projection of some points on the cone surface if

$$AX_{cone} + BY_{cone}CZ_{cone} + D < 0 \tag{4.38}$$

where A, B, C and D are the coefficients of the planar surface under consideration, X_{cone}, Y_{cone} and Z_{cone} are the coordinates of the corresponding point on the cone surface in the $3D$ space.

If this test fails, we test if this image point is the projection of some points on the planar surface. The criterion used for this purpose is expressed in the following inequality.

$$K^2X_{plane}^2 - Y_{plane}^2 + K^2Z_{plane}^2 < 0 \tag{4.39}$$

where X_{plane}, Y_{plane} and Z_{plane} are the coordinates of the corresponding point on the planar surface and K is the same as defined by the Eqs. (4.2). If this condition is satisfied, the point corresponds to the planar surface. If both the criteria fail, it corresponds to the background, and the predefined gray level for the background is assigned to this point on the image plane. This issue is explained with the help of a flow chart in Appendix B.

Once the decision regarding the surface is made, the next step is to assign the

gray level to the image plane point. The gray level functions used here are the same as mentioned previously by Eqs. (4.13) and (4.29), respectively. This completes the formation of the left image. The right image formation involves the rotation of the structure by $-\psi$ to simulate the rotation of the camera position, as discussed in the previous sections. Also, the same procedure is followed to compute derivatives g_x, g_y and g_s, as already described. However, the set of gray functions used has to be in correspondence with the surface identified.

4.3.2 Reconstruction of the composite structure

The method used to recover the surface parameters of the composite structure is essentially the same as discussed in the previous sections. The set of equations used for the recovery is however depicted in the following.

Let us rewrite the Eq. (3.1) to represent an Nth order surface structure as follows:

$$\sum_{j=0}^{K-1} \lambda'(j) X'^{\alpha(j)} Y'^{\beta(j)} Z'^{\gamma(j)} = 0 \tag{4.40}$$

where $0 \leq \alpha(j) + \beta(j) + \gamma(j) \leq N$; K is the number of coefficients that are not identically vanishing in the Nth degree polynomial; $j = 0, 1, \cdots, K-1$ is an arbitrary but fixed index sequence by which all K coefficients are arranged in. Obviously, there are $K-1$ independent coefficients among the total of K coefficients.

For the case of a third degree polynomial, the above equation can be written as follows:

$$\lambda(0)X^3 + \lambda(1)Y^3 + \lambda(2)Z^3 + \cdots +$$

$$\lambda(9)YZ^2 + \lambda(10)X^2 + \lambda(11)Y^2 \cdots +$$

$$\lambda(15)YZ + \lambda(16)X + \lambda(17)Y + \lambda(18)Z + \lambda(19) = 0 \tag{4.41}$$

The *correspondence* between the monomials and the coefficients may be seen in Tables 4.6 and 4.7. The reason for rearranging the equation arises due to the consideration that the normalization is conducted with respect to $\lambda(10)$, i.e. the coefficient of X^2 instead of the constant term for the same reason as discussed previously.

The index sequence in which all of coefficients are arranged is not unique. Replacing $\frac{1}{Z}$ by Q according to eq. (3.16) and normalizing with respect to the coefficient of X^2 term, one then has the same expression as we had in the previous cases.

Hence, the performance function J in this experiment becomes

$$J = \int\int_I \left[\sum_{j=0, j\neq 10}^{19} \lambda(j)G(j) + G(10) \right]^2 dxdy \tag{4.42}$$

A set of necessary conditions for the minimization becomes

$$\frac{\partial J}{\partial \lambda(i)} = \int\int_I 2 \left[\sum_{j=0, j\neq 10}^{19} \lambda(j)G(j) + G(10) \right] G(i)dxdy = 0 \tag{4.43}$$

This is equivalent to

$$\sum_{j=0, j\neq 10}^{19} \left(\int\int_I G(i)G(j)dxdy \right) \lambda(j) = -\int\int_I G(10)G(i)dxdy \tag{4.44}$$

with $j = 0, 1, \ldots, 19$.

In matrix format, these necessary conditions become

$$\int\int_I \begin{bmatrix} G(0)G(0)\cdots G(0)G(9) & G(0)G(11)\cdots G(0)G(19) \\ \vdots\cdots\vdots & \vdots\cdots\vdots \\ G(19)G(0)\cdots G(19)G(9) & G(19)G(11)\cdots G(19)G(19) \end{bmatrix} dxdy \begin{bmatrix} \lambda(0) \\ \vdots \\ \lambda(19) \end{bmatrix}$$

$$= \int\int_I \begin{bmatrix} -G(0)G(19) \\ \vdots \\ -G(19)G(19) \end{bmatrix} dxdy \tag{4.45}$$

Having, found the relation between the unknown coefficients and the known matrices, we get a linear relationship, which can be individually solved for the value of $\lambda(i)$.

4.4 Discussion and Results

This section discusses some aspects of the work done based on the results obtained from the simulation and experimentation.

Although, *the cone* and *the composite structure* have been discussed , the simulation was performed on other surfaces, too; e.g., Parabla.

The percentage of error shows the precision of the method to reconstruct the surface structures. However, the accuracy is expected to decrease for real images.

Coeff.	Terms	Calc. val.	Exp. Val.
$\lambda(0)$	const.	0.000000	-0.000328
$\lambda(1)$	X^2	-0.071796	-0.071835
$\lambda(2)$	Y^2	1.00000	0.999335
$\lambda(3)$	Z^2	0.00000	-0.000282
$\lambda(4)$	XY	0.000000	0.000603
$\lambda(5)$	XZ	0.000000	-0.000325
$\lambda(6)$	YZ	0.000000	0.000127
$\lambda(7)$	X	0.000000	0.000128
$\lambda(8)$	Y	0.000000	-0.00642
$\lambda(9)$	Z	0.000000	0.000000

Table 4.1: Cone ($\phi = 15.0°$,$\psi = 0.05°$)

The results are observed to improve with decrease in the angle ψ. Similarly, the results of the same degree of accuracy are obtained when the same procedure is repeated for $\phi = 30°$, as shown Table 4.5.

Similarly, for the plane, we have the following table.

Coeff.	Terms	Calc. val.	Exp. Val.
$\lambda(0)$	const.	0.000000	-0.0047
$\lambda(1)$	X^2	-0.071969	-0.071945
$\lambda(2)$	Y^2	1.00000	0.998412
$\lambda(3)$	Z^2	0.00000	0.000000
$\lambda(4)$	XY	0.000000	-0.000000
$\lambda(5)$	XZ	0.000000	-0.001932
$\lambda(6)$	YZ	0.000000	-0.000000
$\lambda(7)$	X	0.000000	-0.000642
$\lambda(8)$	Y	0.000000	-0.00642
$\lambda(9)$	Z	0.000000	0.00000

Table 4.2: Cone ($\phi = 15.0°$,$\psi = 0.1°$)

Similarly, we have the accuracy of the same order in the case of the composite surface as can be seen later in this section.

Coeff.	Terms	Calc. val.	Exp. Val.
$\lambda(0)$	X^2	1.00000	1.0000000
$\lambda(1)$	Y^2	-0.071797	0.0717969
$\lambda(2)$	Z^2	1.0000	1.000000
$\lambda(3)$	XY	0.00000	0.000000
$\lambda(4)$	XZ	0.00000	0.000005
$\lambda(5)$	YZ	0.00000	0.000000
$\lambda(6)$	X	0.00000	0.000000
$\lambda(7)$	Y	0.00000	0.000000
$\lambda(8)$	Z	0.00000	0.000000
$\lambda(9)$	Const.	0.000000	0.0003

Table 4.3: Cone ($\phi = 15.0°$, $\psi = 0.01°$)

The combination of the cone and plane, which are included in the table. Another set of values for the composite surface has been included for reference. The value of ψ is 0.01°.

