

https://dspace.org/

A software assistant for manual stereo photometrology

Sheer, Paul

1997

http://hdl.handle.net/10539/22434

Sheer, Paul (1997) A software assistant for manual stereo photometrology, University of the Witwatersrand, Johannesburg, <http://wiredspace.wits.ac.za/handle/10539/22434>

Downloaded from a DSpace repository. https://dspace.org/

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, April 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 15 day of MAY 1997



Paul Sheer.

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.

Contents

Declaration i

Abstract ii

Contents iii

List of Figures vi

List of Tables viii

1 Introduction 1

1.1 Project Requirements 1

1.2 Background to Stereopsis 2

1.2.1 Obtaining stereo measurements 2

1.2.2 Camera calibration 3

1.3 Objectives 3

1.4 Requirements 4

1.5 Work Strategy and Project Plan 4

2 The Software Application 6

2.1 The Application under the X Window System 6

2.2 Emulation of the Existing Setup 6

3 Camera Calibration **8**

4 Triangulation and Geometric Fits **11**

4.1 Triangulation 11

4.2 Geometric Fitting in 3D 11

4.2.1 Fitting lines 11

4.2.2 Fitting 3D lines from 2D projections 13

4.2.3 Fitting 3D cylinders from their edges 14

4.2.4 Fitting circles and planes 16

4.2.5 Fitting ellipses and cylinders 16

4.2.6 Fitting circles from their projected ellipses 17

4.3 Extracting Surface Grids 18

5 Overview of Program Design and Flow **19**

6 Demonstration Run of the Program **23**

7 Calibration Testing **26**

8 Accuracy **33**

9 Discussion **40**

9.1 Geometric Fits 40

9.2 Usage 40

9.3 Calibration 41

9.4 Accuracy 42

9.5 Replacement of Existing Facilities 43

10 Conclusions 44

11 Future Work 45

REFERENCES 46

A Features of the Software 48

B C Language Source Code 50

List of Figures

3.1 Pinhole camera model. 8

3.2 Camera geometry. 9

4.1 Calculation of error when fitting a line. 13

4.2 Geometry of a projected cylinder. 15

5.1 Overall program flowchart 20

5.2 Captured view of the desktop during a sample run. 22

6.1 Photographs taken of "Lee" at different azimuthal and elevation angles. 24

6.2 3D rendering of "Lee" triangulated data at various azimuthal angles. 25

7.1 Photo-pair "A" of plant survey. 27

7.2 Photo-pair "B" of plant survey. 28

7.3 Photo-pair "C" of plant survey. 29

7.4 Photo-pair "D" of plant survey. 30

7.5 Photo-pair "E" of plant survey. 31

8.1 Schematic of artificial setup. 34

8.2 Photographs (a) and (b) taken of an artificial setup for accuracy testing. 36

8.3 Photographs (c) and (d) taken of an artificial setup for accuracy testing. 37

8.4 Close-up of two calibration points, showing the level of detail in the scan. 38

List of Tables

7.1 Calibration results of plant photographs 32

8.1 Results of pipe cylinder fits 34

8.2 Calibration results of artificial test 35

8.3 Triangulation results of artificial test 38

8.4 Results of 3-D circle fits 38

8.5 Results of line fits 39

8.6 Calibration results of ten users 39

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

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

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.

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.

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:

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

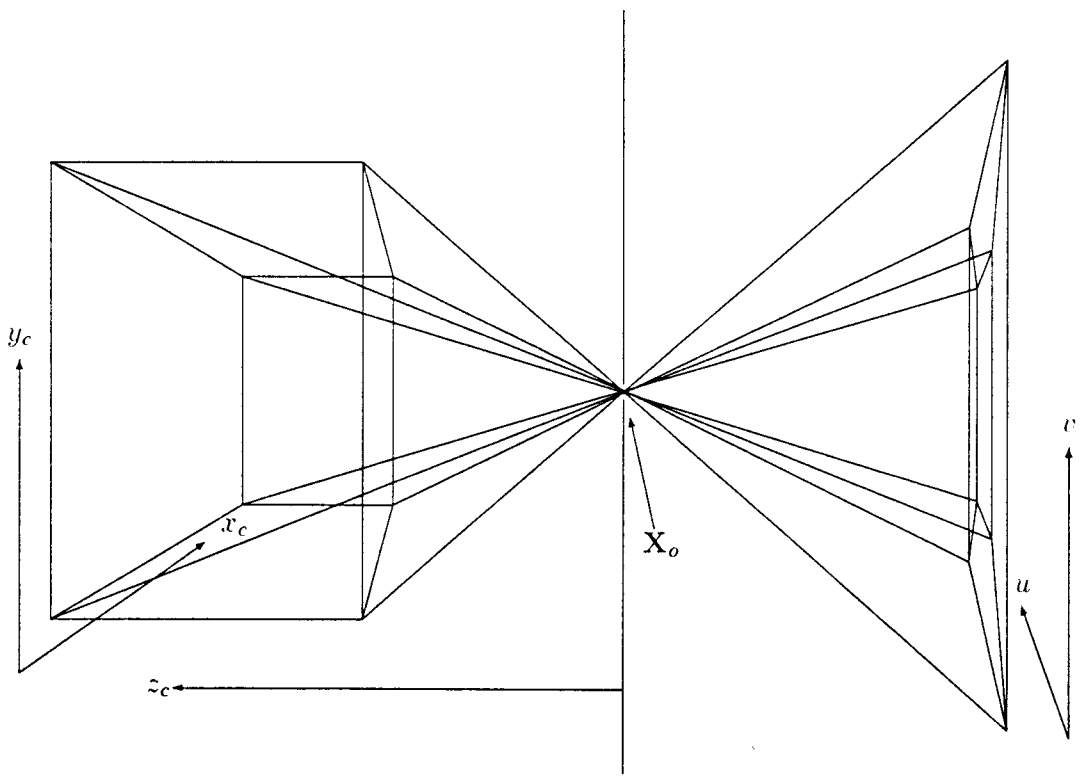


Figure 3.1: Pinhole camera model.

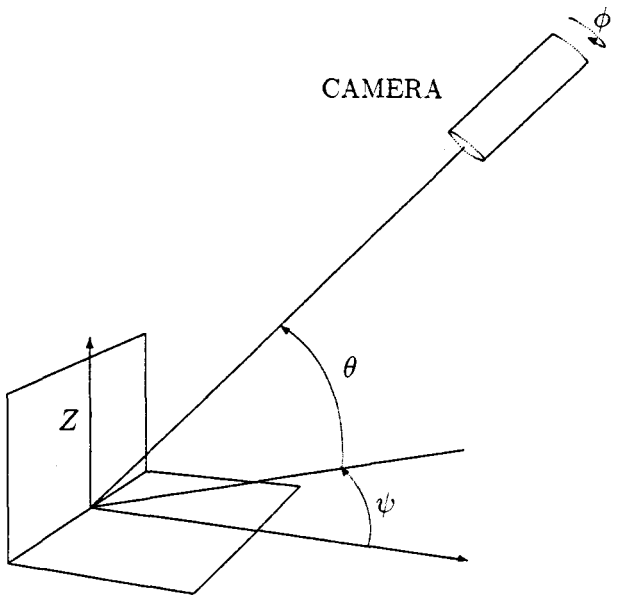


Figure 3.2: Camera geometry.

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,

$$u = x - x_o = -f\frac{(\mathbf{X}-\mathbf{X}_o)^T\mathbf{m_x}}{(\mathbf{X}-\mathbf{X}_o)^T\mathbf{m_s}}$$

$$v = y - y_o = -\sigma f\frac{(\mathbf{X}-\mathbf{X}_o)^T\mathbf{m_y}}{(\mathbf{X}-\mathbf{X}_o)^T\mathbf{m_s}} \quad . \tag{3.1}$$

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,

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

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

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,

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

or,

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

and finally, since A^TA is square,

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

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 overestimation 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.

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

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

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

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,

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

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

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

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

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

giving

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

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:

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

where,

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

and,

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

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,

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

where,

$A = \frac{1}{N}(\sum u_i)(\sum v_i) - \sum u_i v_i$

and,

$B = \frac{1}{N}(\sum u_i)^2 - \frac{1}{N}(\sum v_i)^2 - \sum u_i^2 + \sum v_i^2$.

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

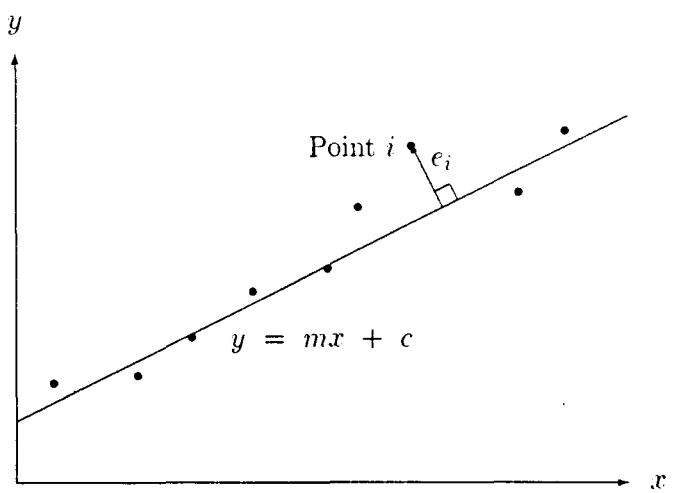


Figure 4.1: Calculation of error when fitting a line.

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.

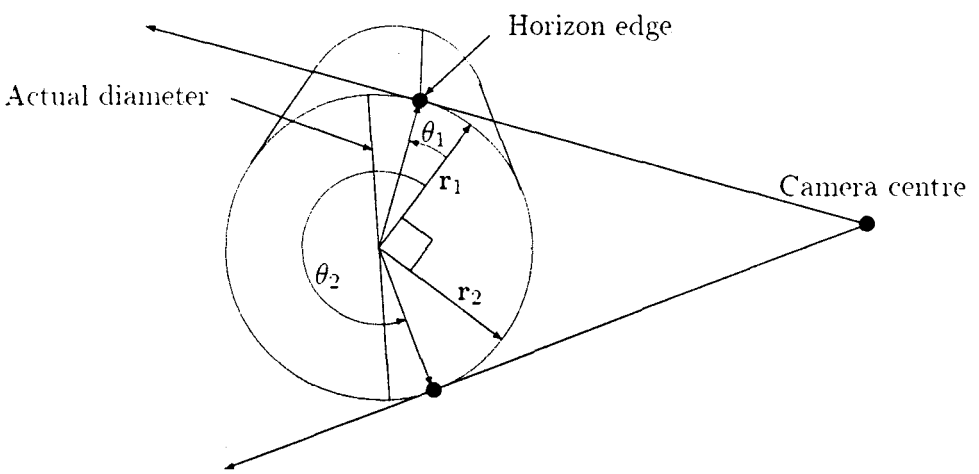


Figure 4.2: Geometry of a projected cylinder.

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.

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

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

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`

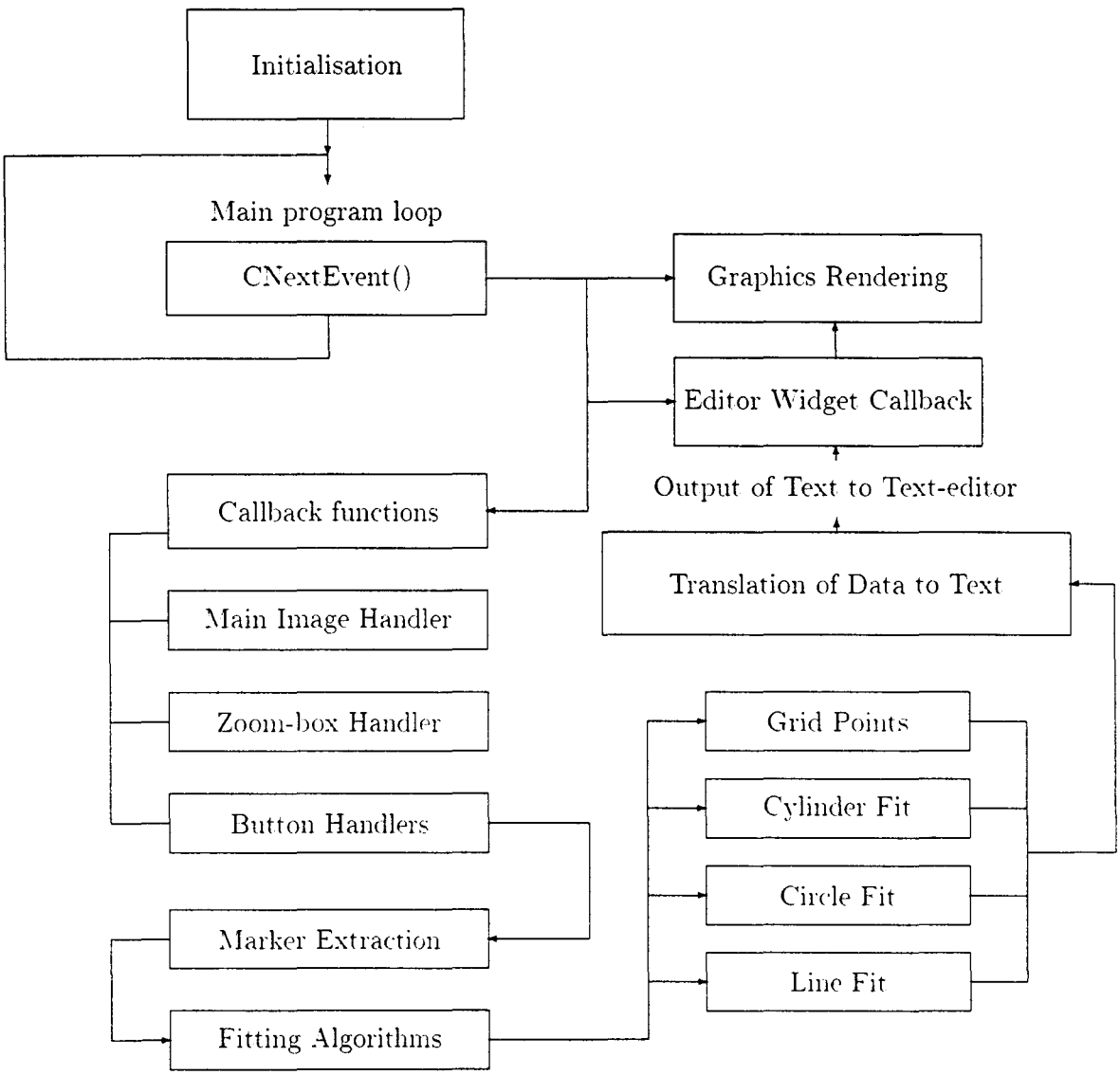


Figure 5.1: Overall program flowchart

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.

