A Software Assistant for Manual Stereo Photometrology

Paul Sheer

A dissertation submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, October 1997

Declaration

I declare that this dissertation is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other university.

Signed this ___ day of _____ 19___

_____

Paul Sheer.

1 Introduction

1.1 Project Requirements

The drawing and surveying department of AECI[1] requires a software assistant for their manual photometrology facility. The department uses stereo photographs to produce CAD drawings (required by legislature) of their industrial plants, where such drawings may be outdated or non-existent. Their current facility is a text-based desktop computer linked to a digital drawing board. This setup is painstaking to use, requiring hand calculations and pencil recording of data points. It lacks many features that would be easy to support with modern graphical computer interfaces and electronic storage. This project aims to create a package which would run on an inexpensive desktop computer, enabling greater throughput and ease of use.

The present procedure for producing CAD drawings is as follows:

1. First approximately six calibration points are placed at strategic points on plant equipment in a region of a plant. A land surveyor then determines the precise location of these points.

2. Two photographs of the scene are taken, and developed prints of approximately 20 by 15 centimetres in size are pasted side by side on the drawing board.

3. A digital mouse is then used to take measurements from the photographs, which are fed to the computer. The computer performs the task of camera calibration, triangulation, and the additional functions of fitting planes and finding direction vectors.

4. The resulting 3D data is then used to manually construct layout drawings of the scene using a commercial CAD package.

[1] AECI ENGINEERING (Pty) Ltd, Modderfontein South Africa, P O Box 796 Germiston 1400

Abstract

A software package was written under the X Window System, Version 11, to assist in manual stereopsis of multiple views. The package enables multiple high resolution (2000 by 1500 pixels and higher) black and white photographs to be viewed simultaneously. Images have adjustable zoom windows which can be manipulated with the pointing device. The zoom windows enlarge to many times the resolution of the image enabling sub-pixel measurements to be extrapolated by the operator. A user-friendly interface allows for fast pinhole camera calibration (from known 3D calibration points) and enables three dimensional lines, circles, grids, cylinders and planes to be fitted to markers specified by the user. These geometric objects are automatically rendered in 3D for comparison with the images. The camera calibration is performed using an iterative optimisation algorithm which also tries multiple combinations of omitted calibration points. This allows for some fault tolerance of the algorithm with respect to erroneous calibration points. Vector mathematics for the geometrical fits is derived. The calibration is shown to converge on a variety of photographs from actual plant surveys. In an artificial test on an array of constructed 3D coordinate markers, absolute accuracy was found to be 1 mm (standard deviation of the Euclidean error) for a distance of 2.5 meters from a standard 35 mm camera. This translates to an error of 1.6 pixels in the scanned views. Lens distortion was assumed to be negligible, except for aspect ratio distortion which was calibrated for. Finally, to demonstrate the efficacy of the package, a 3D model was reconstructed from ten photographs of a human face, taken from different angles.

This straightforward methodology is reputed to produce CAD drawings to within a few millimetres accuracy in scenes that span over ten meters. The alterations proposed are:

- Scan images at high resolution into a database.
- Use a window-based computing environment to allow graphical manipulation of the images.
- Allow dynamic storage and recall of images, calibration data, and 3D measurements.
- Output measured 3D data to a usable format for inclusion into a CAD package.

These alterations are tractable within the scope of an MSc dissertation, and will be a worthwhile improvement to the present facility.

1.2 Background to Stereopsis

Stereoscopic image processing is a computational science that is used to extract 3D information from more than one 2D image of a scene. The image of a 3D scene will have disparity differences when viewed from different angles, and this can be used to extract range information from a scene. Current research aims to achieve such measurements automatically, using complex image processing algorithms that exhibit artificial intelligence. For a thorough review of such research up to 1988 see [4].

Here we are concerned with manual — rather than automatic — reconstruction, where a human operator will find correspondences between two views. This is a well covered area in research, however the sciences of camera calibration and photometrology are still areas of active research.

1.2.1 Obtaining stereo measurements

Typically the operator wishes to measure some 3D coordinate from multiple 2D projections of that coordinate. With the rotation matrix of the camera known, as well as its focal length and 3D position, to find a 3D coordinate merely requires the interception to be found of the vectors extending out from each camera toward the point. Obviously, the smaller the baseline of the cameras (i.e. the distance between their

Contents

optical centres) the closer to parallel are the vectors, and the more prone the triangulation is to error. In automatic disparity measurement a small baseline is desirable so that the images are similar, and the algorithm will have less difficulty in finding feature correspondences. In manual stereo measurement, this baseline would be high for greater accuracy, where the human operator would have little difficulty in finding correspondences between the (often vastly different) views. Three cameras may be used to give the benefits of both schemes, and even more cameras will give a statistical indication of the best 3D triangulation.

[1] compares different camera geometries and relates camera geometry to the correspondence problem. See [7] for a complete reference on 3D machine vision and its geometric problems, as well as [6], [9], [12], [13] for general references.

1.2.2 Camera calibration

For most applications the camera orientation and position are determined from known points in the images and their measured projections. Approaches to this non-linear problem can be found in [5], [14], [15] and [17]. The most thorough and succinct method found is that of [8]. Here, the system is reduced to sets of multivariate polynomials which yield the camera rotation matrix, position and other variables such as the aspect ratio and skew factor. Any other lens distortion is expected to be known beforehand as two bivariate polynomials.

Because the camera used is of high quality, there is no need to calibrate for lens distortion parameters. The present system uses the photographs directly with as few as four calibration points — hence its accuracy is achieved without distortion calibration. The only further distortion might be by an aspect ratio change from the scanner used to digitise the images. An aspect ratio parameter will hence be included optionally in the calibration algorithm for the project.

1.3 Objectives

The above background translates into features that must be present in a software and hardware system. The specific objective of this project is therefore implementation of such a system. AECI has however already investigated commercially available alternatives to their present system. Since none could be found that were suitable, a new software package must be written. The package is also meant to be a simple and

inexpensive replacement of an existing system, and must run under a readily available and inexpensive computer platform.

1.4 Requirements

The requirements of the software are as follows:

- The code must be written in a standard programming language, so as to be maintainable.
- The programming code must be extensible. It must hence be written in a modular style.
- The program must run on inexpensive hardware.
- The accuracy of the existing setup must be determined more certainly. The new system must meet or exceed this accuracy.
- The program must be substantially easier and faster to use than the existing system.
- Thorough testing must be done to establish the precise accuracy of the system. It should also be established, which mechanisms have the most influence on the accuracy in order to provide recommendations for future stereo measurement.

The precise functions which the program is to perform can be outlined as follows:

- Allow multiple images to be loaded simultaneously.
- Calibrate any necessary camera parameters.
- Fit various geometric primitives to coordinates specified in the image.
- Output the measured data in a structured format for rendering, analysis or inclusion into a CAD package.

1.5 Work Strategy and Project Plan

Having reviewed current methods in stereopsis, the remaining work strategy is as follows:

1. Decide on a computing environment to use for development.
2. Establish efficient mathematical techniques for the utilities we would like to implement.
3. Develop the algorithms necessary to perform calibration, triangulation, and geometric fits.
4. Establish a practical graphical layout for the program.
5. Develop the graphical devices necessary to implement the desired desktop layout.
6. Perform thorough tests with regard to each aspect of the program.

List of Figures

2 The Software Application

2.1 The Application under the X Window System

To fulfil the aims discussed above, the package was written under Unix running the X Window System, Version 11. Here the programmer is free to choose from a variety of widget libraries[1]. Although using an available widget library would ease some of the programming effort, many of the graphical devices desirable in the application would require low level programming anyway. It was therefore considered expedient to write the application directly under XLib (the fundamental X Window System interface language, see [2]) which would give it its own unique look and feel as well as making it faster and more memory efficient.

The entire application is written in C. It attempts mostly to follow the GNU coding standards, and uses the Kernighan and Ritchie coding style. See Appendix A for further discussion on the features of the software.

2.2 Emulation of the Existing Setup

The digital mouse and drawing board, presently used to take measurements, give high accuracy with a magnifying lens and crosshairs on the mouse. To make the application familiar to the operator, a zoom box will replace the mouse. A scaled image will also be displayed in place of the scanned image, which is too large to be displayed on conventional raster screens. The images are also too large to be held in memory simultaneously, hence a disk caching mechanism was written to hold the image data required for the zoom box. The result is a single window holding the

[1] The *widgets* are the push buttons, canvases, scrollbars and other elements of a modern graphical software application's user interface. A *widget library* is a traditional C library having functions that create, manipulate and interpret these graphical elements.

entire image at low resolution, and a zoom box window holding a section of the image at high resolution. This enables the user to see any part of the image in its full detail.

Multiple images can also be loaded at the same time. The user is able to place markers on the image to indicate selected features. When the zoom box is at maximum zoom these can be placed with sub-pixel precision. Once markers are placed, the user can then execute some function — such as fitting a line or calibrating the camera — with a button click.

Once an operation has been performed, the output goes to a built-in text editor. For instance, fitting a cylinder will type out a `cylinder` command, with its dimensions, to the editor. The user then has the option of rendering those commands in 3D and rotating and scaling the result. This allows a scene to be gradually built in a virtual world from stereo pairs. The commands typed out in the editor are the results of each triangulation or geometric fit, as well as a log of the session.

Because the text has a very simple format, a conversion program for converting to other file formats would be easy to write. Files could, for example, be converted using MicroStation[2] BASIC for use with the MicroStation CAD software.

A particular desktop setup along with its calibrations and data can also be loaded from, and saved to, disk for reference.

[2]MicroStation is a registered trademark of Bentley Systems, Incorporated.

List of Tables

3 Camera Calibration

In expressing the camera geometry, a more physically representative system was used than given in [8]. Their method still results in non-linear equations (albeit equations that are less computationally expensive), and with only four to eight calibration points, there seems little reason to depart from a basic non-linear optimisation.

The relation between image coordinates and camera coordinates, $\mathbf{X}_c = (x_c,\ y_c,\ z_c)$, is described in Fig. 3.1:

$$\begin{aligned} u &= -f\frac{x_c}{z_c} \\ v &= -\sigma f\frac{y_c}{z_c} \quad . \end{aligned}$$

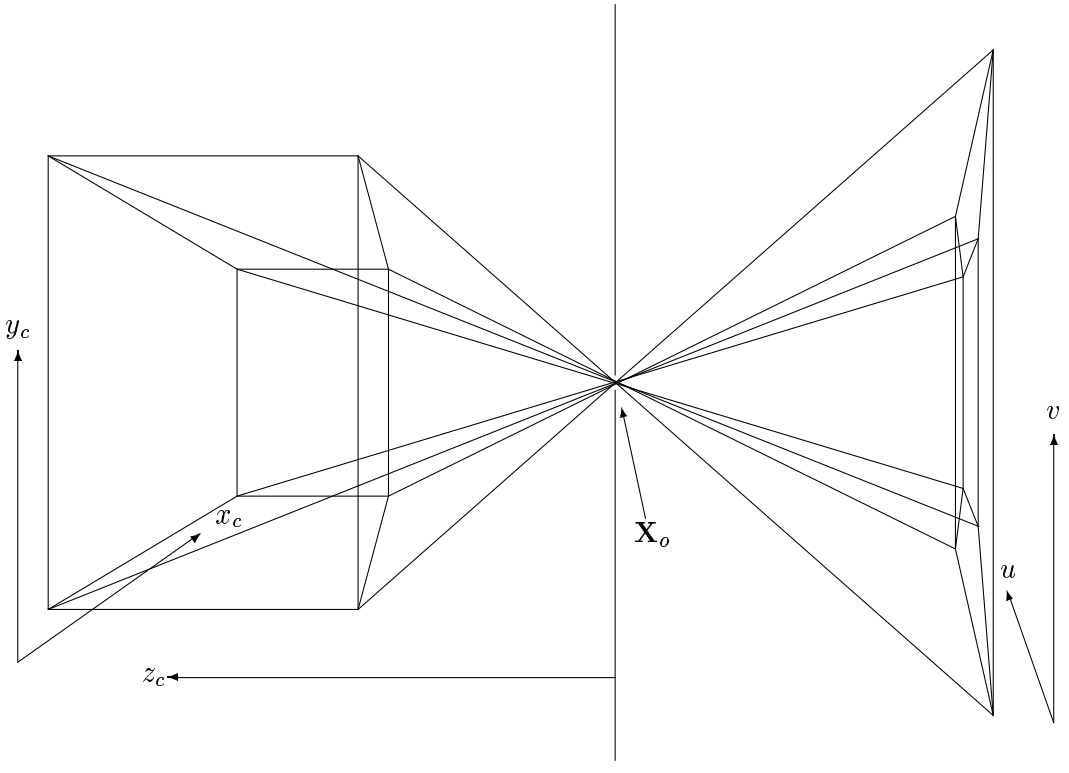


Figure 3.1: Pinhole camera model.