Based upon the results tabulated so far, certain conclusions are drawn, which are discussed in the next section.

4.4.1 Parameter selection

A series of experiments were carried out with different sets of parameters to extract the optimum solution.

The parameters used in the gray level function for the planar surface as well as the cone are found to effect the results, with variation in their values. The reason why the integer values of the constants $K_1, K_2, K_3, K_\theta, K_Y$ are chosen, is the fact that the fractional values may cause discontinuity at the boundaries, which will result in abrupt changes at certain points. Moreover, the choice of the values which are too high will cause the gray level to change too rapidly,

Coeff.	Terms	Calc. val.	Exp. Val.
$\lambda(0)$	X^2	1.00000	1.00000
$\lambda(1)$	Y^2	-0.333333	-0.333333
$\lambda(2)$	Z^2	1.000000	1.000000
$\lambda(3)$	XY	0.00000	0.000000
$\lambda(4)$	XZ	0.00000	0.000001
$\lambda(5)$	YZ	0.00000	0.000007
$\lambda(6)$	X	0.00000	0.000000
$\lambda(7)$	Y	0.00000	0.000003
$\lambda(8)$	Z	0.00000	0.000000
$\lambda(9)$	Const.	0.000000	0.0003

Table 4.4: Cone ($\phi = 30.0°$,$\psi = 0.01°$)

Coeff.	Terms	Calc. val.	Exp. Val.
$\lambda(0)$	X^2	-0.23160310	-0.231576
$\lambda(1)$	Y^2	0.40016	0.399925
$\lambda(2)$	Z^2	0.163001	0.153793
$\lambda(3)$	Const.	1.000000	1.00000

Table 4.5: Plane (ψ =0.05°)

while low values will make the change in the gray level too slow.

The parameters used with the gray level function for the plane as well as with the cone, are found to give best results when they are selected to be equal to 8. However, this is not a very sensitive parameter.

Another parameter is the angle ψ, which is the angle between the coordinates of the left camera and the right camera. The values chosen for this parameter range from 0.001° to 0.5°. However, at larger angles, the experimental values seem to deviate from the calculated values.

Coeff.	Terms	Calc. val.	Exp. Val.
$\lambda(0)$	X^3	-0.2316031	-.236937
$\lambda(1)$	Y^3	0.133372	0.133672
$\lambda(2)$	Z^3	0.163001	0.153766
$\lambda(3)$	X^2Y	-0.400116	-0.400179
$\lambda(4)$	X^2Z	0.163001	0.149600
$\lambda(5)$	XY^2	0.077201	0.076848
$\lambda(6)$	ZY^2	-0.054339	-0.49362
$\lambda(7)$	XZ^2	-0.231603	-0.230135
$\lambda(8)$	YZ^2	-0.400	-0.40022
$\lambda(9)$	XYZ	0.00000	-0.008818
$\lambda(10)$	X^2	1.0000	1.00000
$\lambda(11)$	Y^2	-0.33333	-0.33345
$\lambda(12)$	Z^2	1.00000	0.997984
$\lambda(13)$	XY	0.00000	0.000323
$\lambda(14)$	XZ	0.00000	0.000433
$\lambda(15)$	YZ	0.00000	0.000443
$\lambda(16)$	X	0.00000	0.000117
$\lambda(17)$	Y	0.00000	0.000563
$\lambda(18)$	Z	0.00000	0.000971
$\lambda(19)$	Const.	0.000000	0.0003

Table 4.6: Composite Surface ($\phi = 15.0°$)

Coeff.	Terms	Calc. val.	Exp. Val.
$\lambda(0)$	X^3	-0.2316031	-.230490
$\lambda(1)$	Y^3	0.133372	0.133676
$\lambda(2)$	Z^3	0.163001	0.153698
$\lambda(3)$	X^2Y	-0.400116	-0.400193
$\lambda(4)$	X^2Z	0.163001	0.149470
$\lambda(5)$	XY^2	0.077201	0.076865
$\lambda(6)$	ZY^2	-0.054339	-0.049380
$\lambda(7)$	XZ^2	-0.231603	-0.230060
$\lambda(8)$	YZ^2	-0.400	-0.400196
$\lambda(9)$	XYZ	0.00000	-0.00196
$\lambda(10)$	X^2	1.0000	1.000000
$\lambda(11)$	Y^2	-0.33333	-0.334514
$\lambda(12)$	Z^2	1.00000	0.997949
$\lambda(13)$	XY	0.00000	0.000302
$\lambda(14)$	XZ	0.00000	0.000571
$\lambda(15)$	YZ	0.00000	0.004356
$\lambda(16)$	X	0.00000	0. 000110
$\lambda(17)$	Y	0.00000	0.000560
$\lambda(18)$	Z	0.00000	0.000947
$\lambda(19)$	Const.	0.000000	0.000072

Table 4.7: Composite Surface ($\phi = 30.0°$)

Chapter 5

CONCLUSION

The work done takes us another step towards proving the capabilities of the direct method for recovering the 3-D surface structures represented by polynomials of different order and characteristics . Two contributions of the thesis work are are summarized as follows:

(a) Infinite-size surfaces:
The proposed direct method may be applied to the recover the structures of any order and size as proved by the simulation results (Chapter 4), which is big progress over the previous work.

(b) Factorable surfaces:
Another very interesting result is the capability of the technique to the solution of the surface structures that are represented by *factorable polynomials.*

Besides, we also make the following observations.

- In $UOFF$, the brightness invariance has been extended to spatial domain.

- The method is extended to the solution of the problems involving high order polynomials. The proofs are already provided for the first, second and third order surfaces.

- The setting of the imaging system has great influence upon the final results. Generally, the setting should be such so as to be compatible with the far-field assumption.

As mentioned previously, the research work is based on the unified optical flow field (UOFF) and the direct method approach is used to extract the information from the stereo images. Least square formulation has been used for the minimization of the error.

Spatial domain analysis may be preferable in several cases over the time domain analysis due to several reasons. One of them being its processing in parallel. In addition to this, spatial domain methods may be used along with the time domain technique to improve the efficiency of any algorithm. This is due to the fact that more information is available.

The direct method approach can be used to recover the 3-D surface structure of higher order polynomials (continuous as well as factorable) with certain margin of error, which is ignorable, in general. It is observed that the percentage of error is, however, proportional to the number of factorable surfaces.

It is expected that the new approach can play a vital role in solving the problems of motion analysis. Favorable results have been obtained in this regard

too. Although, the experiments were performed on the simulated images, the success of the simulation results indicates that comparable degree of success may be achieved in case of real images.

It is also hoped that the proposed approach may be applied in real-time applications because of its less computational complexity and simplicity as compared to other prevalent methods. An analysis of the method for its application on real images is under investigation.

Chapter 6

BIBLIOGRAPHY

[1] J. K. Aggarwal and N. Nandhakumar, "*On the computation of motion from sequences of images - a review*," Proceedings of the IEEE, vol 76, no. 8, pp. 917-935, August 1988.

[2] J. Aloimonos and J. Y. Herve, "*Correspondenceless stereo and motion: Planar surfaces*," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 12, no. 5, pp. 504-510, May 1990.

[3] W. H. Narendra Aluja "*Surfaces from Stereo: Integrating feature matching , disparity estimation e contour detection*," IEEE Transaction on Pattern Analysis and Machine Intelligence. vol. 11 no. 2, pp. 121-136, Feb.

[4] J. Heel and S. Begahdaripour, "*Time-sequential structure and motion estimation without optical flow*," SPIE vol. 1260 Sensing and Reconstruction of *Three-dimensional* Objects and Scenes. vol. 1260, pp. 50-61, 1990.