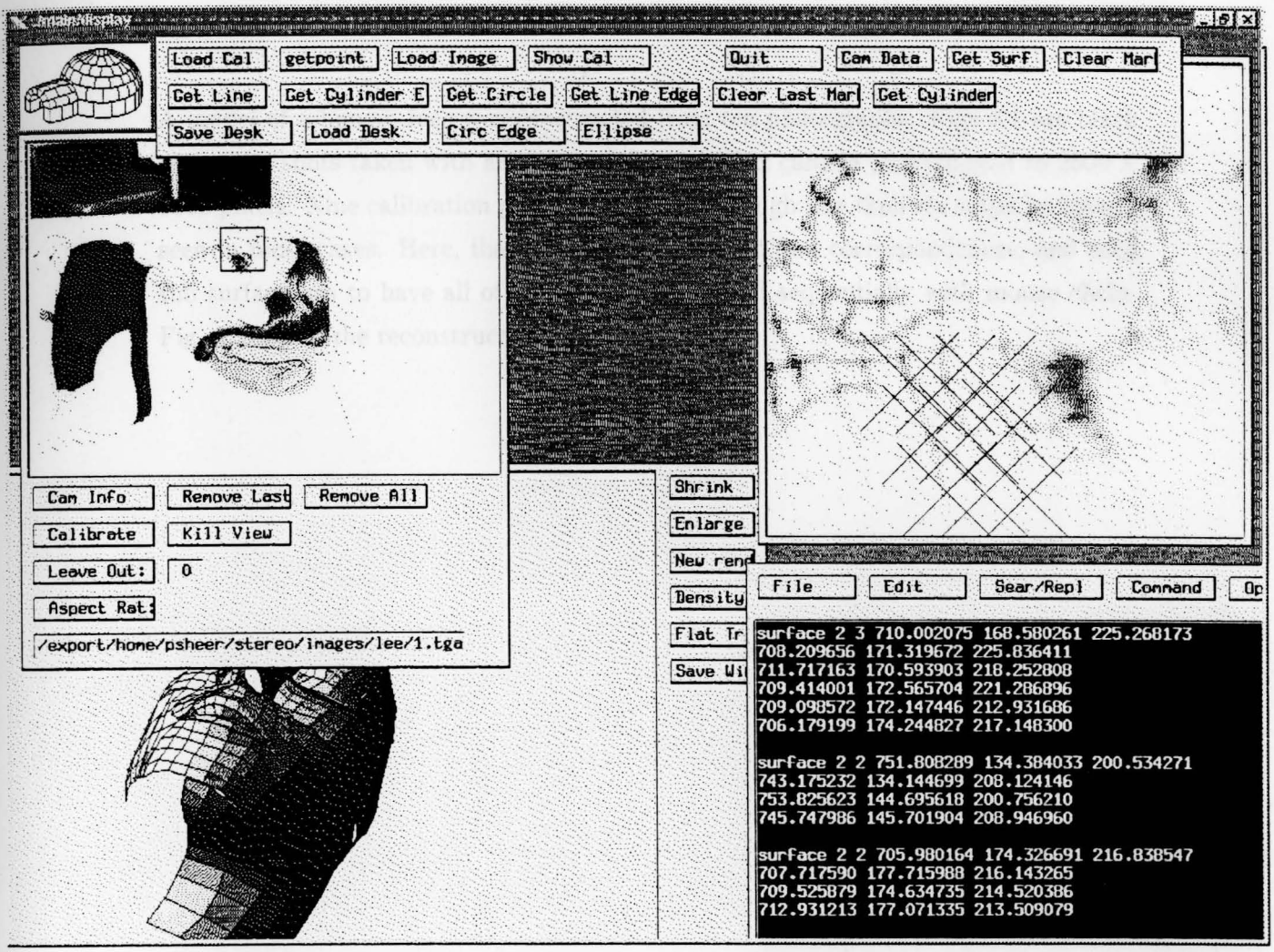


Figure 5.2: Captured view of the desktop during a sample run.

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.

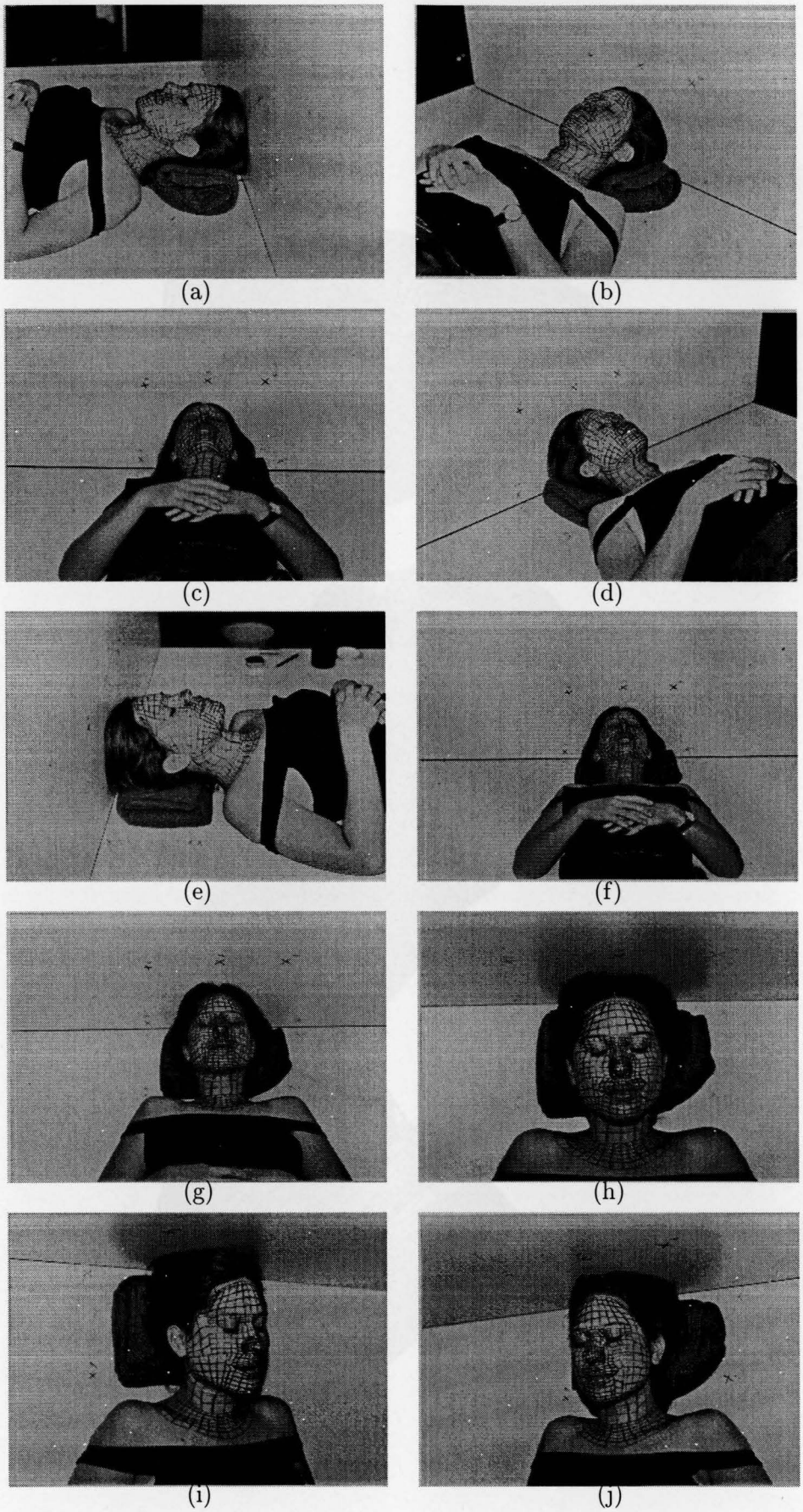


Figure 6.1: Photographs taken of "Lee" at different azimuthal and elevation angles.

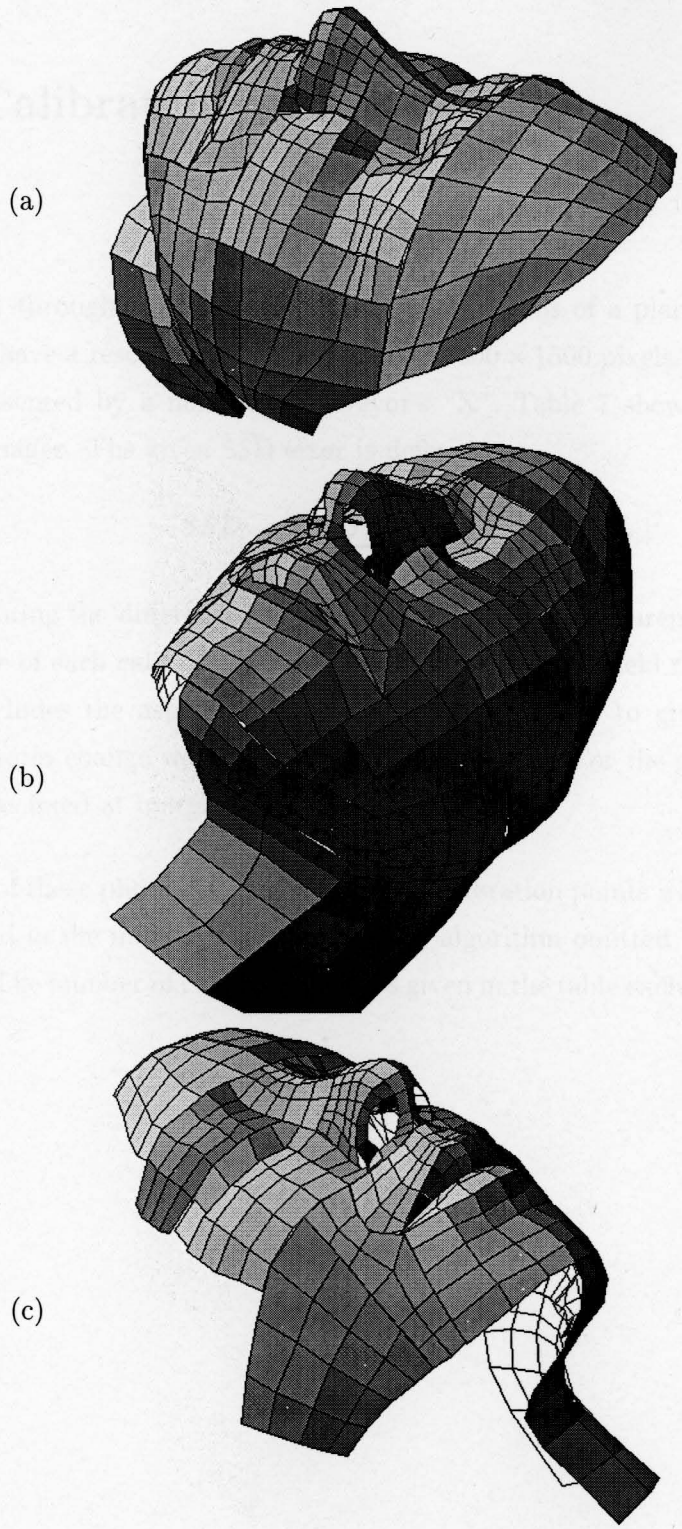


Figure 6.2: 3D rendering of "Lee" triangulated data at various azimuthal angles.