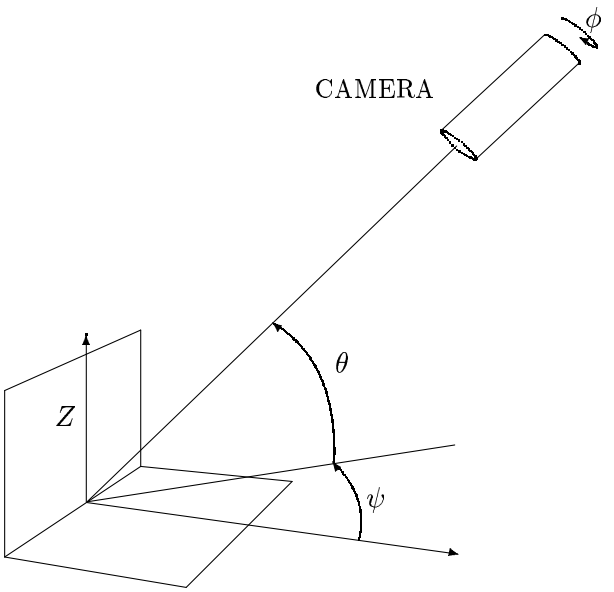


Figure 3.2: Camera geometry.

To obtain the camera coordinates from the real world image coordinates, requires a translation to the camera centre, and a rotation by the camera rotation matrix. The relation between image coordinates and the 3D position is given by,

$$\begin{aligned} u &= x - x_o = -f\frac{(\mathbf{X}-\mathbf{X}_o)^T\mathbf{m_x}}{(\mathbf{X}-\mathbf{X}_o)^T\mathbf{m_s}} \\ v &= y - y_o = -\sigma f\frac{(\mathbf{X}-\mathbf{X}_o)^T\mathbf{m_y}}{(\mathbf{X}-\mathbf{X}_o)^T\mathbf{m_s}} \quad , \end{aligned} \tag{3.1}$$

where $\mathbf{X} = [X \;\; Y \;\; Z]^T$ is the physical position of the coordinate in question; $\mathbf{X}_o = [X_o \;\; Y_o \;\; Z_o]^T$ is the camera centre; (x, y) is the projected position in the image; (x_o, y_o) is the image centre; f is the focal length; and σ is the aspect ratio. $\mathbf{M} = [\mathbf{m_x} \;\; \mathbf{m_s} \;\; \mathbf{m_y}]^T$ is the rotation matrix of the particular camera view based on the Eulerian angles ϕ, θ and ψ.

The choice of rotation matrix and 3D world axis was made to match the land-surveying problem at hand: with no rotation (i.e. ϕ, θ and ψ zero) the y image coordinate is aligned with the Z axis (up), the x image coordinate is aligned with the X axis (east), and the "z" image depth coordinate is aligned with the Y axis (north). The rotation order is (see Fig. 3.2): first by the azimuthal angle ψ, then by the elevation angle θ, and finally by the rotation (about the optical axis) angle ϕ.

To solve, given sufficient calibration points, their 3D positions and their 2D projections, requires a total of eight parameters to be found. These can be reduced to 5 parameters by linear solution of the camera centre, $\mathbf{X}_o$. Rearranging equation 3.1,

$$u_i(\mathbf{X}_i - \mathbf{X}_o)^T\mathbf{m_s} = -f(\mathbf{X}_i - \mathbf{X}_o)^T\mathbf{m_x}$$

$$v_i(\mathbf{X}_i - \mathbf{X}_o)^T\mathbf{m_s} \quad = \quad -f(\mathbf{X}_i - \mathbf{X}_o)^T\mathbf{m_y} \quad for\ i\ =\ 1\ \ldots\ N \quad , \tag{3.2}$$

for N calibration points[1]. For a least squares solution, this rearrangement does not change the problem significantly, since the denominator is large and positive (the calibration points being far in front of the camera). $\mathbf{X}_o$ can be separated out:

$$\begin{aligned} \mathbf{X}_o^T(u_i\mathbf{m_s} + f\mathbf{m_x}) &= \mathbf{X}_i^T(u_i\mathbf{m_s} + f\mathbf{m_x}) \\ \mathbf{X}_o^T(v_i\mathbf{m_s} + f\mathbf{m_y}) &= \mathbf{X}_i^T(v_i\mathbf{m_s} + f\mathbf{m_y}) \quad . \end{aligned}$$

Rewriting in matrix form,

$$A_{(2N\times 3)}\mathbf{X}_o \quad = \quad U_{(2N\times 1)} \quad , \tag{3.3}$$

or,

$$A^TA\mathbf{X}_o \quad = \quad A^TU_{(2N\times 1)}$$

and finally, since A^TA is square,

$$\mathbf{X}_o \quad = \quad (A^TA)^{-1}A^TU \tag{3.4}$$

gives the least squares solution. Hence $\mathbf{X}_o$ can be found given the five remaining parameters. Iterative optimisation in only five variables is reasonably inexpensive. The downhill simplex method of Nelder and Mead [11] was used for this. For an initial guess of the rotation angles, a coarse, exhaustive search is performed over ϕ, θ and ψ. The focal length f is estimated as $-4 \times max(|(x_i, y_i) - (x_j, y_j)|)$. This over-estimation of f will force $\mathbf{X}_o$ to be computed further away from the calibration point cluster, ensuring that there is no chance of a calibration point finding its way behind the camera. The aspect ratio distortion σ is initially set to zero (i.e. no distortion). An iterative optimisation is then performed, provided N is not less than four, which matches the eight degrees of freedom of the system.

Calibration using the above method takes less than a second using an i486 100MHz computer. The algorithm also checks if less (x, y) coordinates were given than $\mathbf{X}$ coordinates. (This might occur when several views use the same set of calibration points, but where one of the views has some calibration points obscured.) In this case it attempts all combinations of omitted points, keeping the one with the lowest error, and also tries various combinations of further omissions if the user suspects that a point is erroneous. This feature was added when AECI explained that calibration points were often recorded erroneously, forcing them to omit selected points for a better calibration.

[1] A least squares solution of equation 3.2 is in fact *not* the same as that of equation 3.1. The reason why the solution is similar is because the denominator of 3.1 is the distance of the calibration point from the camera centre and is always positive and of a similar order.

4 Triangulation and Geometric Fits

4.1 Triangulation

A linear solution to the triangulation problem follows similarly to the previous section.

$$\begin{aligned} u_j(\mathbf{X}-\mathbf{X}_{oj})^T\mathbf{m}_{\mathbf{s}j} &= -f_j(\mathbf{X}-\mathbf{X}_{oj})^T\mathbf{m}_{\mathbf{x}j} \\ v_j(\mathbf{X}-\mathbf{X}_{oj})^T\mathbf{m}_{\mathbf{s}j} &= -f_j(\mathbf{X}-\mathbf{X}_{oj})^T\mathbf{m}_{\mathbf{y}_j} \quad for\ j\ =\ 1\ \ldots\ N \quad , \end{aligned} \tag{4.1}$$

where X is the 3D coordinate we are seeking; (u_j, v_j) are the projections in each view j; and $\mathbf{M}_j = [\mathbf{m}_{\mathbf{x}j}\ \ \mathbf{m}_{\mathbf{s}j}\ \ \mathbf{m}_{\mathbf{y}_j}]^T$ is the rotation matrix for each view. This can be rearranged, as before, into matrix form, giving

$$\mathbf{X} = (B^TB)^{-1}B^TV \quad . \tag{4.2}$$

The number of views, N, is not limited, but must obviously not be less than two.

4.2 Geometric Fitting in 3D

In the following analyses, *fitting* refers to finding the best visual fit, and not necessarily the best statistical fit. Hence rigorous statistical methods are not necessary, and errors are always calculated normal to the closest point on the line or surface. The 3D fits will occur on points only after they have been triangulated. Although some of these are old favourites of numerical analysis, re-invention is probably easier than trying to find original references.

4.2.1 Fitting lines

Although analytic methods exist to fit arbitrary lines in two dimensions, for the case of three dimensional lines, a general non-linear optimisation is easier to implement.

The vector equation for a 3D line is,

$$\mathbf{r} \quad = \quad \mathbf{u}t + \mathbf{c} \tag{4.3}$$

where $\mathbf{r} = [x \quad y \quad z]^T$. The total error of N points, $\mathbf{r}_i$, from the line is,

$$e \quad = \quad \sum_{i=1}^{N} |\mathbf{u}t_i + \mathbf{c} - \mathbf{r_i}|^2 \tag{4.4}$$

where t_i is the closest point on the line to the point i. To find t_i we differentiate e,

$$\frac{\partial e}{\partial t_i} \quad = \quad 2\mathbf{u}^T(\mathbf{u}t_i + \mathbf{c} - \mathbf{r}_i) \quad = \quad 0 \tag{4.5}$$

giving

$$t_i \quad = \quad \frac{\mathbf{u}^T(\mathbf{c} - \mathbf{r}_i)}{\mathbf{u}^T\mathbf{u}} \tag{4.6}$$

The line can now be found by non-linear least squares by substituting for t_i,

$$\min_{[\mathbf{u}\ \mathbf{c}]\in\mathbf{R}^6} \sum_{i=1}^{N} |\frac{\mathbf{u}^T(\mathbf{c} - \mathbf{r}_i)}{\mathbf{u}^T\mathbf{u}}\mathbf{u} + \mathbf{c} - \mathbf{r}_i|^2 \quad . \tag{4.7}$$

Because a 3D line has only four degrees of freedom, t_i will be set to 0 and t_N will be set to 1. $|u|$ will be the length of the line if point 1 and N are the beginning and end points. We can now also reduce the problem from six to three variables by solving for the offset vector $\mathbf{c}$. The formulation requires straight-forward differentiation of 4.7, resulting in:

$$\mathbf{c} \quad = \quad \frac{1}{N}A^{-1}\mathbf{s} \quad , \tag{4.8}$$

where,

$$A \quad = \quad \begin{bmatrix} (y^2+z^2)+2u^2 & -xy & -xz \\ -xy & (x^2+z^2)+2u^2 & -yz \\ -xz & -yz & (x^2+y^2)+2u^2 \end{bmatrix} ,$$

and,

$$\mathbf{s} \quad = \quad \begin{bmatrix} xy\sum y_i + xz\sum z_i - (y^2+z^2)\sum x_i - u^2(x_N + x_1 - x) \\ xy\sum x_i + yz\sum z_i - (x^2+z^2)\sum y_i - u^2(y_N + y_1 - y) \\ xz\sum x_i + yz\sum y_i - (x^2+y^2)\sum z_i - u^2(z_N + z_1 - z) \end{bmatrix} ,$$

where $u = |\mathbf{u}|$ and the summations of $\mathbf{s}$ exclude $i = 1$ and $i = N$. With $\mathbf{c}$ known, the error can be calculated by expanding 4.7. We finally optimise in the remaining three variables of $\mathbf{u}$. Since $\mathbf{u}$ would normally be a unit vector with two degrees of freedom, this method is inefficient by one variable. This is a worthwhile sacrifice, however, since special cases of orientation to the coordinate axis need not be accounted for.

4.2.2 Fitting 3D lines from 2D projections

Fitting a 3D line to two 2D projections is useful because it avoids having to triangulate exact points. In this case two 2D lines are first fitted in the image plane as follows.

The error of point i, e_i, is calculated normal to the line (see Fig. 4.1). The total error of N points, (u_i, v_i), with the line $y = mx + c$ is,

$$\sum_{i=1}^{N} |\mathbf{e}_i|^2 = \sum_{i=1}^{N} \frac{(c + mu_i - v_i)^2}{m^2 + 1} \tag{4.9}$$

Minimising by differentiation with respect to m and c gives two equations which can be solved explicitly,

$$\begin{aligned} m &= \frac{-B \pm \sqrt{B^2 + 4A^2}}{2A} \\ c &= \frac{1}{N}\sum v_i - \frac{m}{N}\sum u_i \end{aligned} \tag{4.10}$$

where,
$A = \frac{1}{N}\left(\sum u_i\right)\left(\sum v_i\right) - \sum u_i v_i$
and,
$B = \frac{1}{N}\left(\sum u_i\right)^2 - \frac{1}{N}\left(\sum v_i\right)^2 - \sum u_i^2 + \sum v_i^2$.
The two solutions of (4.10) represent local maximum and global minimum errors and their lines will be normal to each other. The correct line can be found by calculating the error for each solution.