[5] R. M. Bolle and B. C. Vemuri, "*On Three-dimensional surface reconstruction methods*," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 13, no. 1, pp. 1-11, Jan. 1991.

[6] W. Burger and B. Bhanu "*Estimating 3-D egomotion from perspective image sequences*," IEEE Transactions of Pattern Analysis and Machine Intelligence, vol. 12, pp 1040-1058, 1990.

[7] B. Hayashi and S. Negahdaripour, "*Direct motion stereo*," SPIE vol. 1260, Sensing and Reconstruction of *Three-Dimensional* Objects and Scenes.

[8] B. K. P. Horn and B. G. Schunck "*Determining optical flow*," Artificial Intelligence, pp. 185-203, 1981.

[9] B. K.P Horn and E. J. Weldon Jr., "*Direct methods for recovering motion*," International Journal of Computer Vision, vol. 2. pp. 51-76, 1988.

[10] S. R. Negahdaripour and B.K.P. Horn, "*Direct passive navigation*," IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. PAMI-9, pp. 169-176, Jan. 1987.

[11] Y.C. Shah and R. Chapman. "*A new technique to extract reange information form stereo images*," IEEE Transactions on Pattern Analysis and

Machine Intelligence. vol. 11, no. 7, pp.768-781, Nov. 1990.

[12] B. G Schunck. "*Image flow segmentation & estimation by constraint line clustering*" IEEE Transactions on Pattern Analysis and Machine Intelligence. vol. 11 no. 10, Oct. 1990.

[13] C. Q. Shu and Y. Q. Shi "*On unified flow field,*" Pattern Recognition, vol. 24, no. 6, pp. 579-586, 1991.

[14] C. Q. Shu, Y. Zhu, Y. Q. Shi & C. H. Lu. "*Recovering surface structure characterized by an N^{th} order polynomial equation,*" IEEE Seventh Workshop on Multidimensional Signal Processing, pp 23-25. Lake Placid, NY (Accepted), Sep. 1991.

[15] M. Subbrao "*Interpretation of image: A Spatio-temporal approach,*" IEEE Trans. on Pattern Analysis and Machine Intelligence. vol. 11 no. 3, pp. 226-278, March 1990.

[16] T. Suimechomy "*Direct Analytical methods for solving Poisson Equation in computer vision problem,*" IEEE Transaction of Pattren Analysis of Machine Intelligence, vol. 12 no. 5, pp. 435-446

[17] A. Verri, F. Girosi and V. Torre, "*Differential techniques for optical*

flow," Journal Optical Soc. Am. vol. 7, no. 5, May 1990.

[18] R. C. Gonzalez "*Digital image processing*," second edition, Addison Wesley, Reading, MA, 1987.

APPENDICES

APPENDIX A

(Derivatives)

7.1 Appendix A -Spatial Derivatives

This section of the Appendix includes the first-order spatial derivatives.

g_x, g_y and g_s.

We have the gray function as follows:

$$gray = MAG \times sin(K_\theta \times \theta)cos(K_Y \times Y) \tag{7.1}$$

Calculate $\frac{\partial g}{\partial x}$ Note that in the following calculations several temporary variables have been used which have been substituted in the final result to yield the desired results.

Let us define

$$z' = Z - D \tag{7.2}$$

Therefore,

$$\frac{\partial \rho'}{\partial X} = \frac{X}{\rho}$$
$$\frac{\partial \rho}{\partial Y} = \frac{Y}{\rho}$$
$$\frac{\partial \rho}{\partial z'} = \frac{Z - D}{\rho}$$
$$tmp = \sqrt{(y^2 - K^2x^2)}$$

$$tmp1 = (y^2 - x^2K^2 - K^2)$$
$$\frac{\partial z'}{\partial x} = \frac{xK^3D((-2.0 \times K + 2.0 \times tmp - (\frac{tmp1}{tmp}))}{tmp1^2}$$
$$\frac{\partial X}{\partial x} = x \times \frac{\partial z'}{\partial x} + Z$$
$$\frac{\partial Y}{\partial x} = y \times \frac{\partial z'}{\partial x}$$

$$\frac{\partial g}{\partial \theta} = -MAG \times sin(Y \times K_Y) \times K_\theta \times sin(K_\theta \times \theta)$$

$$tmp2 = \sqrt{(\rho \times \rho sin^2(\phi) - X \times X) \times \rho^2}$$

$$\frac{\partial \theta}{\partial X} = -\frac{(\rho^2 - X \times X)}{tmp2}$$

$$\frac{\partial \theta}{\partial Y} = \frac{(X \times Y)}{tmp2}$$

$$\frac{\partial \theta}{\partial z'} = \frac{(X \times (Z - D))}{tmp2} \tag{7.3}$$

$$\frac{\partial g}{\partial Y} = MAG \times (cos(Y \times K_Y) \times K_Y \times cos(K_\theta \times \theta) + sin(K_Y \times y - obj) \times -sin(K_\theta \times \theta) K_\theta \times \frac{\partial \theta}{\partial Y}) \tag{7.4}$$

$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial Y} \times \frac{\partial Y}{\partial x} + \frac{\partial g}{\partial \theta} \times \frac{\partial \theta}{\partial X} \times \frac{\partial X}{\partial x} + \frac{\partial \theta}{\partial Y} \times \frac{\partial Y}{\partial x} + \frac{\partial \theta}{\partial z'} \frac{\partial z'}{\partial x})) \tag{7.5}$$

Calculate $\frac{\partial g}{\partial y}$

$$\frac{\partial z'}{\partial y} = D \times K \times y \times ((\frac{tmp1}{tmp} + 2.0 \times K - 2.0 \frac{tmp}{tmp1^2})$$

$$\frac{\partial X}{\partial y} = x \times \frac{\partial z'}{\partial y}$$

$$\frac{\partial Y}{\partial y} = y \times \frac{\partial z'}{\partial y} + Z$$

$$\frac{\partial g}{\partial y} = (\frac{\partial g}{\partial Y} \times \frac{\partial Y}{\partial y} + \frac{\partial g}{\partial \theta} \times (\frac{\partial \theta}{\partial X} \times$$

$$\frac{\partial X}{\partial y} + \frac{\partial \theta}{\partial Y} \times \frac{\partial Y}{\partial y} + \frac{\partial \theta}{\partial z'} \times \frac{\partial z'}{\partial y})) \tag{7.6}$$

Calculate $\frac{\partial g}{\partial s}$

$$s = \sqrt{(X^2 + Y^2 + (Z - D)^2} \tag{7.7}$$

$$\delta_g = (MAG \times sin(K_Y \times Y) \times cos(K_\theta \times (\theta - \psi))) - gray \tag{7.8}$$

Plane

The spatial derivatives for the plane are as follows:

$$g(X, Y, Z) = k_1 sin(k_2 x Z) sin(k_2 y Z) sin(k_2 Z)$$
$$= k_1 sin(k_2 \frac{xCD}{Ax + By + C}) sin(k_2 \frac{yCD}{Ax + By + C})$$
$$sin(k_2 \frac{CD}{Ax + By + C} \tag{7.9}$$