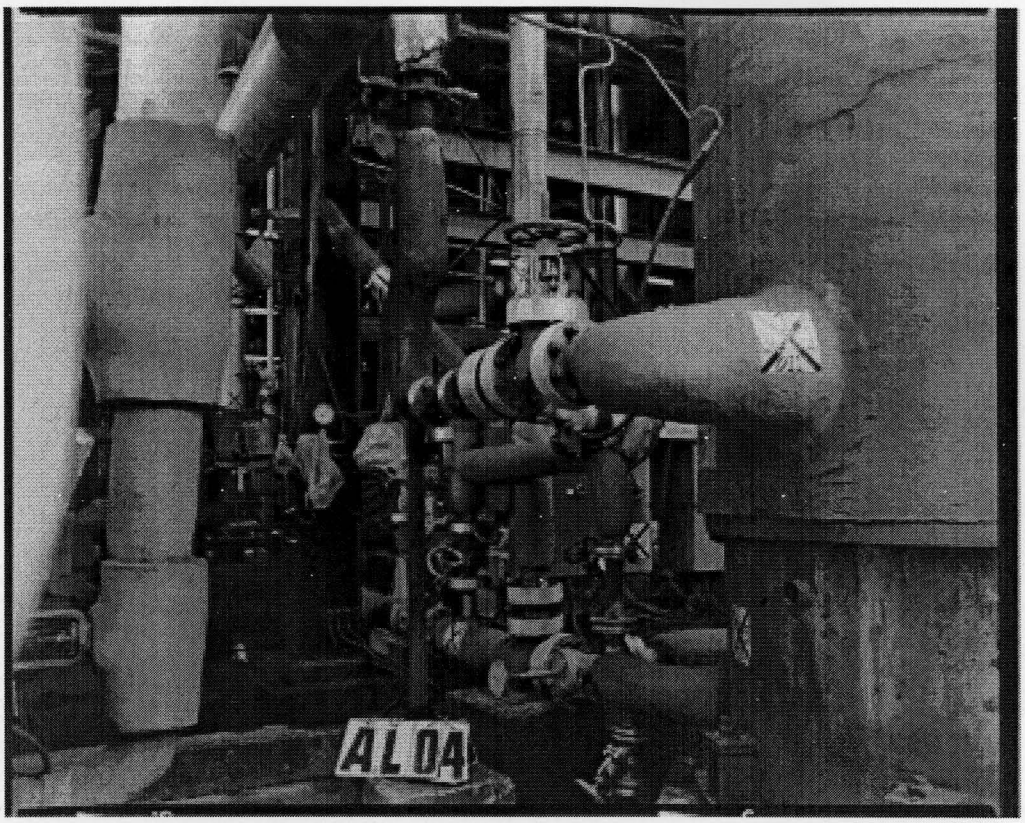
7 Calibration Testing

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,

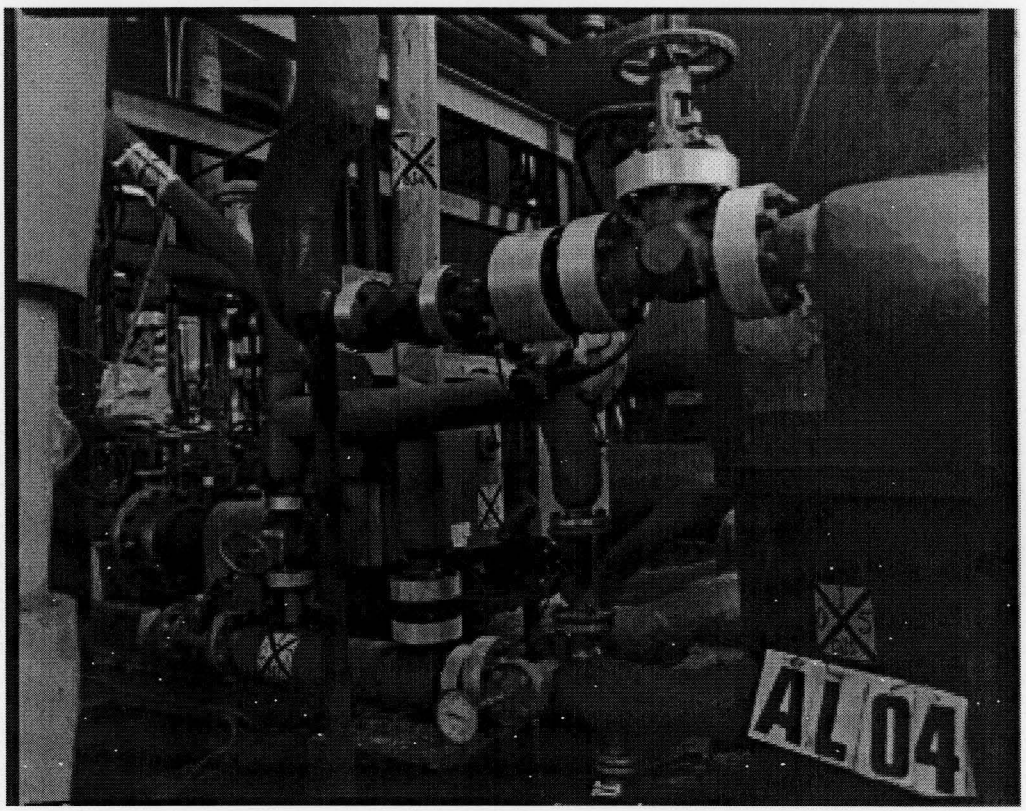
$$SSD = \sum_i |\mathbf{p}_{actual_i} - \mathbf{p}_{projected_i}|^2 \quad , \tag{7.1}$$

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.

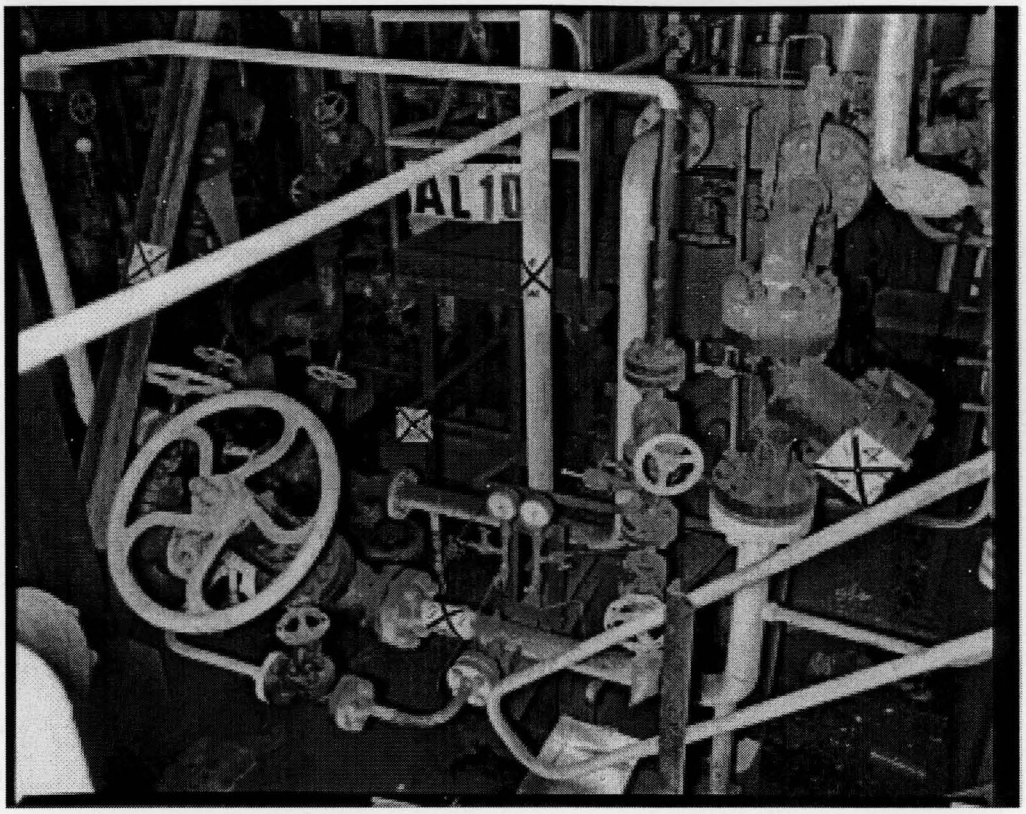


(b)

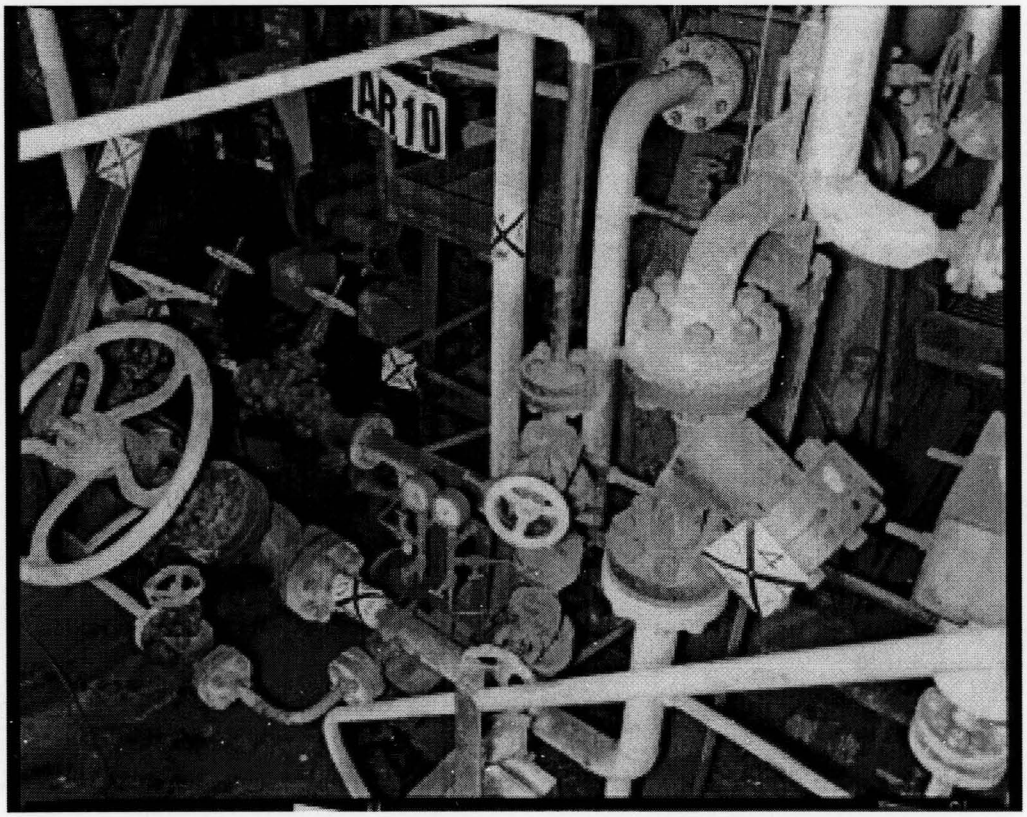


(a)

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

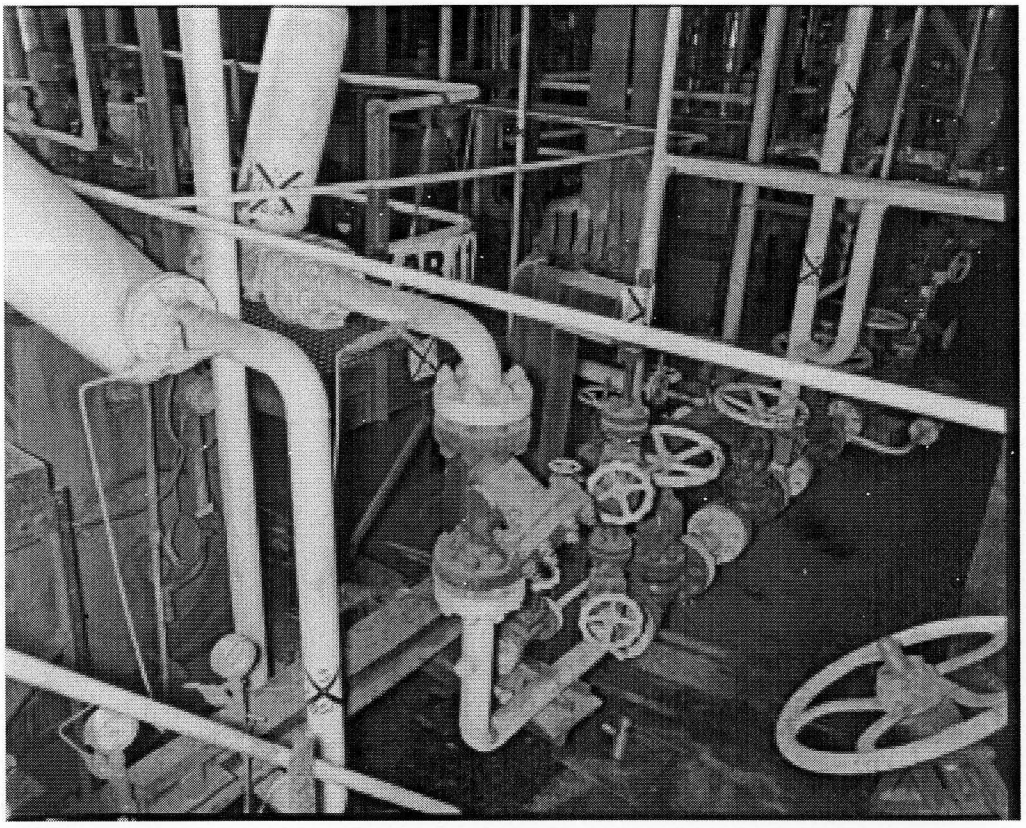


(b)