It is apparent that $A \to 0$ as the x or y values of the points group toward a constant. This results in numerical errors when evaluating (4.10) for lines close to vertical or horizontal. Thus for say $|B| > 1000|A|$ a normal least squares fit is performed instead

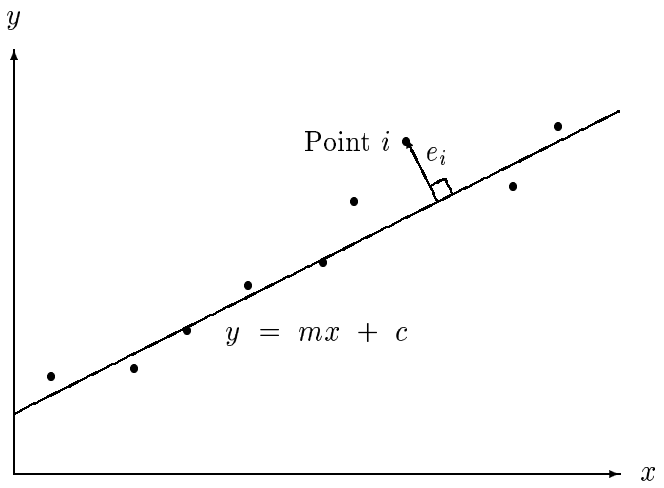


Figure 4.1: Calculation of error when fitting a line.

in the appropriate direction. The inconsistency is small since the error vectors are almost normal to the line under these conditions.

Having obtained equations for the 2D projections, the 3D line can be found. The two image lines can be thought to represent planes going through the camera centres, viewed edge on. The planes can be found by substituting $y = mx + c$ into 3.1. This gives two equations in X, Y and Z. The interception of these planes gives the required line.

This method is satisfactory when the projected lines are normal to the baseline of the cameras. When they are close to parallel, the resulting planes will be also, and small errors in the image lines will be exaggerated. This can give very inaccurate results.

4.2.3 Fitting 3D cylinders from their edges

Provided that the cylinder is not parallel to the camera baseline, the problem of finding the cylinder centreline and radius from only its edge lines is constrained and tractable. Accuracy will depend on the angle of the cylinder axis with the baseline, since the problem becomes less constrained the closer these are to parallel.

The cylinder is identified in two projected views by a number of points (marked by the user) along both horizons of the cylinder. Four lines are fitted to these four sets of points. The horizons of the cylinder are symmetrical about the projected axis of the cylinder, and hence taking the *mean* of a horizon pair gives the correct projection of the cylinder axis. Here, the *mean* is defined as a line that goes through the vanishing point of the two horizon lines, and also having a gradient (in degrees) that is the mean of the gradient of the horizon lines. The cylinder axis can then be found from its projections as in the previous section.

Once the axis is found, the radius can be calculated. To accurately account for perspective, the observed radius has to be calculated by artificial projection of the horizon lines. Trying to calculate the radius, for example, by mere measurement of the fitted lines with the axis would not give an accurate result.

Here, the radius will be found by a coarse to fine search given an initial estimate. A convenient initial estimate can be found by locating the 3D line that projects to one of the horizons. This line is imaginary, and is roughly a radius distant from the axis.

The procedure is this: using an estimate of the radius, the horizons are calculated. These lines are compared to the fitted lines, the estimate is adjusted accordingly, and

the loop is repeated until the radius converges.

The horizon is calculated as follows (see Fig. 4.2) : Consider two vectors $\mathbf{r}_1$ and $\mathbf{r}_2$ which are normal to the axis and to each other, and a point $\mathbf{p}$ on the cylinder axis. The locus in the projected view of a point on the cylinder as it is rotated round the cylinder by an angle θ is,

$$\begin{aligned} x &= -f\frac{(\mathbf{p}-\mathbf{X}_o+\mathbf{r}_1 sin(\theta)+\mathbf{r}_2 cos(\theta))^T\mathbf{m_x}}{(\mathbf{p}-\mathbf{X}_o+\mathbf{r}_1 sin(\theta)+\mathbf{r}_2 cos(\theta))^T\mathbf{m_s}} \\ y &= -f\frac{(\mathbf{p}-\mathbf{X}_o+\mathbf{r}_1 sin(\theta)+\mathbf{r}_2 cos(\theta))^T\mathbf{m_y}}{(\mathbf{p}-\mathbf{X}_o+\mathbf{r}_1 sin(\theta)+\mathbf{r}_2 cos(\theta))^T\mathbf{m_s}} \quad . \end{aligned} \tag{4.11}$$

If (u, v) is a vector normal to the projected axis of the cylinder, then the point is on the horizon when $l = (x, y)\bullet(u, v)$ is a maximum. Simplifying gives,

$$l = \frac{a + b\ sin(\theta) + c\ cos(\theta)}{d + e\ sin(\theta) + f\ cos(\theta)} \quad , \tag{4.12}$$

where a, b etc. are constants. Differentiating and equating to zero gives,

$$(bd - ae)cos(\theta) + (af - cd)sin(\theta) + bf - ce = 0 \tag{4.13}$$

This quadratic has two solutions, θ_1 and θ_2, at either horizon. The projected lines can now be found by substituting back into equation 4.11. These two angles will have a difference of roughly, *but not exactly*, π.

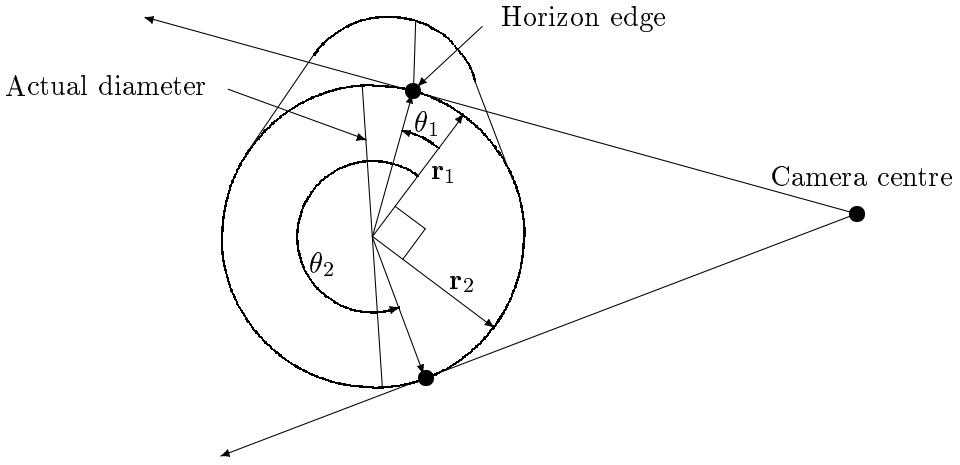


Figure 4.2: Geometry of a projected cylinder.

4.2.4 Fitting circles and planes

Fitting a plane is done with a straight-forward four variable optimisation. To fit a circle, a plane is first fitted to the points and then a 2D circle is fitted in the plane. [3] demonstrates a transformation which linearises the circle fitting problem. The problem is first formulated as follows for a general n dimensional circle,

$$\min_{\mathbf{x}\in\mathbf{R}^n, r\in\mathbf{R}^+} \sum_{i=1}^{N} \left(r^2 - |\mathbf{x} - \mathbf{u_i}|^2\right)^2 , \tag{4.14}$$

where $\mathbf{x}$ is the circle centre, r is the radius, and $\mathbf{u_i}$ are the image points. Alternatively,

$$\min_{\mathbf{x}\in\mathbf{R}^n, r\in\mathbf{R}^+} \sum_{i=1}^{N} \left(r^2 - \mathbf{x}^T\mathbf{x} + 2\mathbf{x}^T\mathbf{u_i} - \mathbf{u_i}^T\mathbf{u_i}\right)^2 , \tag{4.15}$$

which can be linearised by making the substitution

$$\mathbf{y} = \begin{bmatrix} 2\mathbf{x} \\ r^2 - \mathbf{x}^T\mathbf{x} \end{bmatrix}, \qquad \mathbf{v_i} = \begin{bmatrix} \mathbf{u_i} \\ 1 \end{bmatrix}. \tag{4.16}$$

The problem (4.15) becomes

$$\min_{\mathbf{y}\in\mathbf{R}^{n+1}} \sum_{i=1}^{N} \left(\mathbf{u_i}^T\mathbf{u_i} - \mathbf{v_i}^T\mathbf{y}\right)^2 . \tag{4.17}$$

If $n = 2$ this can be solved by differentiation, and simplifies elegantly to

$$\mathbf{y} = \begin{bmatrix} \sum u_i^2 & \sum u_i v_i & \sum u_i \\ \sum u_i v_i & \sum v_i^2 & \sum v_i \\ \sum u_i & \sum v_i & N \end{bmatrix}^{-1} \begin{bmatrix} \sum u_i^3 + \sum u_i v_i^2 \\ \sum v_i^3 + \sum u_i^2 v_i \\ \sum u_i^2 + \sum v_i^2 \end{bmatrix}, \tag{4.18}$$

where $[u_i \;\; v_i]^T = \mathbf{u_i}$. Hence the circle centre is $(\frac{1}{2}y_1, \frac{1}{2}y_2)$ and $r = \sqrt{y_3 + \mathbf{x}^T\mathbf{x}}$ from (4.16).

4.2.5 Fitting ellipses and cylinders

An extension of this theme is developed by the author to fit ellipses and cylinders. Fitting an ellipse (ellipsoid or hyper-ellipsoid) in n dimensions is analogous to fitting a circle in $n+1$ dimensions that appears as an ellipse when projected from an oblique n-dimensional plane. For the two dimensional case substituting $\mathbf{u_i} = [u_i \;\; v_i \;\; au_i + bv_i]^T$ and $\mathbf{x} = [x_c \;\; y_c \;\; ax_c + by_c]^T$ (a and b being the gradients of the plane) in 4.15 enables us to fit a circle for $n = 3$. This gives

$$\min_{[x_c\, y_c\, a\, b]\in\mathbf{R}^4, r\in\mathbf{R}^+} \sum_{i=1}^{N} \left(x_c^2 + y_c^2 - r^2 + (ax_c + by_c)^2 - 2[u_i x_c + v_i y_c\right.$$

$$+ (au_i + bv_i)(ax_c + by_c)] + u_i^2 + v_i^2 + (au_i + bv_i)^2\Big)^2 . \quad (4.19)$$

The substitution $\zeta = x_c^2 + y_c^2 - r^2$ is then used to simplify further manipulations. Solving by differentiation gives five non-linear equations in x, y, ζ, a and b. The advantage of the method is that the summations can now be separated out and hence need only be calculated once. These are of the form

$$S_{pq} = \sum_{i=1}^{N} u_i^p v_i^q \qquad p, q = 0, 1, 2, \ldots \qquad p + q \leq 4. \quad (4.20)$$

Three of the equations contain only linear terms in x, y and ζ, thus reducing the non-linear problem to two variables a and b. The formulation is straight forward, but extremely tedious, and hence best left to a symbolic algebraic manipulator. The source code can be consulted for the results.

Once a and b are known, the orientation of the minor axis of the ellipse can be deduced from $\arctan(\frac{b}{a})$. The gradient of the projection plane in this direction is $\sqrt{a^2 + b^2}$ and hence the minor diameter is $2r/\sqrt{a^2 + b^2 + 1}$. The major diameter is just $2r$.

To fit a cylinder is similar. If we rotate all the points so that the cylinder stands upright, then we can ignore the vertical component of each point and fit a 2D circle as in the previous section. We then optimise for the best rotation, which will just be a direction vector with two degrees of freedom. The optimisation may be restarted with different initial guesses — say direction vectors parallel to the different axis — where the best result is kept. A cylinder is thus fitted by iteration in only two variables.