So,

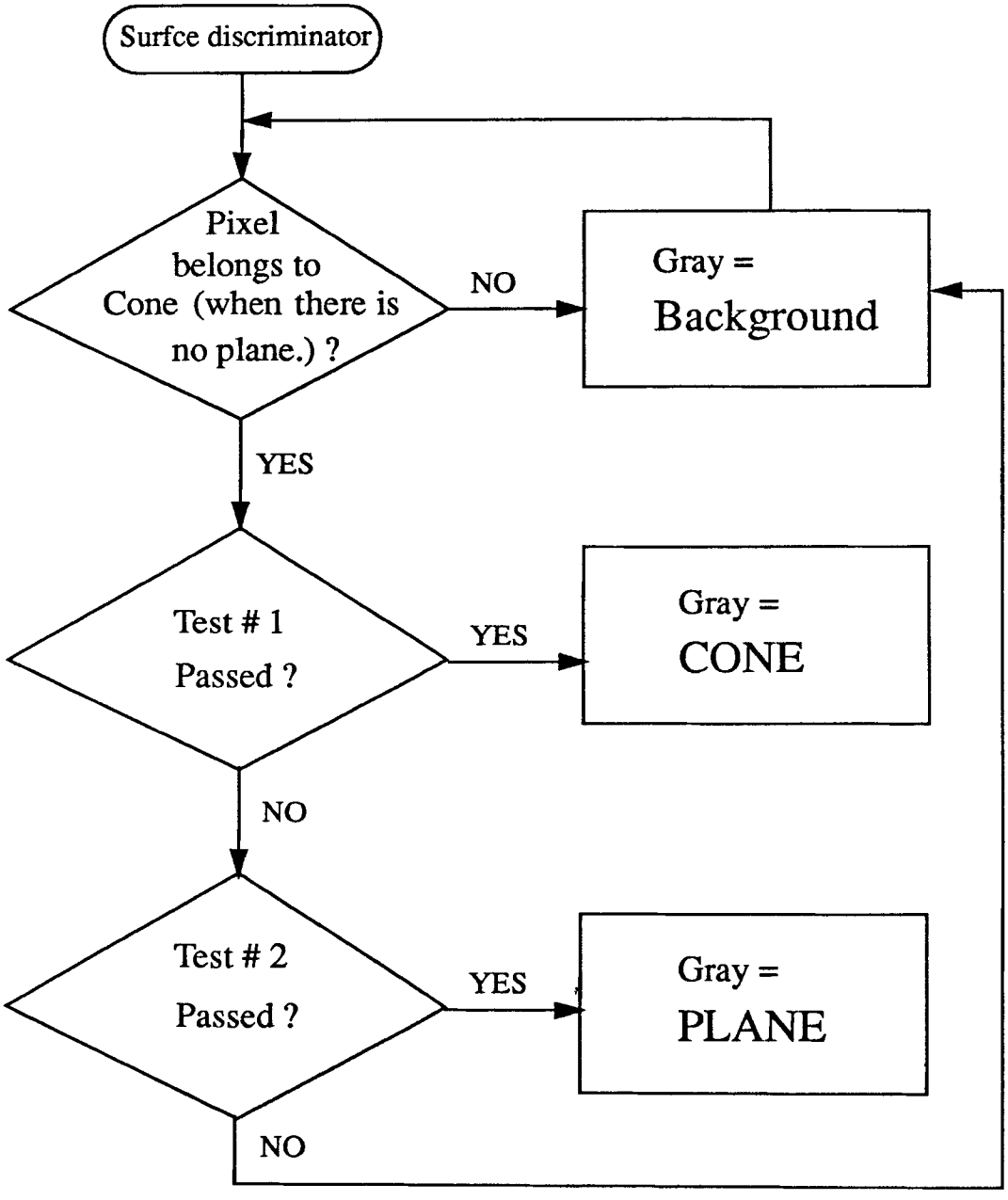
$$g_x = k_1[\frac{k_1 Z^2(By + C)}{CD} cos(k_2 y Z) sin(k_2 Z$$
$$-\frac{k_2 A y Z^2}{CD} sin(k_2 x Z) cos(k_2 y Z) zin(k_2 Z)$$
$$-\frac{k_1 A Z^2}{CD} sin(k_2 x Z) sin(k_2 y Z) cos(k_2 Z)] \tag{7.10}$$

Similaraly,

$$g_y = k_1[\frac{-k_2 B x Z^2}{CD} cos(k_2 y Z) sin(k_2 Z)$$
$$+\frac{k_2 Z^2(Ax + C)}{CD} sin(k_2 x Z) cos(k_2 y Z) sin(k_2 Z)$$
$$-\frac{k_2 B Z^2}{CD} sin(k_2 x Z) sin(k_2 y Z) cos(k_2 Z)] \tag{7.11}$$

APPENDIX B

(Flowchart)



Test # 1: If $A_pX_c + B_pY_c + C_p Z_c + D_p < 0$

Test # 2: If $K^2X_p + Y_p^2 + K^2 Z_p^2 < 0$.

A_p, B_p, C_p and D_p are the coefficients of the PLANE.

X_p, Y_p, Z_p are the coordinates of the PLANE.

X_c, Y_c, and Z_c are the coordinates of the CONE.

CRITERION FOR SURFACE DISTINCTION.

APPENDIX C

(Source Code)

/**

The implementation of the work, as described has been accomplished with the help of two sets of programs.

1. Simulation programs.
2. Reconstruction programs.

As explained earlier, the simulation has been performed on various structures including cone and the composite structure (which is the combination of the cone and the planar surfaces).

This appendix, however, contains the source code for the composite surface only. Different modules constituting the software for the structure are explained one by one and the necessary documentation has been included.

CONE:

The source code to implement the generation and reconstruction of the cone surface consists of several modules, which are discussed in the following.

Generation:

The generation (or the simulation) of the cone is accomplished with the help of two modules:

1. cone.c
2. plane.c
3. globe.par

As evident from the names used, the first module (cone.c) is the program to simulate the cone structure (both for the left and the right camera). Similarly the plane.c simulates the takes care of the pixels associated with the plane of the composite surface. The other file (globe_par) contains the global parameters used by the con.c program. Since, these parameters may be quite

different under different circumstances, the globe_par file makes changes easy to make.

RECONSTRUCTION:

The reconstruction of the cone surface is performed using the following modules:

1. Reconst.c
2. Param.c
3. Gauss.c

Again, here the naming convention used is self-explanatory. That is, the first module (reconst_cone.c) is the main program to recover the structure of the surface. The second file (param_cone) contains the parameters to be adjusted, while the third file (gauss.c) simply implements the Gauss's formula. This program is called from the reconst_cone.c program when the Gauss's formula has to be used for the matrix solution for the linear equations.

Besides, further comments and documentation is provided in the respective modules.