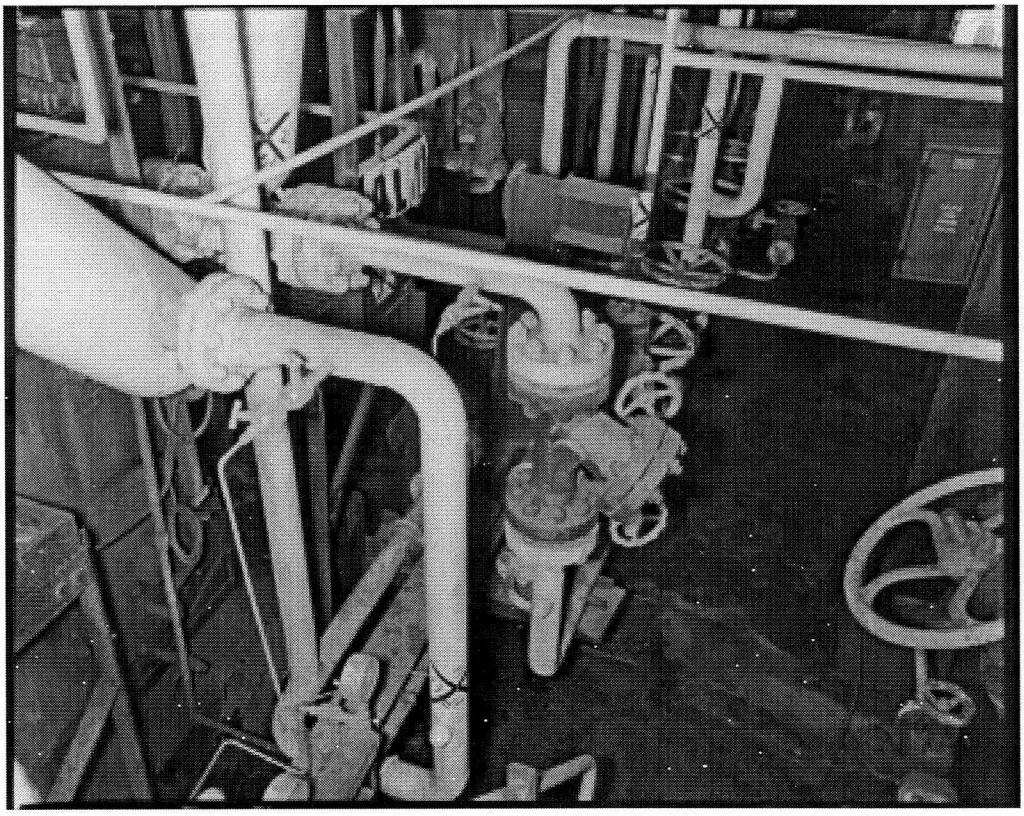


(a)

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

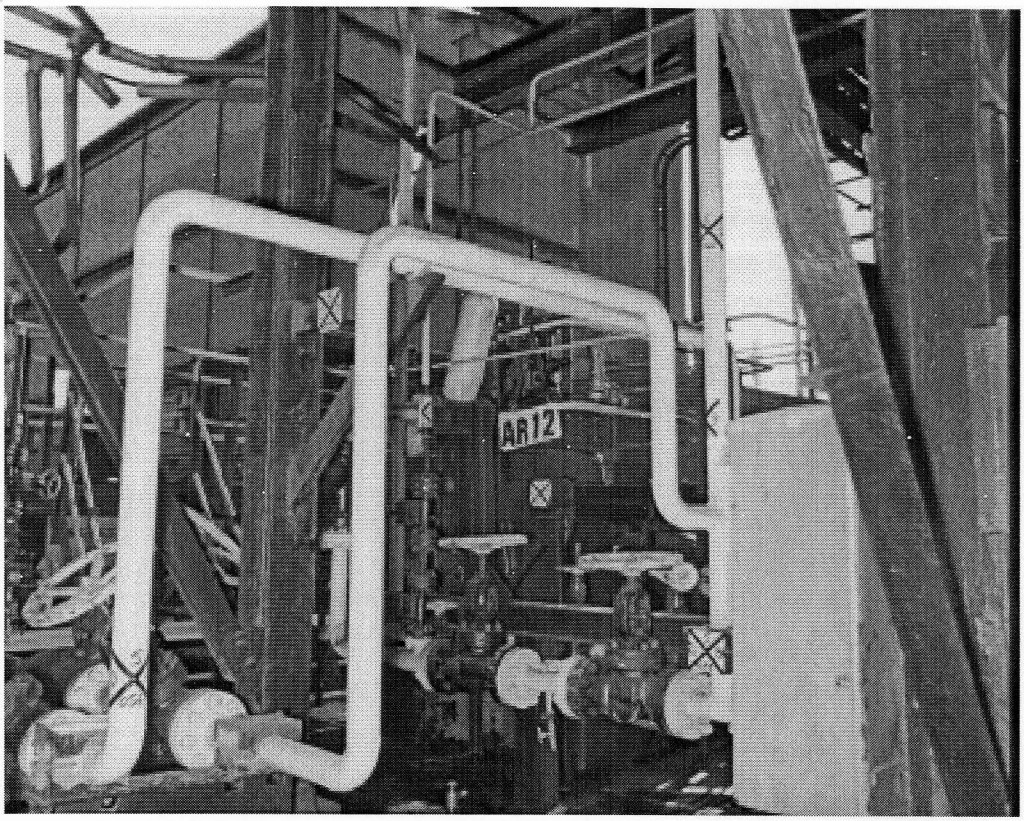


(b)

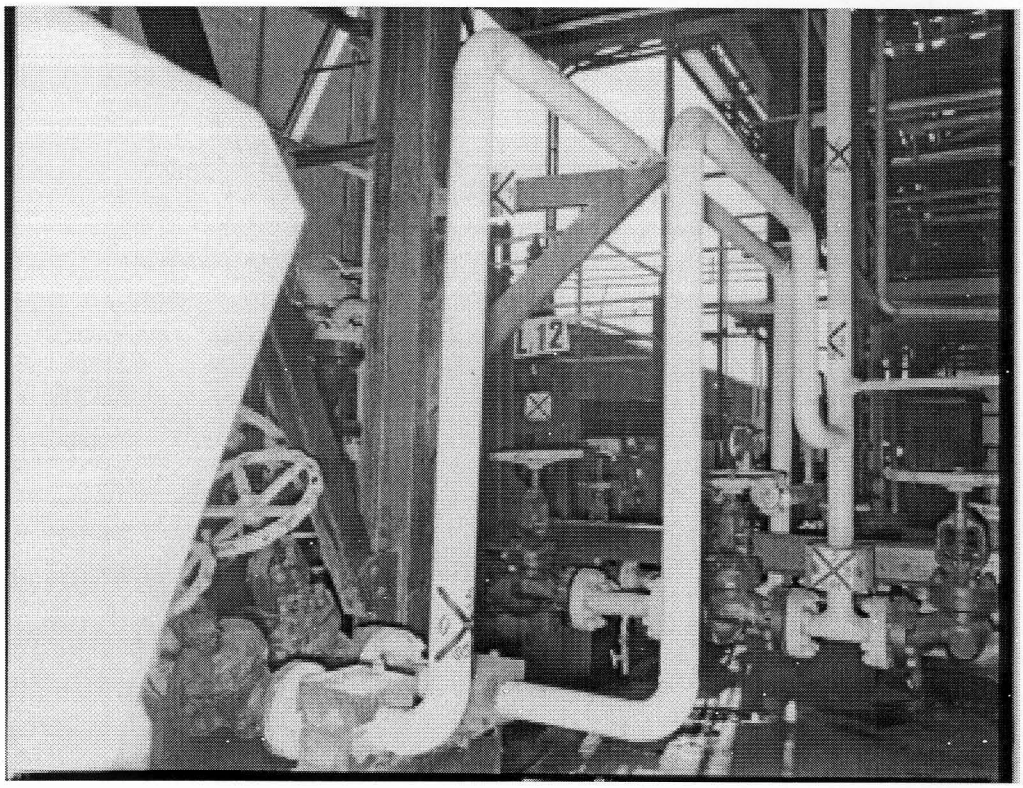


(a)

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

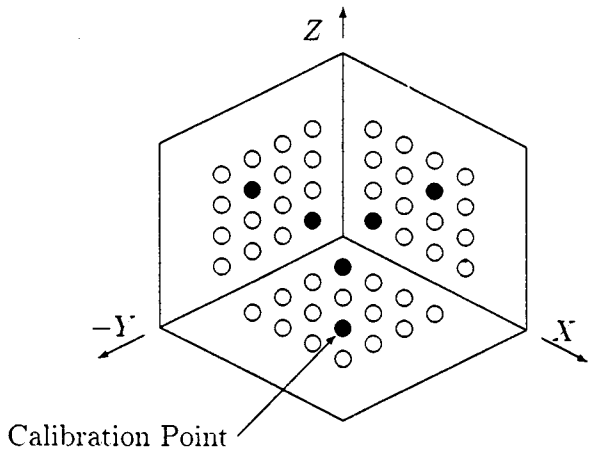


Figure 8.1: Schematic of artificial setup.

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	[°]	1.26	1.25	0.07	0.13
θ	[°]	15.04	15.25	15.24	14.88
ψ	[°]	-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.

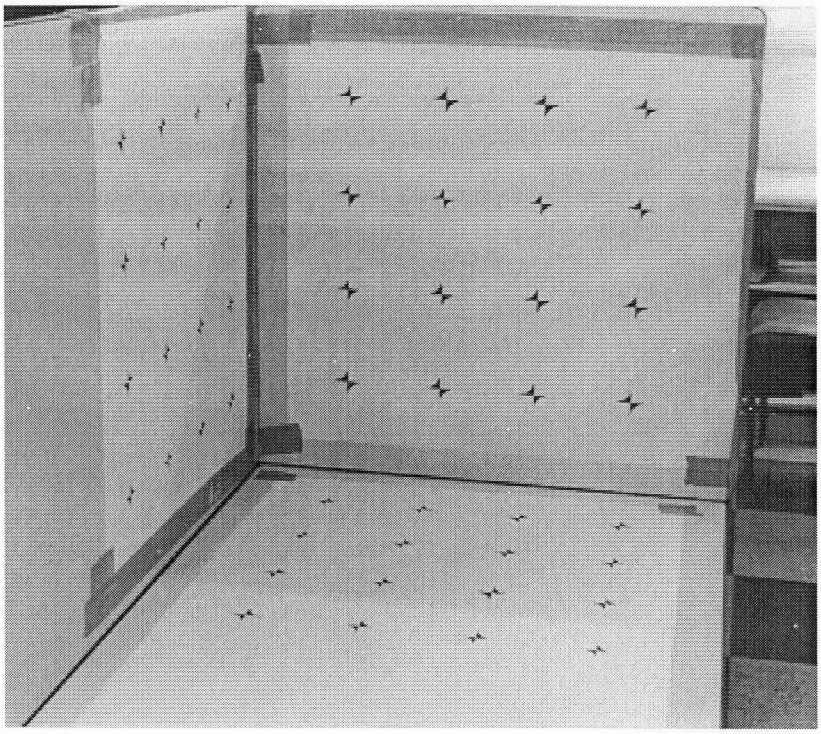
The results of the camera calibrations are shown in Table 8.2. Here the aspect ratio is included in the calibration, since it is relatively consistent over the four views.

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.