4.2.6 Fitting circles from their projected ellipses

Fitting circles from a number of point correspondences has already been discussed. To fit a circle from its projected ellipses will allow the user to find a circle without having to locate specific corresponding points, or even to have the same number of markers in each view. If the ellipses in two views are represented by N and M points respectively, then the total error of an estimated circle with its projected ellipses is,

$$e = \left(\sum_{i=1}^{N}(x_i - f(\theta_i))^2 + \sum_{i=1}^{N}(y_i - g(\theta_i))^2\right)_{left} \quad (4.21)$$

$$+ \quad \left(\sum_{i=1}^{M}(x_i - f(\theta_i))^2 + \sum_{i=1}^{M}(y_i - g(\theta_i))^2\right)_{right} , \quad (4.22)$$

where $f(\theta)$ and $g(\theta)$ are the right hand sides of equation 4.11 ($\mathbf{p}$ being the circle

centre), and θ_i is that which gives the minimum error for each point. A full optimisation of the direction vector, radius and circle centre (six variables), can be performed over an optimisation of the θ for each point. This is most inefficient, but gives an accurate result. For a complete analysis of such problems as projection of ellipses from three dimensions to camera views, see [18] or [19].

4.3 Extracting Surface Grids

An additional feature is provided to extract arbitrary surfaces from a scene. If a surface has $N \times M$ array of grid points, the user can mark these points in sequence, scanning across and then down the surface. The user is then prompted for the value of N and M, and the program triangulates all corresponding points at once. The use of this feature to digitise 3D surfaces is demonstrated in the Section 6.

5 Overview of Program Design and Flow

This section describes the primary functions of the software application. A complete program listing is given in Appendix B.

The designed widget library functions in parallel with XLib instead of on top of it. This gives the XLib programmer greater flexibility than traditional widget libraries, while still allowing him the conveniences of higher level widget commands.

The widget library provides substitute commands for basic XLib functions. It is object orientated in nature, although is written in standard C. The central function is `CNextEvent()` which checks the XLib event queue and does all necessary graphics processing of widgets (such as expose handling) as well as calling user specified call-back functions. Fig. 5.1 shows the overall flow chart of the application.

A sufficient description of the program flow will now be given to allow extensions to be added. The primary buttons are created in the file `display.c` where call-back functions are assigned to each button. The call-back functions are named `cb_xxx()` and are in `callback.c`. A number of images may be loaded for display, zooming and marker allocation by the user. The call-back function will be called when the particular button is pushed, whereupon the following sequence of nested functions will be called.

1. A function `output_xxx()` will be called in `imagefit.c`.
2. This function will call a function `fit_xxx()` in the same file, which will look up the markers from the respective images and call the low level geometric fitting function in the file `fitline.c`.
3. The `fit_xxx()` function will return a generic geometric object structure of the type `Object` to the `output_xxx()` function.
4. The generic function `output_object()` will then be called in the file `output.c`

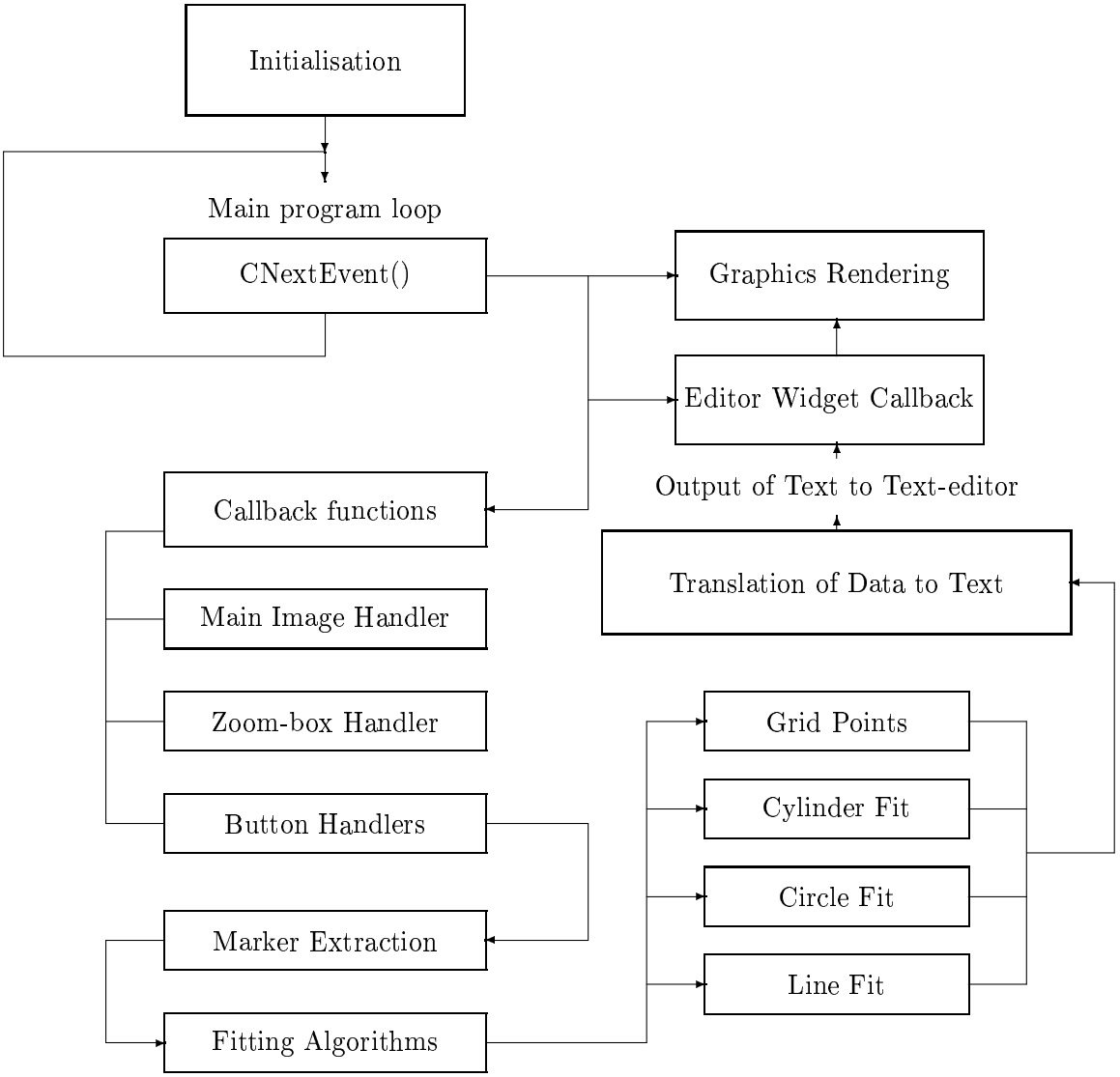


Figure 5.1: Overall program flowchart

to translate the object's floating point data into text, and output that into the text-editor widget.

5. The `output_xxx()` function will return to the call-back function, which will then return to the application.

The above functions make use of a set of marker subroutines which extract and manipulate sets of markers from the images. See the file `marker.c`.

Each image window also has its own buttons which activate utilities. These buttons share a common function over all images that are loaded onto the desktop. The function must therefore first check which image it corresponds to (using the command `set_current_from_pointer()`) before pursuing with any operations on the markers.

The code is extensively commented to give further details. Fig 5.2 gives a view of the desktop environment.

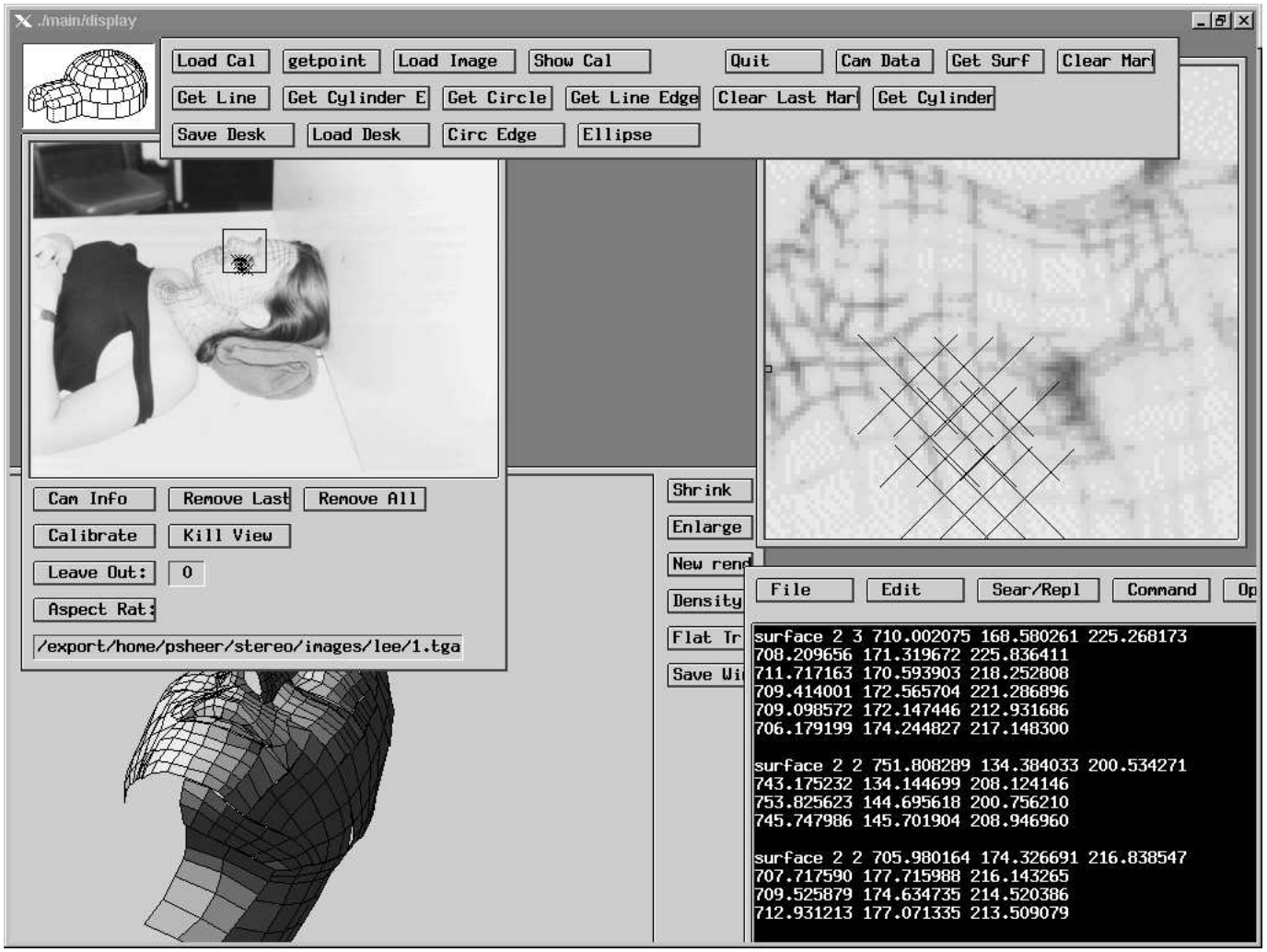


Figure 5.2: Captured view of the desktop during a sample run.

6 Demonstration Run of the Program

A model of a human face was constructed from ten photographs. Figs 6.1 are black and white shots taken with an ordinary 35mm reflex camera and scanned to 2500 $\times$ 1800 pixels. Nine calibration points were used although less than six could be clearly seen in most cases. Here, the grid lines merely help find correspondences, and each sub surface has to have all of its grid points located sequentially with mouse clicks. Fig. 6.2 shows the reconstructed surface.