****

Module # 1: Cone.c (Simulation of the cone surface)

```
#include <stdio.h>
#include <math.h>
#include "globe.par"
#include "thesis_pln.c"

FILE *fp;

main()
{
```

```
/*  Variables to be used in the program  */

int i,j;
int flag;
double x,y;
double rho, theta, sin_rho, cos_theta;
double temp_i, temp_j, grey;
double x_obj, y_obj, z_obj;
double z_den,tmp,tmp1,tmp2,tmp_th,z_dash;

/*
Variables to be used for the partial derivatives. A
little consideration reveals the partial derivative, each
variable refers to.
*/

double g_to_r,g_to_theta,g_to_x,g_to_y,delta_g,g_to_yo;
double r_to_xo,r_to_yo,r_to_z_dash;
double z_dash_to_x,xo_to_x,yo_to_x;
double th_to_x_obj,th_to_y_obj,th_to_z_dash;
double xo_to_y,yo_to_y,z_dash_to_y;
double s;
double gr_pl_l,gr_pl_r;

double x_pln,y_pln,z_pln;
int flag_1;
double theta_p;

flag=0;
flag_1=0;

fp=fopen("CONE_OUT", "w");

/*
Scanning across x-axis and y-axis starts here. 'i'
corresponds to the x-axis and 'j' corresponds to the y-
axis.
*/
```

```
        for(j=0;j<N;j++)
                {
                for(i=0;i<N;i++)
                        {
                        temp_i=(double)(i);
                        temp_j=(double)(j);


/*
Calculations of the points on the image plane. The real
numbers used are to select the window to be taken for
observation.
*/


                        x= -.04  + (.08 * (temp_i/N));
                        y=  .05 - (.08 * (temp_j/N));

        flag= echo_flag(x,y, &x_obj, &y_obj, &z_obj);
/*
If any pixel belongs to the image of the surface of the
cone, the value of the flag returned is non-zero.
*/

        if(flag != 0)

/*
Function to compute the rho and the angle theta (with the
y-axiz) givent the various coordinates of the point.
*/
                {

flag_1=echo_flag_for_plane(x,y,x_obj,y_obj,z_obj,&x_pln
,&y_pln,&z_pln);

if(flag_1 ==1)
                {
                echo_theta_rho(x_obj, y_obj, z_obj,
&theta, &rho);
                sin_rho = sin(y_obj*K_RHO);
```

```
                cos_theta = cos(K_THETA * (theta));

                grey = fabs(mag * sin_rho * cos_theta);


/*-----CALCULATIONS FOR DERIVATIVES W.R.T. X-AXIS-----*/

        r_to_xo=x_obj/rho;
        r_to_yo=y_obj/rho;
        r_to_z_dash=(z_obj - D)/rho;
        tmp=sqrt(y*y-K*K*x*x);
        tmp1=(y*y-x*x*K*K-K*K);
        z_dash_to_x= x*K*K*K*D*((-2.0*K+2.0*tmp-(tmp1/
tmp))/(tmp1*tmp1));
        xo_to_x=x*z_dash_to_x + z_obj;
        yo_to_x=y*z_dash_to_x;

        g_to_theta=(-
mag*sin_rho*K_THETA*sin(theta*K_THETA));
        tmp_th=sqrt(rho*rho*sin(PHI)*sin(PHI)-
x_obj*x_obj)*rho*rho;
        th_to_x_obj=(-(rho*rho-x_obj*x_obj)/tmp_th);
        th_to_y_obj=((x_obj*y_obj)/tmp_th);
        th_to_z_dash=((x_obj*(z_obj-D))/tmp_th);

       g_to_yo=mag*(cos(y_obj*K_RHO)*K_RHO*cos_theta +
sin_rho*
            (-sin(theta*K_THETA)*K_THETA*th_to_y_obj));

        g_to_x=(g_to_yo*yo_to_x  +
g_to_theta*(th_to_x_obj*
                xo_to_x + th_to_y_obj*yo_to_x +
th_to_z_dash*z_dash_to_x));


/*------CALCULATIONS FOR DERIVATIVES W.R.T Y-AXIS-----*/


        z_dash_to_y=D*K*y*((tmp1/tmp + 2.0*K - 2.0*tmp)/
(tmp1*tmp1));
```

```
        xo_to_y= x*z_dash_to_y;
        yo_to_y= y*z_dash_to_y + z_obj;

        g_to_y=(g_to_yo*yo_to_y  +
g_to_theta*(th_to_x_obj*
                 xo_to_y + th_to_y_obj*yo_to_y +
th_to_z_dash*z_dash_to_y));


/*-----------------------------------------------------

 CALCULATION FOR DERIVATIVE W.R.T. THE PARAMETER S.

-----------------------------------------------------*/


        s=sqrt(x_obj*x_obj + y_obj*y_obj + (z_obj-
D)*(z_obj-D));
        delta_g=fabs(mag*sin_rho*cos(K_THETA*(theta-
BETA)))-grey;


/*
Different paramers are printed in the output file to be
read by the file to reconstruct the structure.
*/

        fprintf(fp, "%d %.16e %.16e %.16e %.16e %.16e
%.16e %.16e\n",
              flag_1,x,y,grey,g_to_x,g_to_y,delta_g,s);


         }



        else

                if(flag_1==2)
                {
```

```
            z_dash = z_pln - Z0;


plane(x,y,fp);

                    }
          else

                    {
 fprintf(fp, "%d %.16e %.16e %.16e %.16e %.16e %.16e
%.16e\n",
                  flag_1,x,y,0.0, 0.0, 0.0, 0.0, 0.0);


                    }
                    }
          else

                    {
 fprintf(fp, "%d %.16e %.16e %.16e %.16e %.16e %.16e
%.16e\n",
                  flag_1,x,y,0.0, 0.0, 0.0, 0.0, 0.0);

                    }
           }
     }
}


/*****************************************************

The  function, echo_flag() passes the values of the
coordinates of x and y (image plane), and it returns the
value of the flag, which determines if a certain point on
the  image plane corresponds to the valid point on the
object space (flag=1) or not (flag=0).

*****************************************************/

int echo_flag(x,y, x_cordinate, y_cordinate,
```

```
z_cordinate)

double x,y;
double *x_cordinate;
double *y_cordinate;
double *z_cordinate;
{
double rho, theta;
double x_object, y_object, z_object;
double temp, z_temp;
double temp1, temp1_sq, temp2, temp2_sq;
double z_numerator_pos, z_numerator_neg, z_numerator,
z_denominator;

/*-------------------------------------------------

CALCULATIONS FOR FINDING THE Z_COORDINTE, FOLLOWED BY
                X, Y, THETA AND RHO.
-------------------------------------------------*/

temp1 = y * K * D;
temp1_sq = pow(temp1, 2.0);

temp2 = pow(K,2.0);
temp2_sq = pow(temp2 * x * D, 2.0);
temp= temp1_sq - temp2_sq;

        if(temp < 0.0)
                {
                return(0);
/*
'0' is returned, whenever the pixel belongs to the
background.
*/
                }
        else
                {

z_temp = sqrt(temp);
z_numerator_pos = D * temp2 + z_temp;
```

```
z_numerator_neg = D * temp2 - z_temp;

        if(z_numerator_neg > 0.0 && z_numerator_neg <=
                z_numerator_pos)
                {
                z_numerator = z_numerator_neg;
                }
                else
                {
                z_numerator = z_numerator_pos;
                }

z_denominator = temp2 - pow(y,2.0) + pow((K*x),2.0);

        if(z_denominator == 0.0)
                {
                return(0);
                }
                else
                {

z_object = z_numerator/z_denominator;
x_object = x * z_object;
y_object = y * z_object;

*x_cordinate=x_object;
*y_cordinate=y_object;
*z_cordinate=z_object;



/*
 A certain arbitrary height 'MAX-Y' of the cone structure
is chosen to be used for the simulation.
*/

        if(y_object < 0.0 || y_object > MAX_Y)
                {
                return(0);
                }
```

```
        else
                {

                return(1);
                }
                }
      }

}

/*-----------------------------------------------------

FUNCTION TO RETURN THE VALUE OF THETA AND ROH (for given
x_imageand y_image.)

-----------------------------------------------------*/


int echo_theta_rho(x_cord, y_cord,
z_cord,theta_buf,rho_buf)
double x_cord, y_cord, z_cord;
double *theta_buf, *rho_buf;

{
double rho, theta;
double sin_phi, cos_phi, temp;
double temp_theta, temp_rho, echo_theta;
double temp_z_cord;


/*------------CALCULATIONS FOR ROH--------------*/


temp_z_cord = fabs(z_cord - 100.0);  /*  gives z_cord of
the object space */

temp = pow(x_cord, 2.0) + pow(y_cord, 2.0) +
pow(temp_z_cord, 2.0);
temp_rho = sqrt(temp);
```

```
*rho_buf = temp_rho


/*-----------CALCULATIONS FOR THETA------------*/


sin_phi = sin(PHI);
temp_theta = (x_cord/(temp_rho*sin_phi));
if(temp_theta < -1.0 || temp_theta > 1.0)
        {

        take_care_of_theta(&temp_theta);
        }

theta = M_PI - asin(temp_theta);
*theta_buf = theta;

}

/*-------------------------------------------------

Function to take care of the domain eror because of the
aurguments of arcsin() function, slightly above 1.0 or
below -1.0.

-------------------------------------------------*

take_care_of_theta(arg_arcsin)

double *arg_arcsin;

{
double temp;

temp = *arg_arcsin;
if(temp < -1.0)
        temp = -1.0;
else
        temp = 1.0;
```

```
*arg_arcsin = temp;
}



/*--------------------END----------------------*/
```

```
/*****************************************************

Plane.c starts here:

This is a part of the source code to simulate the plane
of the composite  structure. Basically, it generates the
data required to reconstruct the planar surface.

*****************************************************/


#define CONE  2
#define PLANE 1


#include <stdio.h>
#include <math.h>
#include "globe.par"

double A_L,B_L,C_L,D_L;


/-----------------------------------------------------

Function to check the criterion discriminate between
the surfaces of the cone and plane.

-----------------------------------------------------*/

int
echo_flag_for_plane(x,y,x_cone,y_cone,z_cone,x_plane,y_
plane,z_plane)

double x,y;
double x_cone;
double y_cone;
double z_cone;
double *x_plane;
```

```
double *y_plane;
double *z_plane;
{

int i,j;
double x_coordinate,y_coordinate,z_coordinate,z_dash;
double radius, temp_d;
double plane;


/*--------Coefficients of the planar surface---------*/


A_L =   (cos((double)(61.8*M_PI/180.0)));
B_L =   (cos((double)(35.3*M_PI/180.0)));
C_L =   (-sqrt(1.0-A_L*A_L - B_L*B_L));
D_L =   (-(B_L*Y0 + C_L*Z0));


/*--------Calculate the coordinates of the plane------*/


z_coordinate = (double)(-D_L/(A_L*x+B_L*y+C_L));