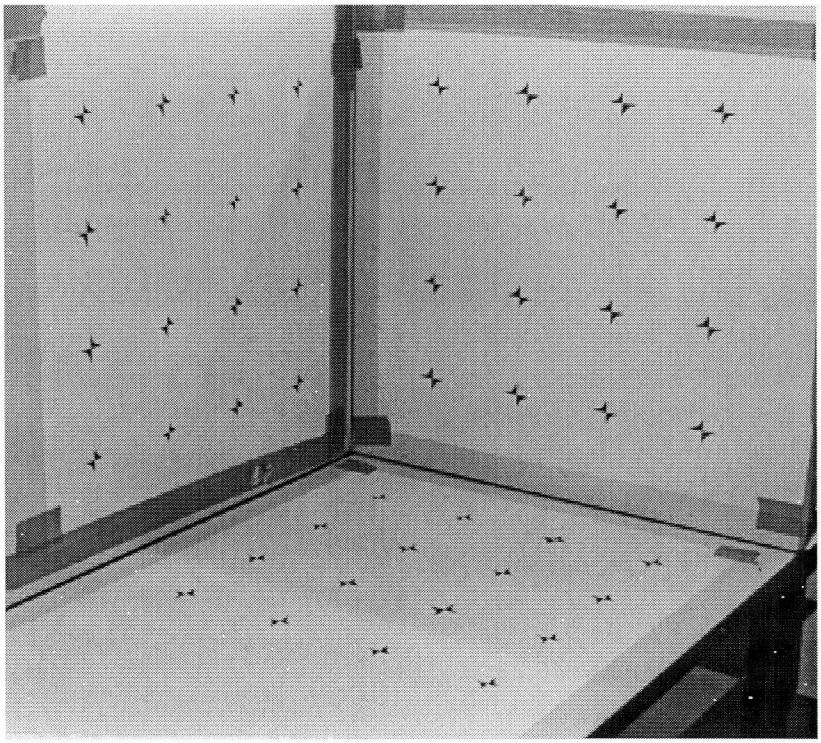
The circle fit was then tested with this setup. If the corner points are omitted, the outer-most points of each group inscribe a circle. Fitting a circle to the eight points gives the three circles of Table 8.4. (Note that the nominal radius is $\sqrt{75^2 + 225^2} = 237.17mm$ from the nominal point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

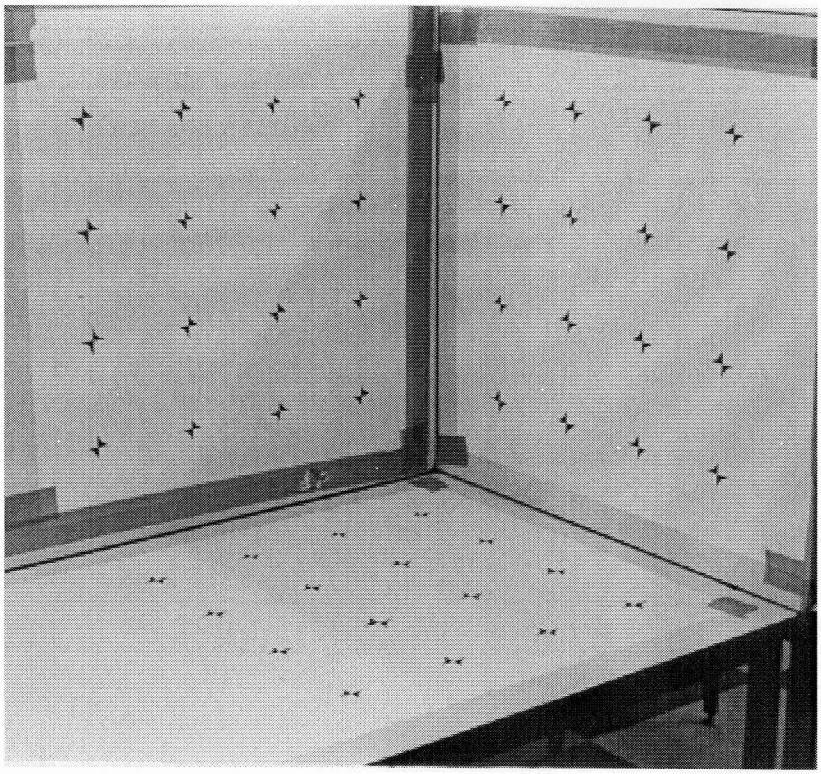


(a)

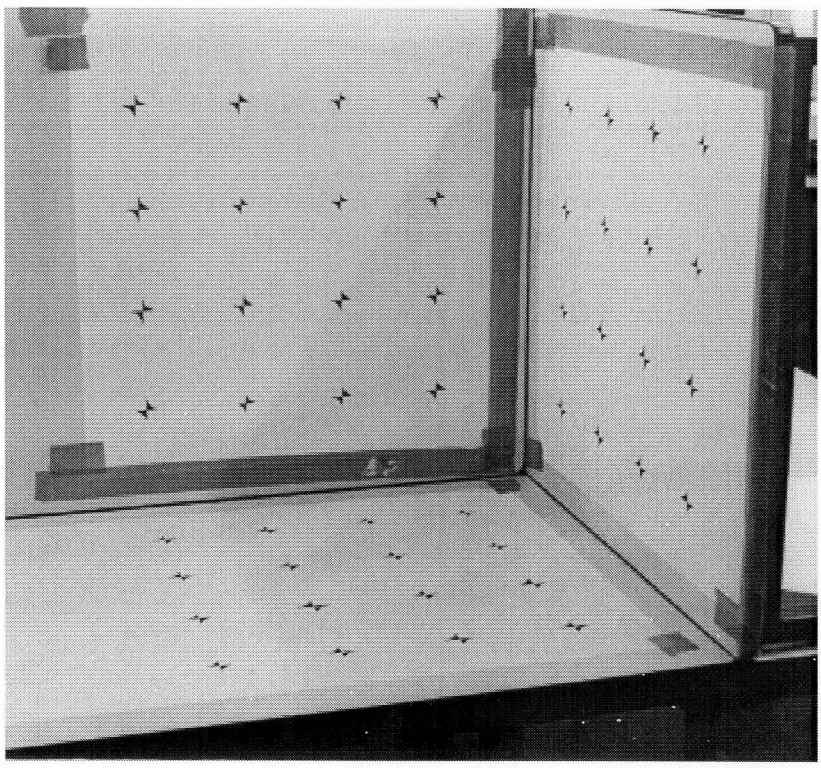


(b)

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.

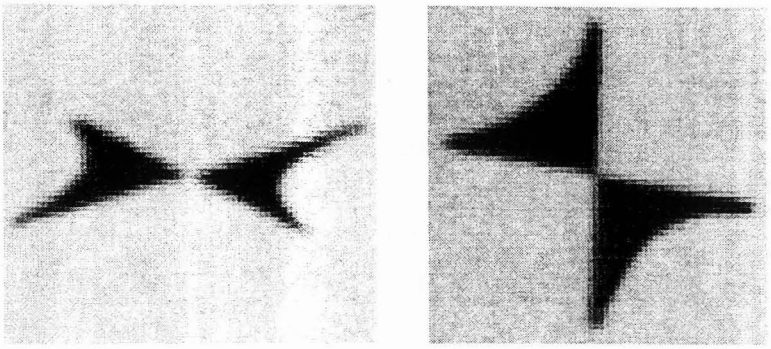


Figure 8.4: Close-up of two calibration points, showing the level of detail in the scan.

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"
```

```
26:26:#include "simplex.h"
27:27:#include "display.h"
28:28:#include "marker.h"
29:29:#include "dialog.h"
30:30:#include "quickmath.h"
31:31:#include "imagefit.h"
32:32:
33:33:#include "mad.h"
34:34:
35:35:extern unsigned char cross_bits[];
36:36:
37:37:#define FTOL 1e-10
38:38:#define ROUGH_TOL 1e-7
39:39:#define MIN_MARKS_NEEDED_TO_CALIBRATE 4
40:40:
41:41:static int stop = 0;
42:42:
43:43:/* call once with omit passed as NULL before using. This is to initialise */
44:44:/*
45:45:   procedure tries all combinarion of leaving out i of the 3D points
46:46:   (because not enough markers where specified) plus the points
47:47:   leaveout1 and leavout2 (if they are passed as non-negative)
48:48: */
49:49:char *recursive_loop (Desktop * d, char omit[], int from, int to, int i, double tol, \
50:50:                       int leaveout1, int leaveout2, int calc_distortion)
51:51:{
52:52:    int j;
53:53:    double e;
54:54:    static double emin = 10e90;
55:55:    static char *omitmin = 0;
56:56:    if (!omit) {
57:57:        emin = 10e90;
58:58:        destroy ((void *) &omitmin);
59:59:        return 0;
60:60:    }
61:61:    if (i) {
62:62:        for (j = from; j <= to - i; j++) {
63:63:            if (stop)
64:64:                break;
65:65:            omit[j] = 1;
66:66:            recursive_loop (d, omit, j + 1, to, i - 1, tol, leaveout1, leaveout2, \
67:67:                            calc_distortion);
68:68:            omit[j] = 0;
69:69:        }
70:70:    } else {
71:71:        Vec *v;
72:72:        double *x, *y;
73:73:        int leave_out = (leaveout1 >= 0) + (leaveout2 >= 0);
74:74:        int k;
75:75:        int n;
76:76:        for (i = 0, j = 0; j < to; i += omit[j++]);
77:77:        n = to - i - leave_out;
78:78:        v = Cmalloc (sizeof (Vec) * n);
79:79:        x = Cmalloc (sizeof (double) * n);
80:80:        y = Cmalloc (sizeof (double) * n);
81:81:        i = 0;
82:82:        k = 0;
83:83:        for (j = 0; j < to; j++) {
84:84:            printf ("%d", (int) omit[j]);
85:85:            if (!omit[j]) {
86:86:                if (k != leaveout1 && k != leaveout2) {
87:87:                    v[i] = d->cal_points[j];
88:88:                    x[i] = d->view[d->current_view].mark[k].x;
89:89:                    y[i] = d->view[d->current_view].mark[k].y;
90:90:                    i++;
91:91:                }
92:92:                k++;
93:93:            }
94:94:        }
95:95:        printf ("\n");
96:96:        e = findcameraposition (x, y, v, n, \
97:97:              &(d->view[d->current_view].cam), tol, 1, calc_distortion);
98:98:        if (e < emin) {
99:99:            emin = e;
100:100:           destroy ((void *) &omitmin);
101:101:           omitmin = Cmalloc (to);
102:102:           memcpy (omitmin, omit, to);
103:103:       }
104:104:       free (x);
105:105:       free (y);
```

```
106:106:    }
107:107:    return omitmin;
108:108:}
109:109:
110:110:
111:111:/*
112:112:   leave out specifies the number of calibration points to omit when
113:113:   making the optimisation. The routine will try omitting every
114:114:   combination of points (eg. 1st then 2nd, then 1st then 3rd, etc)
115:115:   and find which produces the least error. This allows the program
116:116:   to cope with one or two calibration points that are eroneous
117:117:   ('leave_out' can be only 0, 1 or 2). The routine also tries
118:118:   every combination of omition when the number of markers is less
119:119:   than the number of calibration points.
120:120:   'ce' is not used.
121:121: */
122:122:/* returns 0 if succesful, 2 if cal points 'n markers not ok, and 1 if cancelled
123:123:   during operation */
124:124:int calibrate_view (Desktop * d, CEvent * ce, int leave_out, int calc_distortion)
125:125:{
126:126:    char *omitted, *omitmin;
127:127:    int omit;
128:128:    double ermin = 10e90;
129:129:    int leaveout1 = -1, leaveout2 = -1;
130:130:    int l1 = -1, l2 = -1;
131:131:
132:132:    stop = 0;
133:133:
134:134:    set_current_from_pointer (d, ce);
135:135:    omit = d->num_cal_points - d->view[d->current_view].num_marks;
136:136:
137:137:    {
138:138:        if (d->num_cal_points - leave_out < MIN_MARKS_NEEDED_TO_CALIBRATE) {
139:139:            Cerrordialog (0, 0, 0, " Calibrate ", " You have at least %d \
140:140:                calibration points to calibrate ", MIN_MARKS_NEEDED_TO_CALIBRATE);
141:141:            return 2;
142:142:        }
143:143:        if (omit < 0) {
144:144:            Cerrordialog (0, 0, 0, " Calibrate ", \
145:145:                      " You have more markers than calibration points ");
146:146:            return 2;
147:147:        }
148:148:        if (d->view[d->current_view].num_marks < MIN_MARKS_NEEDED_TO_CALIBRATE) {
149:149:            Cerrordialog (0, 0, 0, " Calibrate ", \
150:150:                          " You have at least %d markers to calibrate ", \
151:151:                          MIN_MARKS_NEEDED_TO_CALIBRATE);
152:152:            return 2;
153:153:        }
154:154:    }
155:155:
156:156:    omitted = Cmalloc (d->num_cal_points);
157:157:    memset (omitted, 0, d->num_cal_points);
158:158:
159:159:    recursive_loop (0, 0, 0, 0, 0, 0, 0, 0, 0);  /* reset */
160:160:
161:161:/* try omitting every combination of calibration points without markers and see which
162:162:   gives the minimum error */
163:163:/* Here we are not yet leaving out markers: we are first trying to find out which
164:164:   markers correspond to which calibration point */
165:165:    printf ("Which markers are which?: Trying all combinations:\n");
166:166:    omitmin = recursive_loop (d, omitted, 0, d->num_cal_points, omit, \
167:167:                              ROUGH_TOL, -1, -1, calc_distortion);
168:168:    if (stop)
169:169:        return 1;
170:170:
171:171:/* save the result, there is a 1 in the array if that calibration point is \
172:172:   omitted omitted */
173:173:    memcpy (omitted, omitmin, d->num_cal_points);
174:174:
175:175:    printf ("You think that %d calibration points must be scrapped: Trying all \
176:176:                combinations:\n", leave_out);
177:177:
178:178:/* now try every combination of leaving out two of the markers
179:179:   (and hence two more calibration points) */
180:180:    l1 = l2 = -1;
181:181:    if (leave_out == 2) {
182:182:        for (leaveout1 = 0; leaveout1 < d->num_cal_points - omit - 1; leaveout1++)
183:183:            for (leaveout2 = leaveout1 + 1; leaveout2 < d->num_cal_points - omit; \
184:184:                 leaveout2++) {
185:185:                recursive_loop (d, omitted, 0, d->num_cal_points, 0, FTOL, \
```