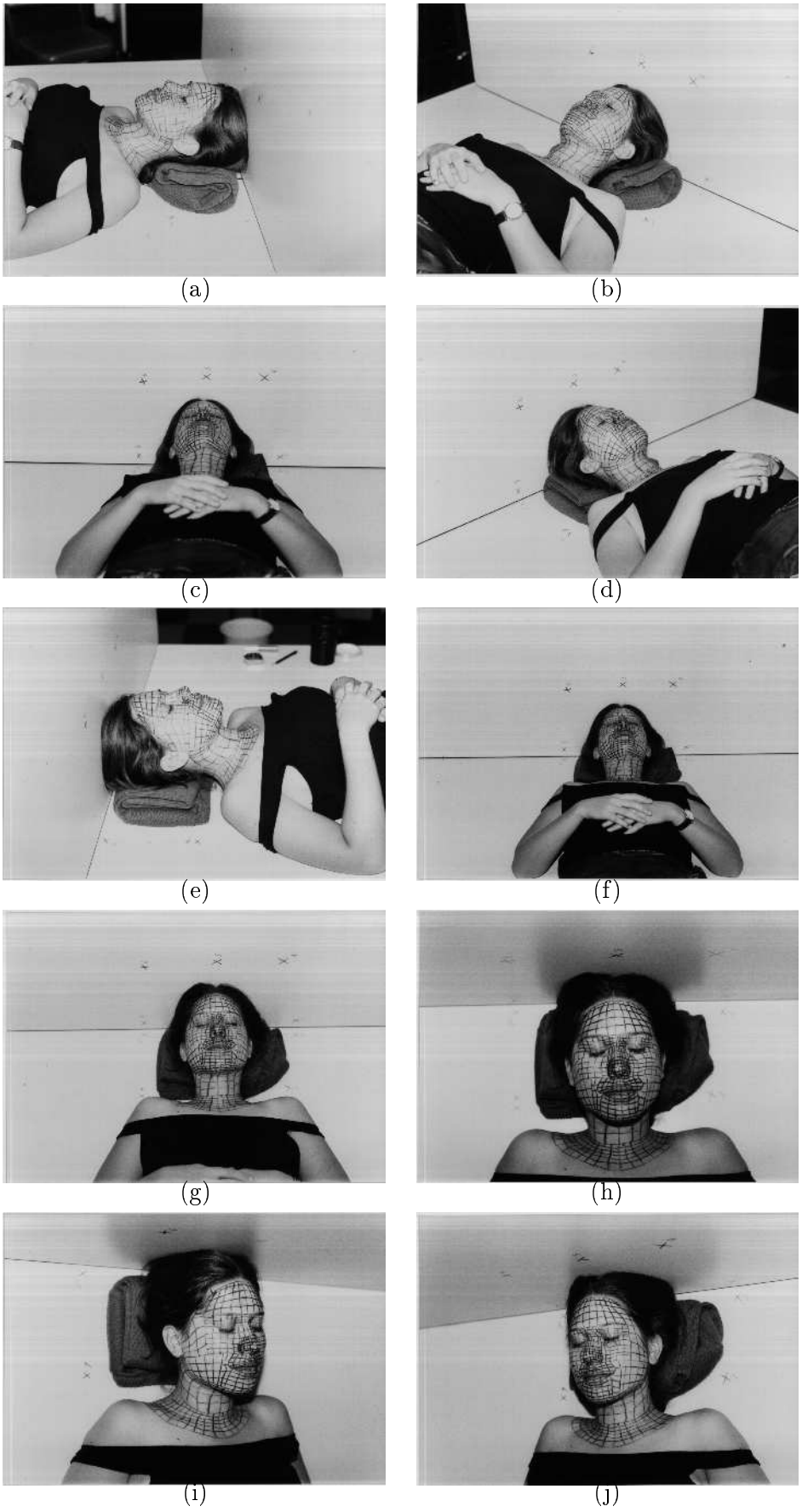


Figure 6.1: Photographs taken of "Lee" at different azimuthal and elevation angles.

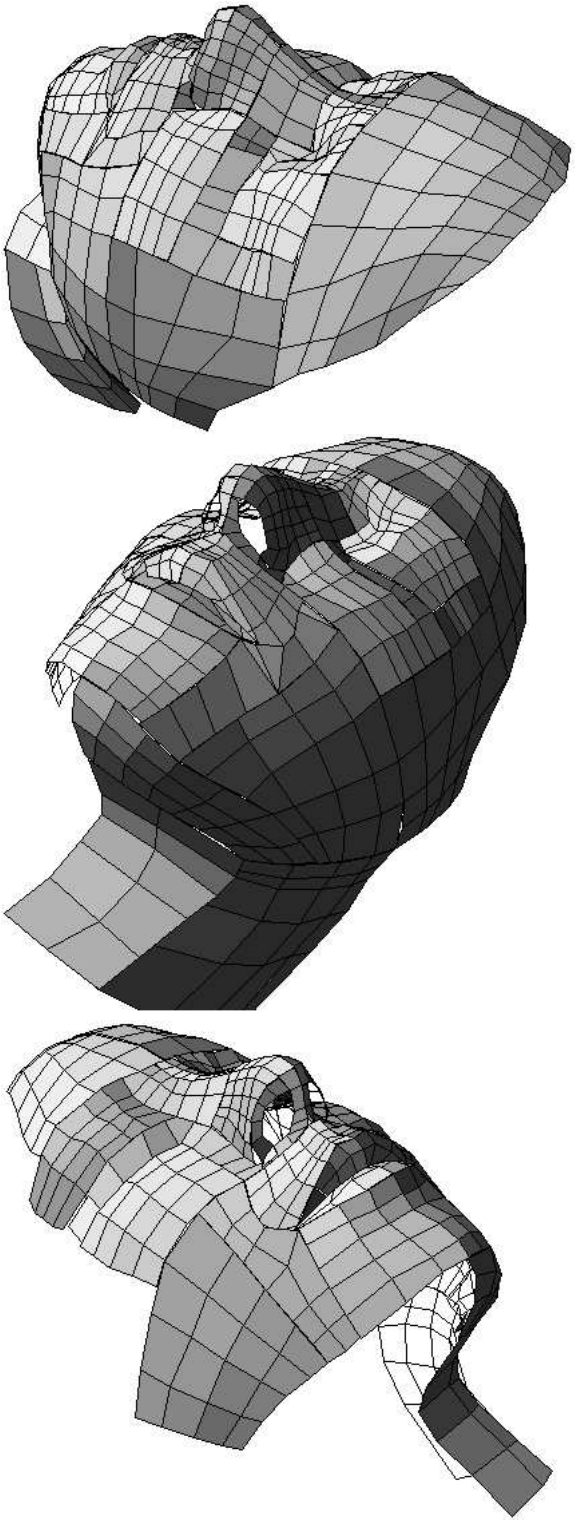


Figure 6.2: 3D rendering of "Lee" triangulated data at various azimuthal angles.

7 Calibration Testing

Figs 7.1 through 7.5 are five arbitrary photographs of a plant survey. The scanned images have a resolution of approximately 2000×1500 pixels. Each calibration point is represented by a numbered surveyor's "X". Table 7 shows calibration results of these images. The given SSD error is defined as,

$$SSD \quad = \quad \sum_i |\mathbf{p}_{actual_i} - \mathbf{p}_{projected_i}|^2 \quad , \tag{7.1}$$

representing the difference between the actual pixel measurements and the projected estimate of each calibration point. The "Aspect Ratio" field refers to a re-calibration that includes the aspect ratio parameter (less 100%), to give an indication if any aspect ratio change was incurred from the scanner. For the given results the aspect ratio was fixed at unity.

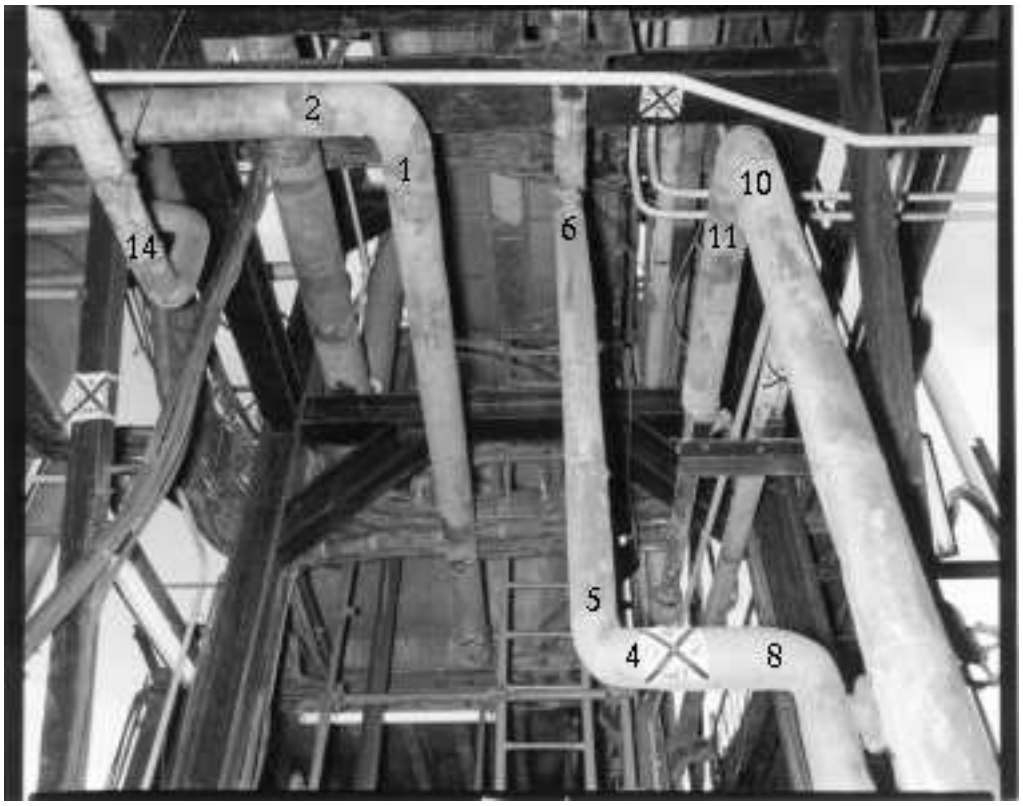
For all of these photographs, one or more calibration points were given that could not be found in the image. In every case, the algorithm omitted the relevant calibration point. The number of calibration points given in the table excludes the omitted points.

(b)

(a)

Figure 7.1: Photo-pair "A" of plant survey.

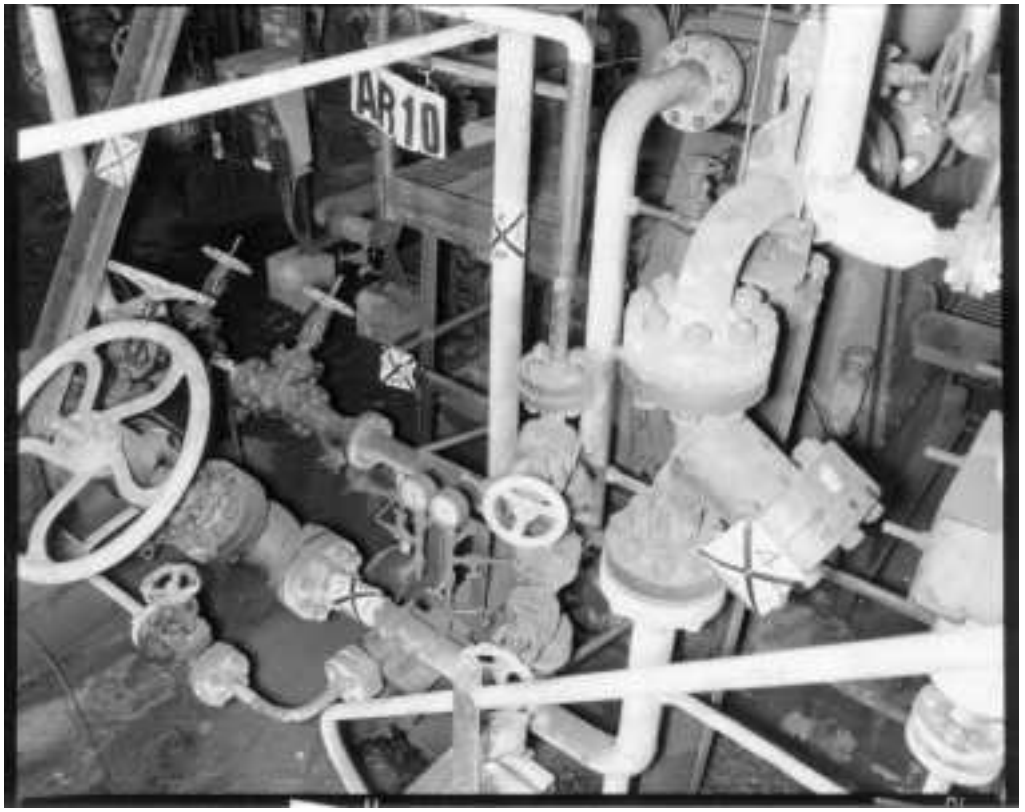
(b)



(a)