y_coordinate=z_coordinate*y;
x_coordinate=z_coordinate*x;


z_dash=z_coordinate - 100.0;

*x_plane=x_coordinate;
*y_plane=y_coordinate;
*z_plane=z_coordinate;



/*--Criterion for discriminating between the surfaces-*/

if(A_L*x_cone + B_L*y_cone + C_L*z_cone + D_L < 0)
        return(CONE);
```

```
else if(K*K*x_plane + y_plane*y_plane +
K*K*z_plane*z*plane < 0)
         return(PLANE);

else
        return(0);
}
}


/*-----------------------------------------------------

Function to calculate the quantities required for the
surface recovery of the planar surface.

-----------------------------------------------------*/

plane(x,y,fp)
double x;
double y;
FILE *fp;

{
int i,j,flag,count;
double gray_l,gray_r,gs,gx,gy,sz;
double sx1_l,sy1_l,sz1_l,A_L,B_L,C_L;
double
sx1_r,sy1_r,sz1_r,A_R,B_R,C_R,sx1_r_l,sy1_r_l,sz1_r_l;
double df1,dd1,dr1,d1,D_L,D_R,rz,ry,rc;

  dd1=(double)(d);
  dr1=(double)(r);


A_L =   (cos((double)(61.8*M_PI/180.0)));
B_L =   (cos((double)(35.3*M_PI/180.0)));
C_L =   (-sqrt(1.0-A_L*A_L - B_L*B_L));
D_L =   (-(B_L*Y0 + C_L*Z0));

/*-----------------------------------------------------
```

```
Coefficients corresponding to the representation of the
plane for the right camera.

-------------------------------------------------------*/

  A_R=A_L*cos(phi)+C_L*sin(phi);
  B_R=B_L;
  C_R=-A_L*sin(phi)+C_L*cos(phi);

 D_R=D_L+A_L*a;

 flag=2;

/*-------------------------------------------------------

Computation for the coordinates of the object space for
the left camera and the right camera.

-------------------------------------------------------*/


      sz1_l=(-D_L)/(A_L*x+B_L*y+C_L);
      sx1_l=sz1_l*x;
      sy1_l=sz1_l*y;

      sz1_r=(-D_R)/(A_R*x+B_R*y+C_R);
      sx1_r=sz1_r*x;
      sy1_r=sz1_r*y;


/*-------Transformation for brightness invariance-----*/


      sx1_r_l=cos(phi)*sx1_r-sin(phi)*sz1_r+a;
      sy1_r_l=sy1_r;
      sz1_r_l=sin(phi)*sx1_r+cos(phi)*sz1_r;


gray_l=mag*sin(k1*sx1_l)*sin(k1*sy1_l)*sin(k1*sz1_l);
```

```
gray_r=mag*sin(k1*sx1_r_1)*sin(k1*sy1_r_1)*sin(k1*sz1_r
_1);


        sz=sz1_l;

/*----Calculate the partial derivitives, gx, gy, gs----*/


      gx=mag*((k1*sz*sz*(B_L*y+C_L)/
(C_L*dd1))*cos(k1*sx1_l)*sin(k1*sy1_l)*sin(k1*sz1_l)-
(k1*A_L*y*sz*sz/
(C_L*dd1))*sin(k1*sx1_l)*cos(k1*sy1_l)*sin(k1*sz)-
(k1*A_L*sz*sz/
(C_L*dd1))*sin(k1*sx1_l)*sin(k1*sy1_l)*cos(k1*sz));
      gy=mag*(-(k1*x*B_L*sz*sz/
(C_L*dd1))*cos(k1*sx1_l)*sin(k1*sy1_l)*sin(k1*sz1_l)+(k
1*sz*sz*(A_L*x+C_L)/
(C_L*dd1))*sin(k1*sx1_l)*cos(k1*sy1_l)*sin(k1*sz)-
(k1*B_L*sz*sz/
(C_L*dd1))*sin(k1*sx1_l)*sin(k1*sy1_l)*cos(k1*sz));
      gs=gray_r-gray_l;


     fprintf(fp,"%d %.16e %.16e %.16e %.16e %.16e %.16e
%.16e\n", flag, x, y,gray_l, gx,gy, gs, sz);

*/
return(1);

}

/**************************END**************************/
```

```
/*****************************************************

Globe.par starts here:

This file is the data file for the global parameters used
for the program to generate, as well as to recover the
image of a conical surface.

*****************************************************/


#define D  100.0
                /* distance between the center
                   point and origin of the
                   coordinate system */

#define mag    1000.0
                /* Magnification for grey level */

#define PHI   30.0*M_PI/180.0
                /* Corresponding phi=5 degree.  */

#define BETA   0.001*M_PI/180.0
                /* Corresponding to .1 degree.  */

#define K_THETA    8.0
                /* theta factor                    */
#define K_RHO      8.0
                /* rho factor                      */

#define N   128

#define K   3.732050808
             /* slope of the edge   y=sqrt(x**2 + z**2) */

#define MAX_Y      5.0

#define Y0         MAX_Y*(1.0/2.0)
```

```
#define Z0          100.0

#define phi 0.1*M_PI/180.0
                  /* angle of image shift        */

#define a phi*100.0

#define a1 61.8*M_PI/180.0

#define b1 35.3*M_PI/180.0
                  /* vector of three planes          */

#define d 100.0
                   /* depth of object                */
#define r 2.0
                  /* width of object                 */

#define k1 8
                 /* freq of three plane gray        */

#define k2 8

#define k3 8

#define k 1

/*****************************END************************/
```

```
/*****************************************************

Reconst.c starts here:

As explained before, it is used for the reconstruction of
the cone surface . It also, calls the other modules like
'gauss.c' and 'param.c' to accomplish the reconstruction.

*****************************************************/

#include "gauss.c"
#include "param"

/*
As explained before, the gauss.c implements the gauss's
formula for the matrix solution, where as 'param'
contains the data for this purpose.
*/

main()
{
   extern quation();
   FILE *fp;
   int i, j;
   int count;
   double coff[ORDER];
   double m[ORDER][ORDER], d[ORDER], lamda[ORDER];
   double m_c[ORDER][ORDER], d_c[ORDER];
   double U, V, W, A, B, C;
   double p, q, r;
   double temp1, temp2;
   int flag;
   double x,y,gray,g_to_x,g_to_y,delta_g;
   double x0, y0, z0, R;
   double Z, Z0;
   double minima, K;

   U=100.0*SMALL;
   V=0.0;
```

```
   W=0.0;
   A=0.0;
   B=-SMALL;
   C=0.0;
   minima=0.0;
   K=10000.0;

   for( i=0; i<ORDER; i++ ) {
      for ( j=0; j<ORDER;  j++ )  {
          m[i][j] = 0.0;
      }
   }

   for( i=0; i<ORDER; i++ ) d[i]=0.0;

/*
The reconstruction program starts here:

First, the output file, written by the simulation program
is read and then the values of the variables manipulated
to recostruct the composite structure surface.
*/

   if( (fp=fopen( CONE_OUT, "r" )) == (FILE *)NULL ) {
      perror( CONE_OUT );
   }
   else {
      for( i=0; i<100; i++ )
         fscanf(fp,"%d%le%le%le%le%le%le%le",
           &flag,
&x,&y,&gray,&g_to_x,&g_to_y,&delta_g,&Z);
      count=1;
      while( fscanf(fp,"%d%le%le%le%le%le%le%le",
           &flag,
&x,&y,&gray,&g_to_x,&g_to_y,&delta_g,&Z) != EOF &&
            count< SIZE*SIZE-200 ) {

/*
Only the pixels corresponding to the cone (flag = 1)
surface are scanned, the background is ignored (flag =
```

```
0).
*/

          count=count+1;
          if( flag == 1 || flag == 2) {
            p=(-A*y+B*x)*(x*g_to_x+y*g_to_y)-g_to_x*(-
B+C*y)
                -g_to_y*(-C*x+A)-delta_g;
            q=x*W*g_to_x+y*W*g_to_y-U*g_to_x-V*g_to_y;
            minima=(q-100.0*p);

/*-------------------------------------------------
Coefficients corresponding to the various terms of the
polynomial equation are computed here. Note that coff[0]
is the constant term (and not the coefficient of the the
term involving X*X*X ). The rest of the terms are in
order.
*/

             coff[0]=p*p*p;
             coff[1]=q*y*q*y*q*y;
             coff[2]=minima*minima*minima;
            coff[3]=q*x*q*y*x*q;
            coff[4]=q*x*x*q*minima;
            coff[5]=q*q*q*y*y*x;
            coff[6]=q*q*y*y*minima;
            coff[7]=q*x*minima*minima;
            coff[8]=y*q*minima*minima;
            coff[9]=q*x*q*y*minima;
            coff[10]=q*q*p*x*x;
            coff[11]=p*y*y*q*q;
            coff[12]=p*minima*minima;
            coff[13]=q*x*y*q*p;
            coff[14]=q*x*p*minima;
            coff[15]=q*y*p*minima;
            coff[16]=q*x*p*p;
            coff[17]=q*y*p*p;
            coff[18]=p*p*minima;
             for( i=0; i<ORDER; i++ ) {
               d[i]=d[i]-coff[i]*q*q*q*x*x*x;
```

```
                for( j=0; j<ORDER; j++ ) {
                   m[i][j]=m[i][j]+coff[i]*coff[j];
                }
             }
          }
       }
    }

    for( i=0; i<ORDER; i++ ) {
       d_c[i]=d[i];
       for( j=0; j<ORDER; j++ ) {
          m_c[i][j]=m[i][j];
       }
    }


/*-------------------------------------------------

Various results are printed here:

-------------------------------------------------*/

    for( i=0; i<ORDER; i++ )  {
       for( j=0; j<ORDER; j++ ) {
          printf( " %f* ", m[i][j] );
       }
       printf( "\n" );
    }

   for ( i=0; i<ORDER; i++ ) printf( "d=%d %f\n", i, d[i]
);

/*
The function gauss() is called for the matrix solution as
explained before.
*/

    gauss( ORDER, m, d, lamda );

    for( i=0; i<ORDER; i++ )
```

```
        printf("coff[%d]: %f\n ",i, lamda[i] );
}

/******************End of reconst.c ***************/
```

```
/*****************************************************

Gauss.c starts here:

This module  is a part of the reconstruction program.
It  implements  the Gauss's  formula  to  recover  the
parameters of  the the surface  structure. It  actually
implements Gauss's formula to solve the  matrices.

It simply codes the mathematics involved in the formula,
which can be referred in the thesis or any good
mathematics book.

*****************************************************/


#include "es.inc"

gauss( n, a, b, result )
int n;
double a[ORDER][ORDER], b[ORDER], result[ORDER];
{
   double s[ORDER], m[ORDER][ORDER], sum;
   int nrow[ORDER];
   int i, j, p, ncopy, jj;

/*
The computation starts here:
*/

   for( i=0; i<n; i++ ) {
      s[i]=fabs( a[i][0] );
      for( j=1; j<n; j++ ) {
         if( fabs( a[i][j] ) > s[i] ) s[i]=fabs( a[i][j]
);
      }
      if( s[i] == 0 ) {
         printf( " no unique solution exists \n " );
         exit( -1 );
```