```
186:186:                                leaveout1, leaveout2, calc_distortion);
187:187:                if (d->view[d->current_view].cam.e < ermin) {  /* find the minimum */
188:188:                    if (stop)
189:189:                        return 1;
190:190:                    ermin = d->view[d->current_view].cam.e;
191:191:                    l1 = leaveout1;
192:192:                    l2 = leaveout2;
193:193:                }
194:194:            }
195:195:
196:196:        printf ("Best error was in leaving out marker %d and marker %d.\n", \
197:197:                l1 + 1, l2 + 1);
198:198:
199:199:/* or, if specified, ONE of the markers (and hence two more calibration points) */
200:200:    } else if (leave_out == 1) {
201:201:        leaveout2 = -1;
202:202:        for (leaveout1 = 0; leaveout1 < d->num_cal_points - omit; leaveout1++) {
203:203:            recursive_loop (d, omitted, 0, d->num_cal_points, 0, FTOL, leaveout1, \
204:204:                            -1, calc_distortion);
205:205:            if (d->view[d->current_view].cam.e < ermin) {  /* find the minimum */
206:206:                if (stop)
207:207:                    return 1;
208:208:                ermin = d->view[d->current_view].cam.e;
209:209:                l1 = leaveout1;
210:210:            }
211:211:        }
212:212:
213:213:        printf ("Best error was in leaving out marker %d.\n", l1 + 1);
214:214:
215:215:    } else {
216:216:
217:217:        printf ("All markers where used.\n");
218:218:
219:219:    }
220:220:    if (stop)
221:221:        return 1;
222:222:
223:223:    printf ("Now refine.\n");
224:224:
225:225:    recursive_loop (d, omitted, 0, d->num_cal_points, 0, FTOL, l1, l2, \
226:226:                    calc_distortion);  /* this is now the minimum */
227:227:
228:228:    recursive_loop (0, 0, 0, 0, 0, 0, 0, 0, 0);  /* reset and free */
229:229:
230:230:    d->view[d->current_view].calibrated = 1;
231:231:
232:232:    return 0;
233:233:}
234:234:
235:235:
236:236:double calerror (double *x, double *y, Vec * v, Camera * c, int n)
237:237:{
238:238:    int i, j;
239:239:    Vec *s;
240:240:    double *r;
241:241:    double xs, ys, e = 0;
242:242:    getrotation (&(c->m_x), &(c->m_s), &(c->m_y), c->phi, c->theta, c->tsi);
243:243:    r = Cmalloc ((n * 2) * sizeof (double));
244:244:    s = Cmalloc ((n * 2) * sizeof (Vec));
245:245:    j = 0;
246:246:    for (i = 0; i < n; i++) {
247:247:        xs = x[i];
248:248:        ys = y[i];
249:249:        imagetocamera (c, xs, ys);
250:250:        s[j] = plus (times (c->m_s, xs), times (c->m_x, c->f));
251:251:        r[j] = dot (s[j], v[i]);
252:252:        j++;
253:253:        s[j] = plus (times (c->m_s, ys), times (c->m_y, c->f));
254:254:        r[j] = dot (s[j], v[i]);
255:255:        j++;
256:256:    }
257:257:    c->x = vec_invert (s, r, j);
258:258:    for (i = 0; i < n; i++) {
259:259:        phystocamera (c, v[i], xs, ys);
260:260:        cameratoimage (c, xs, ys);
261:261:        e += fsqr (x[i] - xs) + fsqr (y[i] - ys);
262:262:    }
263:263:    free (s);
264:264:    free (r);
265:265:    return e;
```

```
266:266:}
267:267:
268:268:
269:269:static double *x_data, *y_data;
270:270:static Vec *XYZ_data;
271:271:static int num_cal_points;
272:272:
273:273:static Window calprogwin;
274:274:static Camera camera;
275:275:static Camera fixedvalues;
276:276:
277:277:static XEvent xevent;
278:278:static CEvent cwevent;
279:279:
280:280:int callback (double tol, double error, double *v, int num_evals)
281:281:{
282:282:    static int i = 0;
283:283:
284:284:    i++;
285:285:    if (!(i % 10)) {
286:286:        Cdrawprogress ("caliprog", calprogwin, 20, 20, 260, 25,
287:287:                       (1 - log (tol)) * 65535 / (0.5 - log (FTOL)));
288:288:        cwevent.ident = "canccali";
289:289:        if (Ccheckifevent (&xevent, &cwevent)) {
290:290:            stop = 1;
291:291:            return 1;
292:292:        }
293:293:    }
294:294:    return 0;
295:295:}
296:296:
297:297:
298:298:double tominimise4 (double *x)
299:299:{
300:300:    camera.phi = x[0];
301:301:    camera.theta = x[1];
302:302:    camera.tsi = x[2];
303:303:    camera.f = x[3];
304:304:    camera.sig = fixedvalues.sig;
305:305:    return calerror (x_data, y_data, XYZ_data, &camera, num_cal_points);
306:306:}
307:307:
308:308:double tominimise5 (double *x)
309:309:{
310:310:    camera.phi = x[0];
311:311:    camera.theta = x[1];
312:312:    camera.tsi = x[2];
313:313:    camera.f = x[3];
314:314:    camera.sig = x[4];
315:315:    return calerror (x_data, y_data, XYZ_data, &camera, num_cal_points);
316:316:}
317:317:
318:318:/*
319:319:   finds f, phi, theta and tsi for a given set of calibration points
320:320:   whose 3D positions are given by v and whose 2D picture postions are given
321:321:   by x and y.
322:322:   pass calc_distortion with CALC_SIGMA set to optimise sigma, otherwise set sigma
323:323: */
324:324:double findcameraposition (double *x, double *y, Vec * v, int n, Camera * c, \
325:325:                 double tol, int make_initial_guess, int calc_distortion)
326:326:{
327:327:    int i, j;
328:328:    int numitters;
329:329:    double d, result[10], emin = 10e90, e;
330:330:
331:331:    if (!Cwidget ("calprogw")) {
332:332:        calprogwin = Cdrawwindow ("calprogw", CMain, 50, 50, 300, 110, "");
333:333:        Cwidget ("calprogw")->position = CFIXED_POSITION | CALWAYS_ON_TOP;
334:334:        Cdrawbitmapbutton ("canccali", calprogwin, 126, 53,
335:335:                           40, 40, Ccolor (18), C_FLAT, cross_bits);
336:336:    }
337:337:    Cdrawprogress ("caliprog", calprogwin, 20, 20, 260, 25, 0);
338:338:    Ccheckifevent (NULL, NULL);
339:339:
340:340:    x_data = x;
341:341:    y_data = y;
342:342:    XYZ_data = v;
343:343:    num_cal_points = n;
344:344:
345:345:    memcpy (&fixedvalues, c, sizeof (Camera));
```

```
346:346:
347:347:    if (make_initial_guess) {
348:348:
349:349:/*
350:350:   first we find an estimate for the order of magnitude of f.
351:351:   The calibration points would be spaced across a reasonable proportion
352:352:   of the image. A camera would typically have a field angle of 20 degrees
353:353:   and the image extents divided by tan(20) gives a rough estimate of -f
354:354: */
355:355:        d = 0;
356:356:        for (i = 0; i < n - 1; i++)
357:357:            for (j = i + 1; j < n; j++)
358:358:                d = fmax (d, sqrt (fsqr (y[i] - y[j]) + fsqr (x[i] - x[j])));
359:359:
360:360:/*
361:361:   we will scale this up a bit to over estimate f. This will
362:362:   result in XC being further from the calibration point cluster and
363:363:   ensure that it does not penetrate it during the guesstimation procedure
364:364:   below.
365:365: */
367:367:        memset (c, 0, sizeof (Camera));
368:368:        c->f = d * (-4.0);
370:370:/*search for an initial guess of the entire space (this is quite fast) */
371:371:        for (c->theta = -PI / 2; c->theta <= PI / 2; c->theta += PI / 8) {
372:372:            cwevent.ident = "canccali";
373:373:            if (Ccheckifevent (&xevent, &cwevent)) {
374:374:                stop = 1;
375:375:                return 1;
376:376:            }
377:377:            for (c->tsi = -PI; c->tsi < PI; c->tsi += PI / 8)
378:378:                for (c->phi = -PI; c->phi < PI; c->phi += PI / 8) {
379:379:                    e = calerror (x_data, y_data, XYZ_data, c, n);
380:380:                    numitters = 0;
381:381:                    if (emin > e) {
382:382:                        emin = e;
383:383:                        memcpy (&camera, c, sizeof (Camera));
384:384:                    }
385:385:                }
386:386:        }
387:387:    } else {                    /* initial guess is given */
388:388:        memcpy (&camera, c, sizeof (Camera));
389:389:    }
390:390:    /*
391:391:       now we have an estimate for f, theta, and tsi and phi.
392:392:       First guess for the other camera parameters is zero (if there are any).
393:393:     */
394:394:    result[0] = camera.phi;
395:395:    result[1] = camera.theta;
396:396:    result[2] = camera.tsi;
397:397:    result[3] = camera.f;
399:399:    if (calc_distortion & CALC_SIGMA) {
400:400:        result[4] = 0;
401:401:        numitters = simplex_optimise (result, 5, tol, PI / 16, tominimise5, callback);
402:402:    } else {
403:403:        numitters = simplex_optimise (result, 4, tol, PI / 16, tominimise4, callback);
404:404:        result[4] = fixedvalues.sig;
405:405:    }
407:407:    c->phi = result[0];
408:408:    c->theta = result[1];
409:409:    c->tsi = result[2];
410:410:    c->f = result[3];
411:411:    c->sig = result[4];
413:413:    getrotation (&(c->m_x), &(c->m_s), &(c->m_y), c->phi, c->theta, c->tsi);
415:415:    c->e = calerror (x_data, y_data, XYZ_data, c, n);
416:416:    return c->e;
417:417:}
418:1:#ifndef CALIBRATE_H
419:2:#define CALIBRATE_H
420:3:
421:4:#include "camera.h"
422:5:#include "display.h"
423:6:
424:7:#define CALC_SIGMA 1
425:8:
426:9:double findcameraposition (double *x, double *y, Vec * v, int n, Camera * c, \
427:10:                double tol, int make_initial_guess, int calc_distortion);
428:11:
429:12:int calibrate_view (Desktop * d, CEvent * ce, int leave_out, int calc_distortion);
430:13:
431:14:#endif
```

```
/*****************************************************************************/
/* callback.c - main callback routines from button presses                   */
/*****************************************************************************/