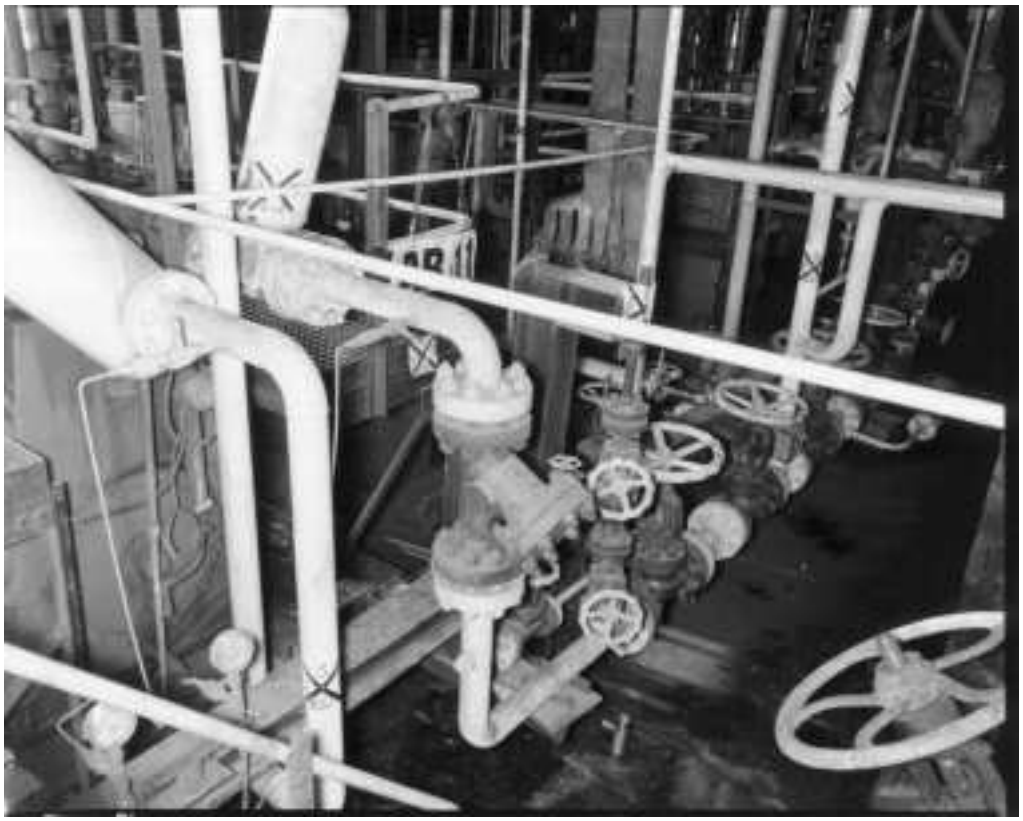
Figure 7.2: Photo-pair "B" of plant survey.

(b)



(a)

Figure 7.3: Photo-pair "C" of plant survey.



(b)

(a)

Figure 7.4: Photo-pair "D" of plant survey.

(b)

(a)

Figure 7.5: Photo-pair "E" of plant survey.

Table 7.1: Calibration results of plant photographs

	Pair A		Pair B		Pair C	
	Left	Right	Left	Right	Left	Right
ϕ [^{o}]	-1.05	-2.53	0.38	-3.40	3.03	-0.65
θ [^{o}]	-4.78	-2.31	-43.14	-38.44	32.64	26.54
ψ [^{o}]	-61.23	-34.44	87.48	114.25	54.91	66.36
$-f$ [$pixels$]	1821.0	1779.5	1777.4	1776.0	1704.6	1726.4
X [mm]	186277.3	184834.5	188546.7	188545.2	186677.9	185976.8
Y [mm]	-214015.1	-214227.4	-217036.2	-215319.9	-220850.0	-220542.7
Z [mm]	1631141.6	1631157.9	1631119.6	1631286.9	1632484.0	1632481.8
SSD [$pixels^2$]	0.8	6.3	50.5	21.7	30.5	13.0
Asp. Ratio. [%]	0.23	0.34	0.71	0.33	1.05	1.54
No. Pts.	4	5	5	5	5	5

	Pair D		Pair E	
	Left	Right	Left	Right
ϕ [^{o}]	2.72	3.19	1.54	0.41
θ [^{o}]	33.53	23.13	-3.81	-7.24
ψ [^{o}]	-17.29	-34.78	-117.61	-133.93
$-f$ [$pixels$]	1759.9	1770.8	1773.0	1816.7
X [mm]	186870.0	187750.1	190144.8	189620.8
Y [mm]	-221190.0	-221120.8	-216373.8	-214921.0
Z [mm]	1632877.3	1632681.7	1631216.2	1621235.8
SSD [$pixels^2$]	17.0	24.7	51.5	9.9
Asp. Ratio. [%]	0.79	0.76	0.19	0.30
No. Pts.	5	5	5	5

8 Accuracy

Several cylinders were fitted from the pipe horizons in Fig. 7.2. The results are tabulated in Table 8.1. Since AECI is unable to fit cylinders using horizon edge lines, cylinder data are rather obtained by extracting a circle from either end of the cylinder; the circle centre being on the axis of the cylinder. The cylinder centreline can be drawn by joining these points. The numbers in Fig. 7.2 show roughly the position of these extractions. Here we are concerned with the normal distance (s in Table 8.1) of these axis points to the axis that has been found using our cylinder (horizon-edge) fitting technique, as well as the difference in diameters of each cylinder.

These results compare poorly with with AECI's measurements. This is probably because of inaccuracy in the surveying of the calibration points — since the SSD pixel errors obtained here were much larger than those of the "Lee" images. AECI's method is also far more prone to error than the present method, since our fit effectively takes a statistical mean of the entire edge of the cylinder. Their method may however be preferable where pipes are not exactly round. (Our package does support both methods of triangulation, however.) AECI's claim of 1mm accuracy (per meter range) is probably only relative accuracy between local points in the view.

Hence, such a comparison does not provide an indication of the accuracy of the package. Ironically, accuracy for these surveys is actually not critical, since the CAD drawings are created from scratch — the drawing lines being adjusted to align properly regardless of the stereo measurements. The diameters too, are irrelevant, since the actual diameter of interest is concealed by a thick layer of insulation material.

For a statistical measure of the absolute accuracy, 48 points shown in Fig. 8.1 were located and photographed. The points are spaced 150mm apart and are on average about 2.5 meters from the camera. Their absolute physical position is known to within 1mm.

Photographs were taken with black and white, 35mm ISO-400 film, f/stop 11, with an ordinary reflex camera at 20^o intervals (see Fig. 8.2 and Fig. 8.3). These were

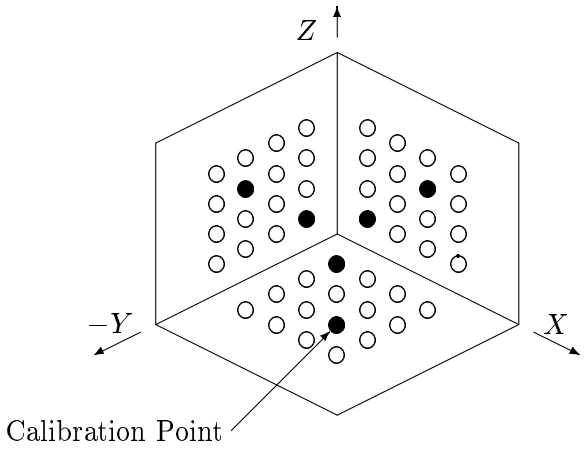


Figure 8.1: Schematic of artificial setup.

scanned at 300 dpi to give images of 1900 × 1700 pixels.

Table 8.1: Results of pipe cylinder fits

Cylinder	Start point [mm]	End point [mm]	diameter [mm]
1	(190340, -216607, 1633891)	(191707, -216608, 1633891)	168
2	(189836, -215394, 1633559)	(189881, -216332, 1633650)	146
4-8	(190523, -217188, 1632225)	(190514, -217567, 1632217)	144
5-6	(190553, -217154, 1632350)	(190550, -217158, 1634009)	116
10	(190017, -217658, 1632913)	(190026, -217661, 1633509)	140
11	(191105, -217655, 1633649)	(190160, -217656, 1633664)	141
14	(189499, -215856, 1633364)	(190296, -215857, 1633400)	99

Circle	Centre point [mm]	radius [mm]	s [mm]
1	(190299.2, -216620.7, 1633798.9)	178	93.3
2	(190072.7, -216256.5, 1633903.6)	173	323
4	(190533.0, -217237.0, 1632216.0)	150	13.7
8	(190591.8, -217571.8, 1632247.0)	164	83.3
5	(190561.4, -217166.5, 1632329.9)	126	15.0
6	(190554.6, -217164.3, 1633989.3)	125	7.5
10	(190030.6, -217676.3, 1633411.4)	171	17.2
11	(190265.1, -217663.5, 1633686.2)	171	25.3
14	(190173.1, -215863.0, 1633388.4)	113	8.6

Table 8.2: Calibration results of artificial test

		(a)	(b)	(c)	(d)
ϕ	[^{o}]	1.26	1.25	0.07	0.13
θ	[^{o}]	15.04	15.25	15.24	14.88
ψ	[^{o}]	-10.08	-32.12	-52.55	-73.99
$-f$	$[pixels]$	3936.0	3864.0	4163.5	4117.8
X	$[mm]$	695.5	1439.5	2253.0	2591.3
Y	$[mm]$	-2493.5	-2119.3	-1780.3	-934.2
Z	$[mm]$	859.1	839.8	917.7	880.7
SSD	$[pixels^2]$	0.6	4.7	9.3	0.2
Asp. Ratio.	[%]	0.30	0.27	0.53	0.31
No. Pts.		6	6	6	6

An unusual shape of calibration point marker is chosen here. These markers have the advantage of giving a precise point even in the presence of affine distortions. Close-ups of the scanned image are shown in Fig. 8.4.

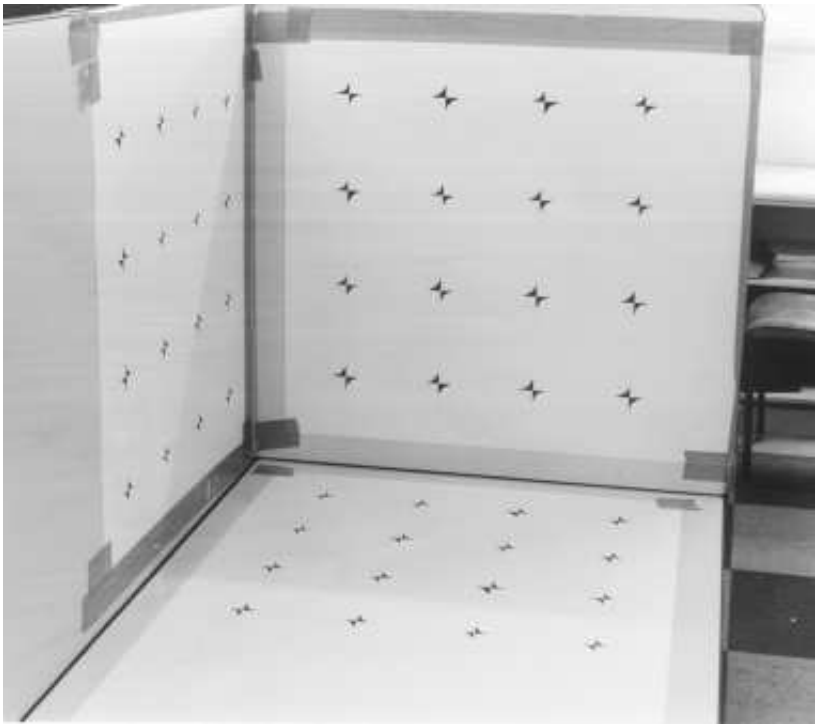
The results of the camera calibrations are shown in Table 8.2. Here the aspect ratio is included in the calibration, since it is relatively consistent over the four views.

The 48 points were triangulated using different combinations of views, including a range of different baselines. Table. 8.3 shows the standard deviation of the triangulation error over all points, the *error* being the Euclidean distance to the nominal position. Also of interest are the mean X, Y and Z errors of each group of sixteen points — groups 1, 2 and 3 represent the right, left and bottom planes respectively.