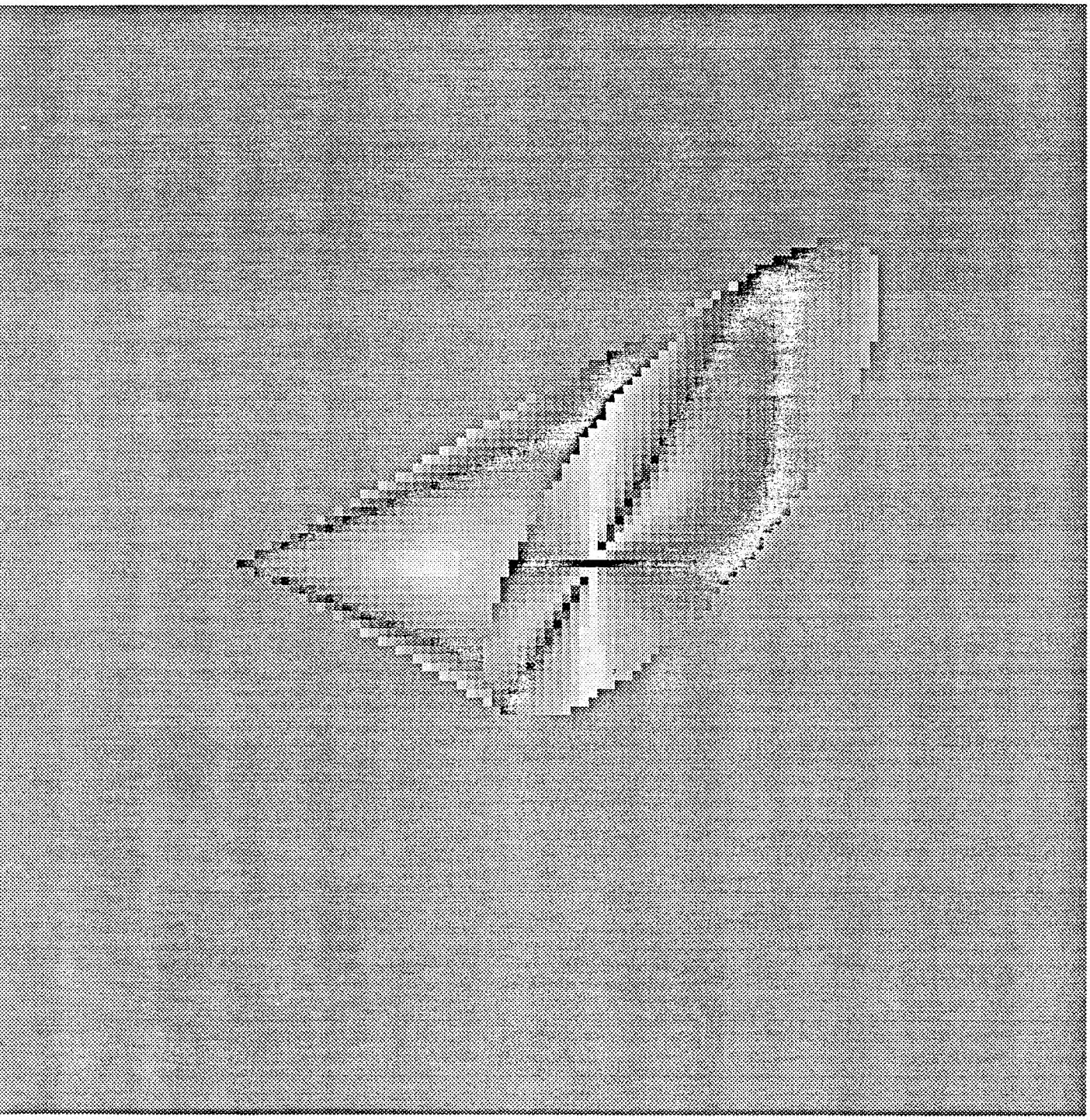
```
        }
    }

    for ( i=0; i<n; i++ ) nrow[i]=i;

    for( i=0; i<n-1; i++ ) {

        /* step 3 */
        p=i;
        for( j=i+1; j<n; j++ ) {
            if( fabs( a[nrow[j]] [i] ) / s[ nrow[j] ] >
                fabs( a[nrow[p]] [i] ) / s[ nrow[p] ]   )
p=j;
        }

        /* step 4 */
        if( a[nrow[p]][i]==0 ) {
            printf( " no unique solution exists \n " );
            exit( -2 );
        }

        /* step 5 */
        if( nrow[i] != nrow[p] ) {
           ncopy=nrow[i];
            nrow[i]=nrow[p];
            nrow[p]=ncopy;
        }

        /* step 6 */
        for ( j=i+1; j<n; j++ ) {
           m[nrow[j]][i]=a[nrow[j]][i]/a[nrow[i]][i];
            for( jj=0; jj<n; jj++ ) {
               a[nrow[j]][jj]-
=m[nrow[j]][i]*a[nrow[i]][jj];
            }
            b[nrow[j]]-=m[nrow[j]][i]*b[nrow[i]];
        }
    }

    if( a[nrow[n-1]][n-1] == 0 ) {
```

```
      printf( " no unique solution exists \n " );
      exit ( -3 );
   }

   result[n-1]=b[nrow[n-1]]/a[nrow[n-1]][n-1];

   for( i=n-2; i>=0; i-- ) {
      sum=0.0;
      for( j=i+1; j<n; j++ ) {
         sum=sum+a[nrow[i]][j]*result[j];
      }
      result[i]=( b[nrow[i]]-sum )/a[nrow[i]][i];
   }
}



improvement( n, a, b, result )
int n;
double a[ORDER][ORDER], b[ORDER], result[ORDER];
{
   double delta[ORDER];
   double sum;
   int i, j;

   for ( i=0; i<n; i++ ) {
      sum=0.0;
      for ( j=0; j<n; j++ ) {
          sum=sum+a[i][j]*result[j];
      }
      b[i]=sum-b[i];
   }
   gauss( n, a, b, delta );
   for ( i=0; i<n; i++ ) {
       result[i]-=delta[i];
   }
}


/**********************END OF GAUSS.C*****************/
```

```
/*****************************************************

This is a parameter file for the reconst.c to reconstruct

the surface.

*****************************************************/

#include <math.h>
#include <stdio.h>

#define CONE_OUT "CONE_OUT"
#define SMALL 0.1*M_PI/180.0
#define SIZE 128
#define ORDER 19

/
**************************END**************************/
```