#include "display.h"
#include "widget3d.h"
#include "main/marker.h"
#include "main/displaycam.h"
#include "loadcalfile.h"
#include "imagehandler.h"
#include "picsetup.h"
#include "edit.h"
#include "dialog.h"
#include "loadfile.h"
#include "desktop.h"
#include "imagefit.h"
#include "calibrate.h"

extern Desktop desktop;

int save_window_to_file (Window win, int x, int y, int width, int height);

int cb_save_window (CWidget * none, XEvent * xevent, CEvent * cwevent)
{
    CWidget *w;
    w = Cwidget ("3dview");
    save_window_to_file (w->winid, 0, 0, w->width, w->height);
    return 0;
}


int cb_getsurface (CWidget * none, XEvent * xevent, CEvent * cwevent)
{
    output_surface (&desktop);
    return 0;
}

int cb_getcircle (CWidget * none, XEvent * xevent, CEvent * cwevent)
{
    output_circle (&desktop);
    return 0;
}

int cb_getellipse (CWidget * none, XEvent * xevent, CEvent * cwevent)
{
    output_ellipse (&desktop);
    return 0;
}

int cb_getpoint (CWidget * none, XEvent * xevent, CEvent * cwevent)
{
    output_point (&desktop);
    return 0;
}

int cb_getline (CWidget * none, XEvent * xevent, CEvent * cwevent)
{
    output_line (&desktop);
    return 0;
}

int cb_getcircleedge (CWidget * none, XEvent * xevent, CEvent * cwevent)
{
    output_circle_edge (&desktop);
    return 0;
}

int cb_getlineedge (CWidget * none, XEvent * xevent, CEvent * cwevent)
{
    output_line_edge (&desktop);
    return 0;
}

int cb_getcylinderedge (CWidget * none, XEvent * xevent, CEvent * cwevent)
{
    output_cylinder_edge (&desktop);
    return 0;
}

int cb_getcylinder (CWidget * none, XEvent * xevent, CEvent * cwevent)
```

```
512:81:{
513:82:    output_cylinder (&desktop);
514:83:    return 0;
515:84:}
516:85:
517:86:int cb_save_desktop (CWidget * w, XEvent * xe, CEvent * ce)
518:87:{
519:88:    save_desktop (&desktop);
520:89:    return 0;
521:90:}
522:91:
523:92:int cb_load_desktop (CWidget * w, XEvent * xe, CEvent * ce)
524:93:{
525:94:    load_desktop (&desktop);
526:95:    return 0;
527:96:}
528:97:
529:98:int cb_leave (CWidget * w, XEvent * xe, CEvent * ce)
530:99:{
531:100:    Picture *p;
532:101:    set_current_from_pointer (&desktop, ce);
533:102:    p = &desktop.view[desktop.current_view].pic;
534:103:    Credrawtext (p->Tleave->ident, " %d ", (atoi (p->Tleave->text + 1) + 1) % 3);
535:104:    return 0;
536:105:}
537:106:
538:107:int cb_sigma (CWidget * w, XEvent * xe, CEvent * ce)
539:108:{
540:109:    Picture *p;
541:110:    set_current_from_pointer (&desktop, ce);
542:111:
543:112:    p = &desktop.view[desktop.current_view].pic;
544:113:    if (p->Isigma) {
545:114:        Cundrawwidget (p->Isigma->ident);
546:115:        p->Isigma = 0;
547:116:    } else {
548:117:        p->Isigma = Cdrawtextinput (catstrs (p->Ssig->ident, ".input", 0),
549:118:                                    p->main_win->winid, 120, p->main_win->height - 60, 50, 20, 10, "0"
550:119:    }
551:120:    return 0;
552:121:}
553:122:
554:123:
555:124:int cb_calibrate (CWidget * w, XEvent * xe, CEvent * ce)
556:125:{
557:126:    int calc_distortion = 0;
558:127:    Picture *p;
559:128:    double sigma;
560:129:    set_current_from_pointer (&desktop, ce);
561:130:    p = &desktop.view[desktop.current_view].pic;
562:131:    if (p->Isigma) {
563:132:        sigma = atof (p->Isigma->text);
564:133:    } else {
565:134:        sigma = 0;
566:135:        calc_distortion |= CALC_SIGMA;
567:136:    }
568:137:    desktop.view[desktop.current_view].cam.sig = sigma;
569:138:    if (calibrate_view (&desktop, ce, atoi (p->Tleave->text + 1), calc_distortion) == 1)
570:139:        clear (&desktop.view[desktop.current_view].cam, Camera);
571:140:    Cundrawwidget ("calprogw");
572:141:    return 0;
573:142:}
574:143:
575:144:int cb_showcal (CWidget * w, XEvent * xe, CEvent * ce)
576:145:{
577:146:    show_cal_points (&desktop, ce);
578:147:    return 0;
579:148:}
580:149:
581:150:void insert_marker (Picture * p, CEvent * e)
582:151:{
583:152:    Vec v;
584:153:    v = zoom_event_to_image_coord (p, e);
585:154:    new_marker (&desktop, v);
586:155:    draw_markers (&desktop);
587:156:}
588:157:
589:158:void move_marker (Picture * p, CEvent * e)
590:159:{
591:160:    Vec v;
```

```
592:161:    v = zoom_event_to_image_coord (p, e);
593:162:    move_closest_marker (&desktop, v);
594:163:    draw_markers (&desktop);
595:164:}
596:165:
597:166:void remove_marker (Picture * p, CEvent * e)
598:167:{
599:168:    Vec v;
600:169:    v = zoom_event_to_image_coord (p, e);
601:170:    remove_closest_marker (&desktop, v);
602:171:    draw_markers (&desktop);
603:172:}
604:173:
605:174:
606:175:int cb_draw_cam_data (CWidget * w, XEvent * xe, CEvent * ce)
607:176:{
608:177:    draw_camera_data (&desktop, ce);
609:178:    return 0;
610:179:}
611:180:
612:181:int cb_loadcal (CWidget * w, XEvent * xe, CEvent * ce)
613:182:{
614:183:    load_calibration (&desktop);
615:184:    return 0;
616:185:}
617:186:
618:187:int cb_zoomimage (CWidget * w, XEvent * xe, CEvent * ce)
619:188:{
620:189:    handle_zoom_box (&desktop, ce);
621:190:    return 0;
622:191:}
623:192:
624:193:int cb_mainimage (CWidget * w, XEvent * xe, CEvent * ce)
625:194:{
626:195:    handle_main_box (&desktop, ce);
627:196:    return 0;
628:197:}
629:198:
630:199:int cb_newimage (CWidget * w, XEvent * xe, CEvent * ce)
631:200:{
632:201:    load_view (&desktop);
633:202:    return 0;
634:203:}
635:204:
636:205:int cb_killimage (CWidget * w, XEvent * xe, CEvent * ce)
637:206:{
638:207:    set_current_from_pointer (&desktop, ce);
639:208:    destroy_current_view (&desktop);
640:209:    return 0;
641:210:}
642:211:
643:212:int cb_clearallmarkers (CWidget * none, XEvent * xe, CEvent * ce)
644:213:{
645:214:    set_current_from_pointer (&desktop, ce);
646:215:    clear_markers (&desktop);
647:216:    draw_markers (&desktop);
648:217:    return 0;
649:218:}
650:219:
651:220:int cb_removelastmarker (CWidget * none, XEvent * xe, CEvent * ce)
652:221:{
653:222:    set_current_from_pointer (&desktop, ce);
654:223:    remove_last_marker (&desktop);
655:224:    draw_markers (&desktop);
656:225:    return 0;
657:226:}
658:227:
659:228:int cb_newrender (CWidget * w, XEvent * xevent, CEvent * cwevent)
660:229:{
661:230:    w = Cwidget ("3dview");
662:231:    switch (w->solid->render) {
663:232:    case TD_MESH:
664:233:        w->solid->render = TD_MESH_AND_SOLID;
665:234:        break;
666:235:    case TD_MESH_AND_SOLID:
667:236:        w->solid->render = TD_SOLID;
668:237:        break;
669:238:    case TD_SOLID:
670:239:        w->solid->render = TD_EDGES_ONLY;
671:240:        break;
```

```
672:241:    case TD_EDGES_ONLY:
673:242:        w->solid->render = TD_MESH;
674:243:        break;
675:244:    }
676:245:    Credraw3dobject ("3dview", 0);
677:246:    return 0;
678:247:}
679:248:
680:249:#define CK_Save                 101
681:250:
682:251:int Cdraw3d_from_text (const char *ident, const char *text);
683:252:
684:253:int cb_editor (CWidget * editor, XEvent * xevent, CEvent * cwevent)
685:254:{
686:255:    char *t;
687:256:    long tlen;
688:257:
689:258:    if ((cwevent->key == XK_r || cwevent->key == XK_R) && (cwevent->state & Mod1Mask)) {
690:259:        Cclear_all_surfaces ("3dview");
691:260:        Cedit_execute_command (editor->editor, CK_Save, -1);
692:261:        t = loadfile (path_compress (editor->editor->dir, editor->editor->filename), &tlen);
693:262:        if (t && tlen) {
694:263:            Cdraw3d_from_text ("3dview", t);
695:264:        } else
696:265:            Cerrordialog (0, 0, 0, "Get Surface", \
697:266:                          "Error reloading file \"%s\" in call to cb_editor - check file permissions.'
698:267:                          editor->editor->filename);
699:268:    }
700:269:    return 0;
701:270:}
702:271:
703:272:int cb_3dplane (CWidget * w, XEvent * xevent, CEvent * cwevent)
704:273:{
705:274:    static int y3dprev = 0, x3dprev = 0;
706:275:    static float alphaprev = 0, betaprev = 0;
707:276:    if (cwevent->type == ButtonPress && cwevent->button == Button1) {
708:277:        x3dprev = cwevent->x;
709:278:        y3dprev = cwevent->y;
710:279:        alphaprev = w->solid->alpha;
711:280:        betaprev = w->solid->beta;
712:281:    }
713:282:    if ((cwevent->type == MotionNotify && (cwevent->state & Button1Mask)) ||
714:283:        (cwevent->type == ButtonRelease && cwevent->button == Button1)) {
715:284:        w->solid->alpha = (float) alphaprev + (float) ((float) cwevent->x - x3dprev) / 100;
716:285:        w->solid->beta = (float) betaprev + (float) ((float) cwevent->y - y3dprev) / 100;
717:286:        Credraw3dobject ("3dview", 0);
718:287:    }
719:288:    return 0;
720:289:}
721:1:
722:2:#ifndef _CALLBACK_H
723:3:#define _CALLBACK_H
724:4:
725:5:void move_marker (Picture * p, CEvent * e);
726:6:void remove_marker (Picture * p, CEvent * e);
727:7:void insert_marker (Picture * p, CEvent * e);
728:8:int cb_calibrate (CWidget * w, XEvent * xe, CEvent * ce);
729:9:int cb_loadcal (CWidget * w, XEvent * xe, CEvent * ce);
730:10:int cb_zoomimage (CWidget * w, XEvent * xe, CEvent * ce);
731:11:int cb_mainimage (CWidget * w, XEvent * xe, CEvent * ce);
732:12:int cb_newimage (CWidget * w, XEvent * xe, CEvent * ce);
733:13:int cb_killimage (CWidget * w, XEvent * xe, CEvent * ce);
734:14:int cb_clearallmarkers (CWidget * none, XEvent * xe, CEvent * ce);
735:15:int cb_removelastmarker (CWidget * none, XEvent * xe, CEvent * ce);
736:16:int cb_newrender (CWidget * w, XEvent * xevent, CEvent * cwevent);
737:17:int cb_editor (CWidget * editor, XEvent * xevent, CEvent * cwevent);
738:18:int cb_3dplane (CWidget * w, XEvent * xevent, CEvent * cwevent);
739:19:int cb_draw_cam_data (CWidget * w, XEvent * xe, CEvent * ce);
740:20:int cb_showcal (CWidget * w, XEvent * xe, CEvent * ce);
741:21:int cb_leave (CWidget * w, XEvent * xe, CEvent * ce);
742:22:int cb_sigma (CWidget * w, XEvent * xe, CEvent * ce);
743:23:int cb_save_desktop (CWidget * w, XEvent * xe, CEvent * ce);
744:24:int cb_load_desktop (CWidget * w, XEvent * xe, CEvent * ce);
745:25:int cb_getsurface (CWidget * none, XEvent * xevent, CEvent * cwevent);
746:26:int cb_getcircle (CWidget * none, XEvent * xevent, CEvent * cwevent);
747:27:int cb_getellipse (CWidget * none, XEvent * xevent, CEvent * cwevent);
748:28:int cb_getline (CWidget * none, XEvent * xevent, CEvent * cwevent);
749:29:int cb_save_window (CWidget * none, XEvent * xevent, CEvent * cwevent);
750:30:int cb_getcylinderedge (CWidget * none, XEvent * xevent, CEvent * cwevent);
751:31:int cb_getlineedge (CWidget * none, XEvent * xevent, CEvent * cwevent);
```

```
752:32:int cb_getcircleedge (CWidget * none, XEvent * xevent, CEvent * cwevent);
753:33:
754:34:#endif
755:1:
756:2:/*****************************************************************************************/
757:3:/* camera.c - undestort camera coordinates                                               */
758:4:/*****************************************************************************************/
759:5:
760:6:#include <config.h>
761:7:#include "global.h"
762:8:#include "matrix.h"
763:9:#include "camera.h"
764:10:
765:11:Matrix *Macamundestortmany (Camera * c, Matrix * x)
766:12:{
767:13:    int i = 0;
768:14:    for (; i < x->columns; i++)
769:15:        imagetocamera (c, (Mard (*x, 0, i)), (Mard (*x, 1, i)));
770:16:    return x;
771:17:}
772:18:
773:19:void camundestortmany (Camera * cam, double *x, double *y, int n)
774:20:{
775:21:    int i;
776:22:    for (i = 0; i < n; i++)
777:23:        imagetocamera (cam, x[i], y[i]);
778:24:}
779:1:#ifndef CAMERA_H
780:2:#define CAMERA_H
781:3:
782:4:#include "matrix.h"
783:5:#include "quickmath.h"
784:6:
785:7:typedef struct {
786:8:    double phi;
787:9:    double theta;
788:10:    double tsi;
789:11:    double f;
790:12:    double sig;                /*sigma */
791:13:    double s;
792:14:    Vec x;                     /* centre */
793:15:    Vec m_x, m_y, m_s;
794:16:/*
795:17:   distortion correction could come here:
796:18: */
797:19:    double a1;
798:20:    double a2;
799:21:    double a3;
800:22:    double a4;
801:23:    double a5;
802:24:    double a6;
803:25:
804:26:/* sum squared error in calculation */
805:27:    double e;
806:28:} Camera;
807:29:
808:30:/*
809:31:   some definitions:
810:32:   a PICTURE is the structure containing black and white of that
811:33:   view. It has coords top left and down.
812:34:
813:35:   an IMAGE is the physical photograph. It has coords centre and up.
814:36:
815:37:   a CAMERA is same as the IMAGE but undestorted
816:38:
817:39:   finally PHYS is the unrotated actual 3D coord.
818:40:
819:41:   sequence of transformation is.
820:42:
821:43:   phystocamera (cam,v,x,y)  /* result in x, y */
822:44:   cameratoimage (cam, v.x, v.y)        /* v.x and v.y altered */
823:45:   imagetopic (pic, v.x, v.y)   /* v.x and v.y altered */
824:46:   the result is the picture coord (from the top left corner and
825:47:   always positive)
826:48:   the reverse is just pictoimage, imagetocamera,
827:49:   camtophys (c1, c2, v, x1, y1, x2, y2)
828:50: */
829:51:
830:52:#define imagetocamera(c,x,y) { \
831:53:    (y) /= (1 + (c)->sig / 100); \
```