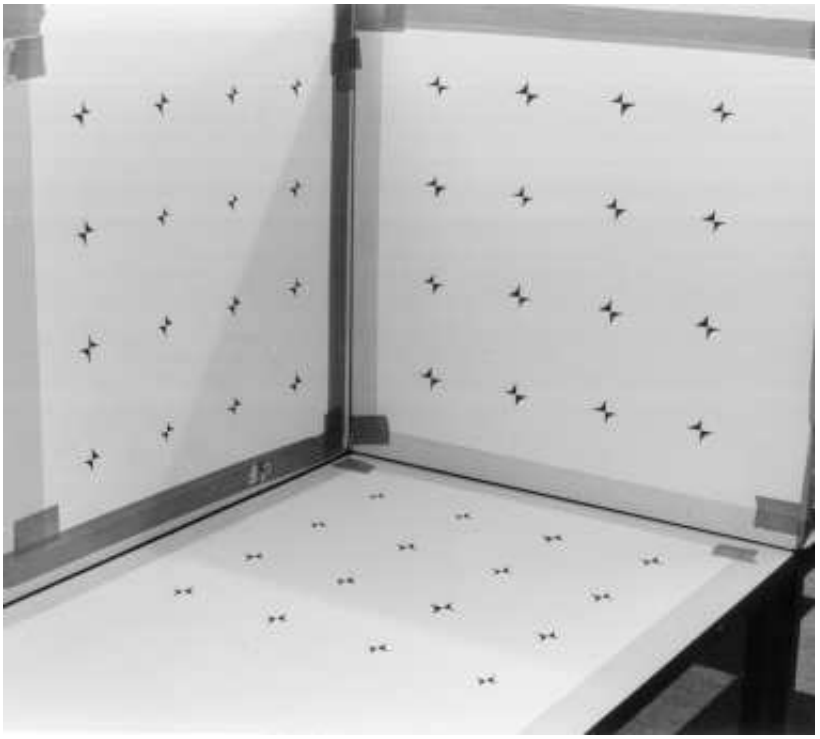
The circle fit was then tested with this setup. If the corner points are omitted, the outer-most points of each group inscribe a circle. Fitting a circle to the eight points gives the three circles of Table 8.4. (Note that the nominal radius is $\sqrt{75^2 + 225^2} = 237.17mm$ from the nominal point spacing.)

To test the line-fitting algorithm, lines were fitted to some of the vertical rows of points. The results of these are shown in Table 8.5.

The calibrations above were performed only once. If the error was high, the calibration may have been repeated to check that the user was not at fault. To test the variation of the calibration error with different users, ten users were given the image in Fig 8.2(a) to calibrate. The results of these are shown in Table 8.6. None of the users had previously used such software.

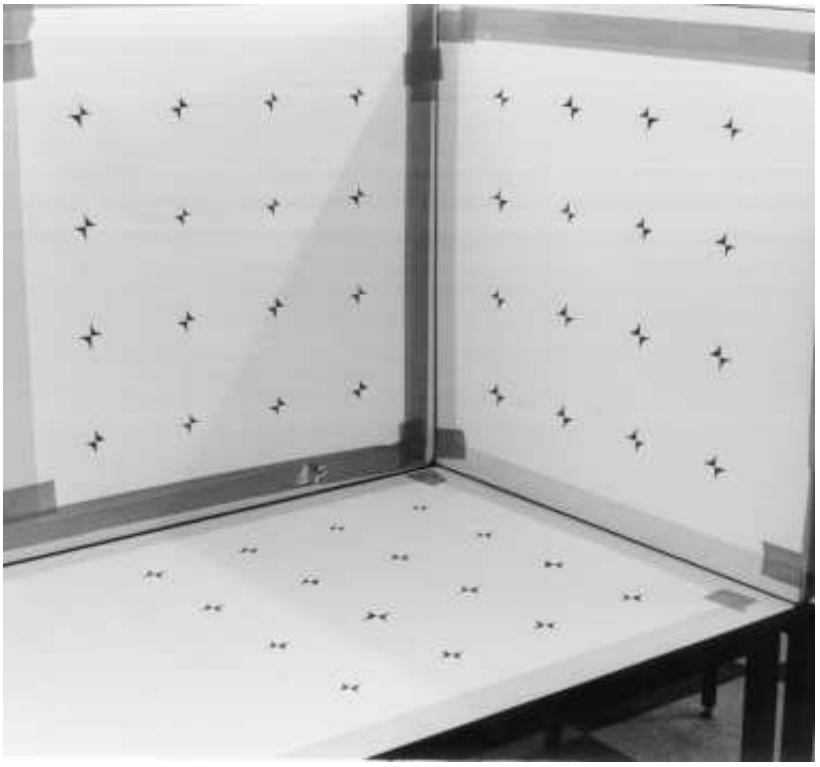


(a)

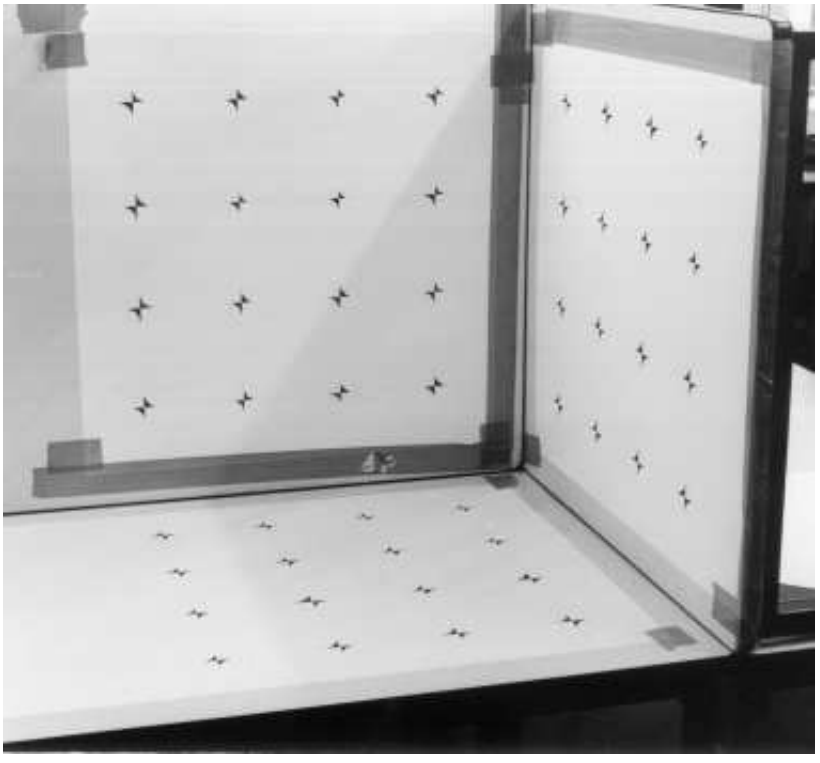


(b)

Figure 8.2: Photographs (a) and (b) taken of an artificial setup for accuracy testing.



(c)



(d)

Figure 8.3: Photographs (c) and (d) taken of an artificial setup for accuracy testing.

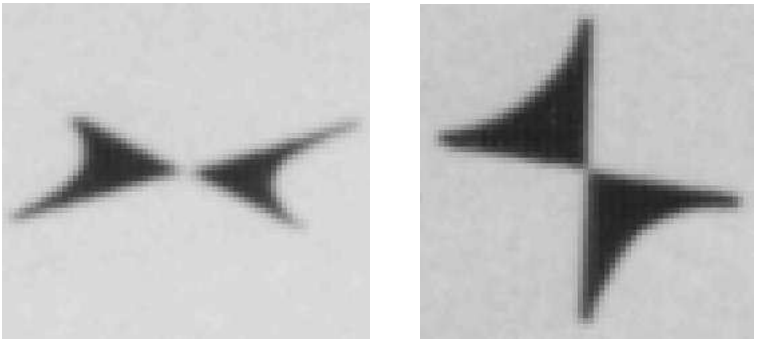


Figure 8.4: Close-up of two calibration points, showing the level of detail in the scan.

Table 8.3: Triangulation results of artificial test

Using view (a) and (d)
Baseline 2.5m

SD $[mm]$	0.95		
$[mm]$	$\overline{X}$	$\overline{Y}$	$\overline{Z}$
group 1	0.064	0.113	-0.302
group 2	-0.078	0.086	-0.212
group 3	-0.026	0.237	-0.045

Using view (a) and (b)
Baseline 0.8m

SD $[mm]$	2.16		
$[mm]$	$\overline{X}$	$\overline{Y}$	$\overline{Z}$
group 1	0.011	-0.657	-0.322
group 2	0.104	-0.541	0.160
group 3	-0.113	0.333	-0.218

Using view (a), (b) and (c)
Total baseline 1.7m

SD $[mm]$	1.64		
$[mm]$	$\overline{X}$	$\overline{Y}$	$\overline{Z}$
group 1	0.151	-0.548	-0.439
group 2	0.111	-0.390	0.104
group 3	-0.038	0.470	-0.116

Using all views
Total baseline 2.5m

SD $[mm]$	1.05		
$[mm]$	$\overline{X}$	$\overline{Y}$	$\overline{Z}$
group 1	-0.046	-0.035	-0.476
group 2	-0.148	-0.060	-0.053
group 3	0.016	0.310	-0.040

Table 8.4: Results of 3-D circle fits

Measurent				Error			
Circle centre			Radius	Circle centre			Radius
X [mm]	Y [mm]	Z [mm]	r [mm]	X [mm]	Y [mm]	Z [mm]	r [mm]
375.00	0.20	374.47	237.35	0.00	0.20	-0.53	0.18
-0.06	-375.22	374.73	237.03	-0.06	-0.22	-0.27	-0.14
375.07	-374.93	-0.02	237.74	0.07	0.07	-0.02	0.57

Table 8.5: Results of line fits

3D fit of triangulated points		Error	
Direction [mm]	Start [mm]	Direction [mm]	Start [mm]
(-0.69, 0.73, 449.05)	(0.03, -150.31, 149.82)	(-0.69, 0.73, -0.95)	(0.03, -0.31, -0.18)
(0.10, 0.31, -449.05)	(-0.27, -299.64, 599.09)	(0.10, 0.31, 0.95)	(-0.27, 0.36, -0.91)
(-0.50, -1.00, 449.50)	(0.13, -449.23, 149.90)	(-0.50, -1.00, -0.50)	(0.13, 0.77, -0.1)
(-0.31, 0.66, -448.90)	(0.45, -600.51, 599.18)	(-0.31, 0.66, 1.10)	(0.45, -0.51, -0.82)

Fit from 2D lines		Error	
Direction [mm]	Start [mm]	Direction [mm]	Start [mm]
(0.14, -0.08, 449.37)	(-0.40, -149.93, 149.76)	(0.14, -0.08, -0.63)	(-0.40, 0.07, -0.24)
(0.06, 0.28, -449.25)	(-0.25, -299.62, 599.24)	(0.06, 0.28, 0.75)	(-0.25, 0.38, -0.76)
(-0.37, 0.60, -449.84)	(0.06, -450.03, 599.71)	(-0.37, 0.60, 0.16)	(0.06, -0.03, -0.29)
(-0.32, 0.73, -449.64)	(0.45, -600.54, 599.68)	(-0.32, 0.73, 0.36)	(0.45, -0.54, -0.32)

Table 8.6: Calibration results of ten users

User	X $[mm]$	Y $[mm]$	Z $[mm]$	SSD
1	695.9	-2493.1	859.9	0.66
2	696.4	-2494.6	859.9	0.59
3	696.3	-2496.6	859.1	0.46
4	697.1	-2503.5	863.5	0.56
5	699.8	-2512.9	865.0	0.18
6	694.9	-2491.2	858.4	0.79
7	695.9	-2498.6	859.9	0.73
8	697.6	-2499.5	859.8	1.32
9	697.2	-2502.1	861.9	0.68
10	699.7	-2503.7	861.1	0.34
Mean	697.1	2499.6	860.9	
SD	1.6	6.3	2.2	

9 Discussion

9.1 Geometric Fits

For some of the geometric fits described, finding the general geometric parameters is only the first step. In the case of the line fit, for example, the line start point and end point are needed to complete its description. Thus each object is clipped by finding the outermost points in the fit.

When fitting the cylinder from its edge points, the user marks points along both of the cylinder's horizon edges. However only one set of points is given in each view; the grouping of each set of points into two lines is purely visual and is not explicitly defined. The program must divide the set into two lines which it does as follows:

1. A line is fitted through the first two points.
2. If the next point, P_i, is further than some constant from the line (normal distance) then the set is divided at i and the procedure ends.
3. A line is fitted through all points up to and including the point P_i.
4. Repeat from 2.

This algorithm will divide the points correctly, provided the user places all the markers of one line before marking the other line.