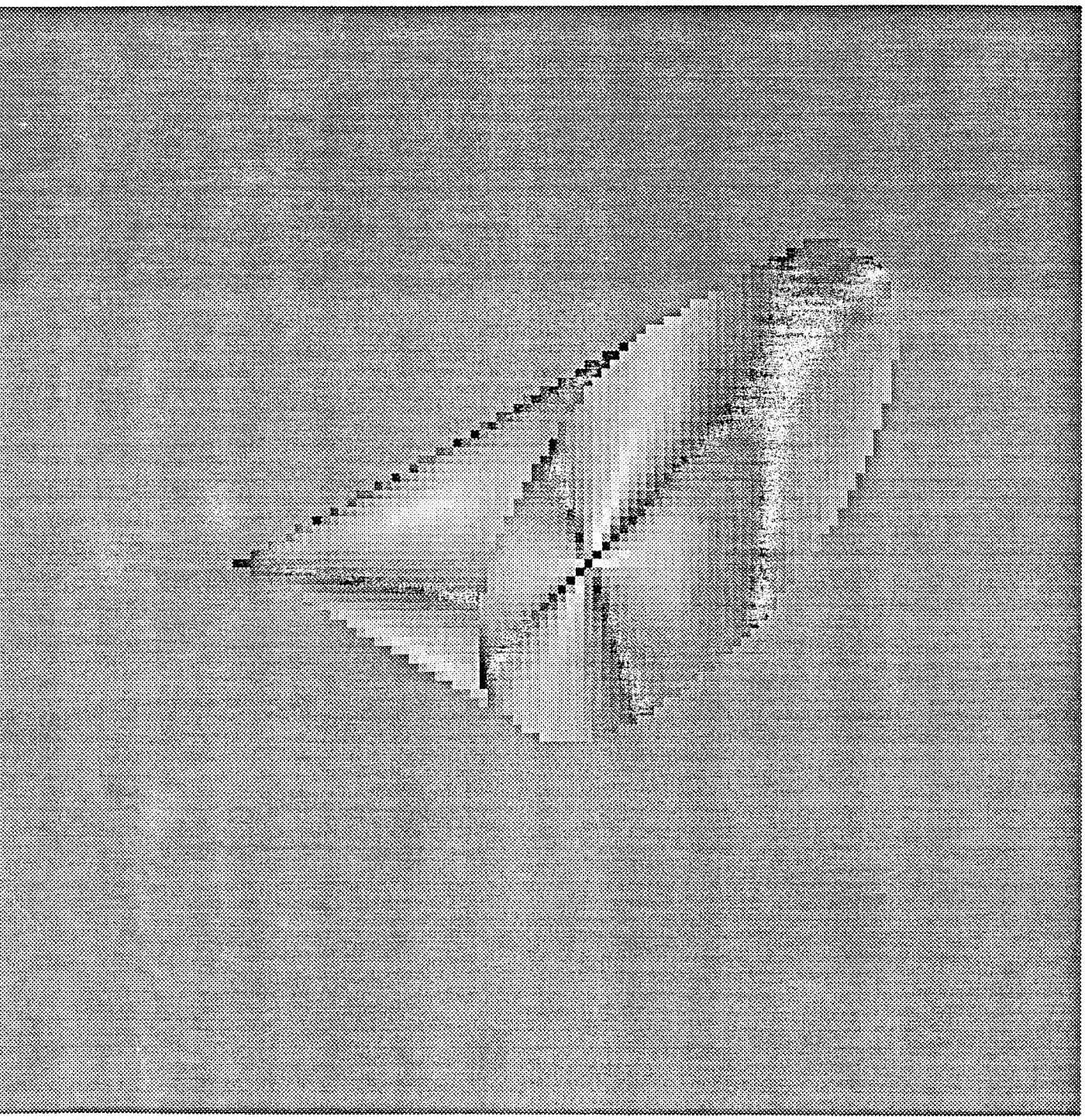
APPENDIX D

(Stereo Images)



COMPOSITE: RIGHT CAMERA IMAGE

6



COMPOSITE LEFT CAMERA IMAGE