9.2 Usage

The reconstruction of the "Lee" images, provided an opportunity to thoroughly test the graphical interface. This type of usage is painstaking and requires considerable

planning to efficiently segment the surface into rectangular subsections. The accuracy of the reconstruction is also poor because the calibration points were not properly measured. To align sub-surfaces properly required the coding of an additional algorithm which "zips" together surface edges that are in close proximity. This gave the seamless reconstructions of Fig. 6.2. The rendering algorithm is also not able to calculate the normals at the surface edges properly — which requires searching for the neighbouring surfaces and finding the mean normal at the joins. In spite of these shortcomings, the rendition is an excellent representation of the model's face.

9.3 Calibration

Altogether 24 images have been calibrated in this project, and in all cases, the algorithm converged. The calibration takes less than a second using our i486 100 MHz computer. If, however, the program has to search for omitted calibration points, more time is needed. Note that the program requires no initial estimate, and converges correctly regardless of the scale or orientation of the scene. Different users were also asked to calibrate the same image in order to account for the human factor in calibration point location. The small variation between different, totally inexperienced, users shows that the calibration is not appreciably user dependent.

The calibration results of the plant photographs vary greatly in accuracy. An SSD of 50 corresponds to a mean error of 3 pixels per calibration point. From the calibration model, this is an error of approximately 10 mm at 6 meters from the camera — roughly the same error obtained from triangulations in the scene. By comparison, the maximum SSD for the artificial test was 9 (or 1.2 pixels per calibration point), corresponding to an average error of only 1.8 mm at 6 m. The artificial test also used six calibration points instead of five, which naturally tends to increase the error, and hence favours the conclusion that the plant calibration points are the main source of inaccuracy.

The artificial test also does not show the camera parameters to be consistent through the four different calibrations (Table 8.2). The variation can be accounted for in the movements of the camera tripod. The camera focus, however, was not changed and hence the focal length should have remained the same. It is however possible that the images were not scaled consistently during the developing process, accounting for a varying focal length. There also is not any way of accounting for the larger SSD error of calibration (c).

In the plant calibrations, the aspect ratio varied randomly. In the artificial test, however, aspect ratio was around 0.3%, except for one view where it was 0.5%. This view also had the highest calibration error. Because of the consistency of the aspect ratio, it was left in as a calibration parameter. As suspected at the outset, the scanner is the most probable cause of the aspect ratio change since it is the only apparatus that is directionally sensitive. The error due to the aspect ratio can be calculated as $1700/2 \times 0.3/100 = 2.5$ pixels[1]; giving an indication of its importance.

In comparison to the method of [8], the calibration algorithm developed in this project is easier to implement, although very inefficient where a large number of calibration points are present. It is also easy to add in an extra parameter (such as skew factor, or radial distortion), whereas [8] will require construction of a different set of equations that have to be reducible to polynomial form.

9.4 Accuracy

The artificial test was chosen to provide an absolute indicator of the accuracy of the system. The test points are spread over most of the image to allow any spatial distortion to be accounted for. In addition, test points are not completely contained within the cluster of calibration points, because most real scenes require measurement outside the cluster, where the accuracy diminishes.

The data show increasing accuracy with the number of views used, as well as with the width of the baseline. Triangulation using all four views shows a disappointing SD equal to that using only views (a) and (d) (see Table 8.3). This is explained by the poor calibration of view (c). The small mean X, Y and Z error for the different planes of points suggest that no global offsets were in effect, and that the errors were mostly random.

The SD of 1 mm translates into an error of 1.6 pixels in the view ($f = 4000$, camera distance $= 2.5m$). This indicates that sub-pixel location of points may be unnecessary.

The geometric fits on the test points have even better accuracy. When fitting a geometric object, it is statistically expected that the accuracy will increase with the number of points used in the fit. Hence the triangulation results of individual points are a worst case experiment.

It was originally proposed that the images be scanned at a resolution higher than 300

[1] 1700 is the image height in pixels, and the aspect ratio is defined with respect to the y coordinate.

dpi. This is obviously unnecessary, since other sources of inaccuracy are overriding. In particular the surveying of calibration points is the principle source of inaccuracy for plant surveys.

9.5 Replacement of Existing Facilities

By computerising the entire surveying task, the user now has far more flexibility than with existing, instrument based stereo-metrology. The principle advantage is that images or data can be transfered, retrieved and modified at no expense to the operator. The package is also extensible, allowing further features for geometric extraction to be added. The existing features provide for less (possibly none) hand calculations, and greater automation.

10 Conclusions

The software package allows high resolution stereo photographs to be manipulated within a graphical environment, and is efficient and easy to use. Photographs may be calibrated with optional inclusion of an aspect ratio distortion parameter. In a variety of test conditions, calibration is shown always to converge regardless of the camera position, and without requiring an initial estimate. The calibration is also shown to be relatively consistent across a number of different users. The package allows the user to extract various geometric primitives from images, and hence make three-dimensional measurements of the scene. The primitives presently supported are lines, circles, cylinders and grid meshes. These may be extracted by individual point correspondences, or from their geometric projections (for example a line may be fitted only from its projections in two views). Calibrations, photographs, and measurements may be stored and retrieved by the software, making the maintenance of extensive CAD models easier. The combination of features in the software — and the inherent flexibility of digital media — make it a significant advancement over systems where photographs are physically measured, which the software is intended to replace.

In an artificially constructed test, accuracy was shown to be 1 mm (standard deviation) at a camera distance of 2.5 m. This translates to 1:1200 of the width of the view. For digitising the photographs, a scanner was used. As expected at the outset, the scanner did not have an aspect ratio of precisely unity — the distortion was about 3:1000. The quoted accuracy is also achieved without the aid of expensive photographic equipment. The lens distortion of the ordinary 35 mm camera used is obviously not an important factor.

11 Future Work

The package requires a few minor additions to be ready for use. At the moment, the output of the user's measurements is in the form of a text file of a non-standard format. This needs to be converted into common formats of commercial CAD packages to be useful. Other tools may also be added, such as edge detection, fitting of further primitives and so forth. Additional lens distortion parameters are easy to add and experiments should be performed to determine precisely the extent of image distortion. It is quite likely that the system could be tuned to give sub-pixel accuracy.

REFERENCES

[1] Alvertos, N., Brzakovic, D. and Gonzalez, R. Camera geometries for image matching in 3-D machine vision, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 9, pp. 897–909, September 1989.

[2] Barkakati, N, *X Window System Programming*, SAMS, 1994.

[3] Coope, I.D. Circle fitting by linear and nonlinear least squares, *Journal of Optimization Theory and Applications*, Vol. 76, no. 2, Feb. 1993, pp. 381–388.

[4] Dhond, U. R. and Aggarwal, J. K. Structure from stereo—a review, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 19, No. 6, November 1989.

[5] Faig, W. Calibration of close-range photogrammetry systems: mathematical formulation, *Photogrammetric Eng. and Remote Sensing*, Vol. 41, pp. 1479–1486, 1975.

[6] Freeman, H. *Machine Vision for Three-dimensional Scenes*, Academic Press, Inc., 1990.

[7] Faugeras, O. *Three-dimensional Machine Vision, A Geometric Viewpoint*, The MIT Press, Cambridge, Massachusetts, London, England, 1993.

[8] Grosky, W. I. and Tamburino, L. A. A unified approach to the linear camera calibration problem, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, no. 7, pp. 663–671, July 1990.

[9] Kanade, T. *Three Dimensional Machine Vision*, Kluwer Academic Publisher, 1987

[10] Lenz, R. K. and Tsai, R. Y. Techniques for calibrating the scale factor and image centre for high accuracy 3-D machine vision metrology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, no. 5, pp. 713–720, September 1988.

[11] Press, W. H. *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1992.

[12] Rosenfeld, A. *Techniques for 3-D Machine Perception*, Elsevier Science Publishers B. V., 1986.

[13] Shirai, Y. *Three Dimensional Computer Vision*, Springer Verlag, 1987.

[14] Sobel, I. On calibrating computer controlled cameras for percieving 3-D scenes, *Artificial Intell.* Vol. 5, pp. 185–198, 1974.

[15] Tsai, R. An efficient and accurate camera calibration technique for 3-D machine vision, *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, Miami Beach, FL, pp. 364–374, June 1986.

[16] Foley, J. D. and Van Dam, A. *Fundamentals of Computer Graphics*, Addison-Wesley, Philippines, 1982.

[17] Wei, G. and De Ma, S. Implicit and explicit camera calibration: theory and experiments, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 469–481, May 1994.

[18] Wu, J., *IRLM: An Intensity Image System for Object Recognition, Localization, and Motion Recovery*, PhD. Thesis, Department of Electrical Engineering, University of Alberta, Edmonton, Alberta, Canada, 1988.

[19] Wu, J., Rink, R. E., Caelli, T. M. and Gourishankar, V. Recovery of 3D locations and motions of rigid objects through camera imaging (extended Kalman filter approach), *International Journal of Computer Vision*, 1988.

APPENDIX A Features of the Software

The Linux operating system was used to code the software for the following reasons:

1. It is extremely stable,
2. Powerful programming tools are freely available,
3. Many Free-ware example programs, libraries and other tools are available to refer to.
4. It conforms to well established standards.
5. It is available at no cost.
6. Programs can be made to be configurable for compilation under any other Unix system.
7. It runs on a PC which is inexpensive.
8. Unix has inherent modularity which enables the application to run client-server style if desired.

The package reads and converts AECI's generic calibration data files, which are given in north-south, east-west, up-down format. These are converted to positive and negative (X, Y, Z) coordinates.

The rendering window allows the generated 3D scene to be scaled and rotated in real time. It also has a Gouraud shading option (see [16]) for giving realistic 3D shadowing effects, and correctly performs hidden surface removal.

Each of the operations requires the user to lay markers in the images. Markers may also be added, deleted and moved arbitrarily.

Automake version 1.0[1] was used to configure the program. Although this is the first public release of Automake, it worked with virtually no problems and generates Makefiles according to the GNU makefile standards. Autoconf version 2.12[2] was also used. This should allow compilation on any Unix system that supports ANSI style header files. It has already been tested with Solaris, Irix and Linux.

[1] Available from `ftp://sunsite.unc.edu/pub/gnu`

[2] Available from same.

APPENDIX B C Language Source Code

The following is the partial source code for the application, including all the mathematical routines discussed in this thesis. Source code for the widget library is not included, neither are the configuration and make files. The entire C source code is over 26000 lines and 500 kilobytes. Some GNU C utilities make up a further 200 kilobytes of code. The complete alpha release of the application can be obtained from sunsite.unc.edu.

```
3:3:     calibrate.c - camera calibration minimisation algorithms
433:2:   callback.c - main callback routines from button presses
757:3:   camera.c - undestort camera coordinates
866:2:   desktop.c - save and restore the desktop to *.dsk files
1029:4:  display.c - main() function
1418:2:  displaycam.c - draw a list of the camera data
1559:2:  fitline.c - line, circle, ellipse, plane, and cylinder fitting routines
3102:2:  hugeimage.c - widget to draw the tiff file, zoom box and do caching of image data
3692:2:  imagefit.c - commands to fit objects to desktop markers
4383:2:  imagehandler.c - handles events from the view windows
4535:2:  join.c - for a number of grid surfaces, this zips up edges of one
4536:3:           surface that are close to an adjacent surface
5038:3:  loadcalfile.c - loads a calibration file into the desktop structure
5135:3:  marker.c - marker handling and finding real pointer positions in images
5560:2:  matrix.c - a matrix manipulation library
6745:3:  output.c - this outputs an object (line, circle, etc) to the editor
6894:3:  picsetup.c - setup and destroy pictures and views
7110:2:  savewindow.c - saves the rendered 3D scene as a targa file
7233:2:  simplex.c - simplex optimisation algorithm
```

```
1:1:
2:2:/*****************************************************************************/
3:3:/* calibrate.c - camera calibration minimisation algorithms                  */
4:4:/*****************************************************************************/
5:5:
6:6:#include <config.h>
7:7:#include "global.h"
8:8:#include <stdio.h>
9:9:#include <my_string.h>
10:10:#include <stdlib.h>
11:11:#include <stdarg.h>
12:12:#include <math.h>
13:13:
14:14:#include <X11/Xlib.h>
15:15:#include <X11/Xutil.h>
16:16:#include <X11/Xresource.h>
17:17:#include <X11/keysym.h>
18:18:
19:19:#include "stringtools.h"
20:20:#include "app_glob.c"
21:21:#include "dirtools.h"
22:22:
23:23:#include "coolwidget.h"
24:24:#include "matrix.h"
25:25:#include "calibrate.h"
```