```
832:54:}
833:55:
834:56:#define cameratoimage(c,x,y) { \
835:57:    (y) *= (1 + (c)->sig / 100); \
836:58:}
837:59:
838:60:#define phystocamera(c,v,xs,ys) { \
839:61:    Vec ___v; \
840:62:    double ___s = dot ((c)->m_s, ___v = minus ((v), (c)->x)); \
841:63:    if (___s <= 0) \
842:64:        ___s = -1e-10; \
843:65:    (xs) = -(c)->f * dot ((c)->m_x, ___v) / ___s; \
844:66:    (ys) = -(c)->f * dot ((c)->m_y, ___v) / ___s; \
845:67:}
846:68:
847:69:/* camtophys(Camera *c1, Camera *c2, Vec v, double x1,
848:70:   double y1, double x2, double y2);  */
849:71:
850:72:#define cameratophys(c1,c2,v,x1,y1,x2,y2) { \
851:73:    Camera *___c[2]; \
852:74:    double ___x[2], ___y[2]; \
853:75:    ___x[0] = (x1); \
854:76:    ___x[1] = (x2); \
855:77:    ___y[0] = (y1); \
856:78:    ___y[1] = (y2); \
857:79:    ___c[0] = (c1); \
858:80:    ___c[1] = (c2); \
859:81:    (v) = triangulate_camera_point (___x, ___y, c, 2); \
860:82:}
861:83:
862:84:void camundestortmany (Camera * cam, double *x, double *y, int n);
863:85:
864:86:#endif
865:1:/*****************************************************************************************/
866:2:/* desktop.c - save and restore the desktop to *.dsk files                               */
867:3:/*****************************************************************************************/
868:4:
869:5:#include "display.h"
870:6:#include "widget3d.h"
871:7:#include "marker.h"
872:8:#include "displaycam.h"
873:9:#include "loadcalfile.h"
874:10:#include "imagehandler.h"
875:11:#include "picsetup.h"
876:12:#include "edit.h"
877:13:#include "dialog.h"
878:14:#include "loadfile.h"
879:15:
880:16:/*


 */
881:17:
882:18:static void strwrite (int f, const char *s)
883:19:{
884:20:    int len;
885:21:    if (!s) {
886:22:        len = -1;
887:23:        write (f, (char *) &len, sizeof (int));
888:24:    } else {
889:25:        len = strlen (s);
890:26:        write (f, (char *) &len, sizeof (int));
891:27:        write (f, s, len);
892:28:    }
893:29:}
894:30:
895:31:static char *strread (int f)
896:32:{
897:33:    int len;
898:34:    char *s;
899:35:    read (f, (char *) &len, sizeof (int));
900:36:    if (len < 0) {
901:37:        return 0;
902:38:    } else {
903:39:        s = Cmalloc (len + 1);
904:40:        read (f, s, len);
905:41:        s[len] = 0;
906:42:        return s;
907:43:    }
908:44:}
```

```
909:45:
910:46:
911:47:int do_save_desktop (Desktop * d, char *filename)
912:48:{
913:49:    int f;
914:50:    if ((f = creat (filename, 0644)) >= 0) {
915:51:        int i;
916:52:        Desktop save;
917:53:        memcpy (&save, d, sizeof (Desktop));
918:54:        save.cal_points = 0;
919:55:        save.cal_file = 0;
920:56:        save.temp_dir = save.image_dir = 0;
921:57:        for (i = 0; i < save.num_views; i++) {
922:58:            save.view[i].filename = 0;
923:59:            memset (&save.view[i].pic.main_image, 0, (unsigned long) \
924:60:                    (&save.view[i].pic.last_pointer) - (unsigned long) \
925:61:                    (&save.view[i].pic.main_image));
926:62:        }
927:63:        strwrite (f, "stereo\n - saved desktop\n\n");
928:64:        write (f, &save, sizeof (Desktop));
929:65:        write (f, d->cal_points, d->num_cal_points * sizeof (Vec));
930:66:        strwrite (f, d->cal_file);
931:67:        strwrite (f, d->temp_dir);
932:68:        strwrite (f, d->image_dir);
933:69:        if (d->num_views)
934:70:            for (i = 0; i < d->num_views; i++)
935:71:                strwrite (f, d->view[i].filename);
936:72:        close (f);
937:73:        return 0;
938:74:    } else {
939:75:        Cerrordialogue (CMain, 20, 20, " Save Desktop ", \
940:76:                        get_sys_error (" Error trying to save file. "));
941:77:    }
942:78:    return 1;
943:79:}
944:80:
945:81:
946:82:int save_desktop (Desktop * d)
947:83:{
948:84:    char *filename;
949:85:
950:86:    filename = Cgetfile (0, 0, 0, home_dir, "", " Save Desktop ");
951:87:
952:88:    if (filename)
953:89:        if (*filename)
954:90:            return do_save_desktop (d, filename);
955:91:    return 1;
956:92:}
957:93:
958:94:int setup_view (Desktop * d, char *filename, int x, int y, int i);
959:95:
960:96:int do_load_desktop (Desktop * d, char *filename)
961:97:{
962:98:    int f;
963:99:    if ((f = open (filename, O_RDONLY)) >= 0) {
964:100:        int i, x = 40, y = 40;
965:101:        char *sign;
966:102:        sign = strread (f);
967:103:        if (strcmp (sign, "stereo\n - saved desktop\n\n")) {
968:104:            close (f);
969:105:            Cerrordialogue (CMain, 20, 20, " Load Desktop ", \
970:106:                            " This is not a desktop file ");
971:107:            free (sign);
972:108:            return 1;
973:109:        }
974:110:        free (sign);
975:111:        for (i = 0; i < d->num_views; i++)
976:112:            destroy_view (&(d->view[i]));
977:113:        clear (d, Desktop);
978:114:        read (f, d, sizeof (Desktop));
979:115:        destroy ((void *) &d->cal_points);
980:116:        d->cal_points = Cmalloc (d->num_cal_points * sizeof (Vec));
981:117:        read (f, d->cal_points, d->num_cal_points * sizeof (Vec));
982:118:        d->cal_file = strread (f);
983:119:        d->temp_dir = strread (f);
984:120:        d->image_dir = strread (f);
985:121:        if (d->num_views)
986:122:            for (i = 0; i < d->num_views; i++) {
987:123:                char *v;
988:124:                v = strread (f);
```

```
989:125:                if (v) {
990:126:                    setup_view (d, v, x += 20, y += 20, i);
991:127:                    d->view[i].filename = v;
992:128:                }
993:129:            }
994:130:        draw_markers (d);
995:131:        close (f);
996:132:        return 0;
997:133:    } else {
998:134:        Cerrordialogue (CMain, 20, 20, " Load Desktop ", \
999:135:                        get_sys_error (" Error trying to save file. "));
1000:136:    }
1001:137:    return 1;
1002:138:}
1003:139:
1004:140:int load_desktop (Desktop * d)
1005:141:{
1006:142:    char *filename;
1007:143:
1008:144:    filename = Cgetfile (0, 0, 0, home_dir, "", " Load Desktop ");
1009:145:
1010:146:    if (filename)
1011:147:        if (*filename)
1012:148:            return do_load_desktop (d, filename);
1013:149:    return 1;
1014:150:}
1015:1:
1016:2:
1017:3:#ifndef _DESK_TOP_H
1018:4:#define _DESK_TOP_H
1019:5:
1020:6:int do_save_desktop (Desktop * d, char *filename);
1021:7:int do_load_desktop (Desktop * d, char *filename);
1022:8:int save_desktop (Desktop * d);
1023:9:int load_desktop (Desktop * d);
1024:10:
1025:11:#endif
1026:1:
1027:2:
1028:3:/*****************************************************************************************/
1029:4:/* display.c - main() function                                                           */
1030:5:/*****************************************************************************************/
1031:6:
1032:7:#include <config.h>
1033:8:#include "global.h"
1034:9:#include <stdio.h>
1035:10:#include <stdlib.h>
1036:11:
1037:12:#include <X11/Xlib.h>
1038:13:#include <X11/Xutil.h>
1039:14:#include <X11/keysym.h>
1040:15:
1041:16:#include "app_glob.c"
1042:17:#include "coolwidget.h"
1043:18:#include "imagewidget.h"
1044:19:
1045:20:#include "stringtools.h"
1046:21:#include "dirtools.h"
1047:22:#include "hugeimage.h"
1048:23:#include "widget3d.h"
1049:24:#include "edit.h"
1050:25:#include "camera.h"
1051:26:#include "fitline.h"
1052:27:
1053:28:#include <sys/types.h>
1054:29:#include <dirent.h>
1055:30:#include <string.h>
1056:31:#include <sys/stat.h>
1057:32:#include <unistd.h>
1058:33:#include <math.h>
1059:34:
1060:35:#include "loadfile.h"
1061:36:#include "display.h"
1062:37:#include "main/callback.h"
1063:38:#include "main/imagefit.h"
1064:39:#include "dialog.h"
1065:40:#include "matrix.h"
1066:41:#include "calibrate.h"
1067:42:#include "3dkit.h"
1068:43:#include "desktop.h"
```

```
1069:44:#include "mad.h"
1070:45:
1071:46:Desktop desktop;
1072:47:
1073:48:#define WIDTH3D 640
1074:49:#define HEIGHT3D 480
1075:50:
1076:51:/* options */
1077:52:
1078:53:char *editor_options_file = 0;
1079:54:
1080:55:char *option_display = 0;
1081:56:char *option_geometry = 0;
1082:57:char *option_background_color = 0;
1083:58:char *option_foreground_red = 0;
1084:59:char *option_foreground_green = 0;
1085:60:char *option_foreground_blue = 0;
1086:61:char *option_font = 0;
1087:62:
1088:63:
1089:64:int leave_out_calibration_points = 0;
1090:65:
1091:66:void goto_error (char *message)
1092:67:{
1093:68:
1094:69:}
1095:70:
1096:71:void init_desktop (Desktop * d)
1097:72:{
1098:73:    clear (d, Desktop);
1099:74:    d->image_dir = strdup (DATADIR);
1100:75:    d->temp_dir = strdup (TEMPDIR);
1101:76:}
1102:77:
1103:78:
1104:79:void stereo_init (char *name)
1105:80:{
1106:81:    CInitData stereo_startup =
1107:82:    {
1108:83:        20, 0, 0, 0, 0, 0, 0, 0,
1109:84:        0, 0, 0, 0, 0, 0, 0, 0,
1110:85:        0
1111:86:    };
1112:87:
1113:88:    stereo_startup.height_plus = 900;
1114:89:    stereo_startup.lines = 0;
1115:90:    stereo_startup.width_plus = 700;
1116:91:    stereo_startup.columns = 0;
1117:92:    stereo_startup.options |= CINIT_OPTION_USE_GREY;
1118:93:
1119:94:    stereo_startup.name = name;
1120:95:
1121:96:/* initialise: */
1122:97:    Cinit (&stereo_startup);
1123:98:}
1124:99:
1125:100:void main (int argc, char **argv)
1126:101:{
1127:102:    int menuopen = 0;
1128:103:    int x, y, x1;
1129:104:    CEvent cwevent;
1130:105:    XEvent xevent;
1131:106:    Window choicewin, editorwin, win3d;
1132:107:
1133:108:    init_desktop (&desktop);
1134:109:
1135:110:    stereo_init (argv[0]);
1136:111:
1137:112:    choicewin = Cdrawwindow ("choicewin", CMain, 0, 0, 10, 10, "");
1138:113:    Cgethintpos (&x, &y);
1139:114:    x1 = x;
1140:115:    Cdrawbutton ("load", choicewin, x, y, AUTO_SIZE,
1141:116:                 " Load Image ");
1142:117:    Cgethintpos (&x, 0);
1143:118:    Cdrawbutton ("loadcal", choicewin, x, y, AUTO_SIZE,
1144:119:                 " Load Calibration File ");
1145:120:    Cgethintpos (&x, 0);
1146:121:    Cdrawbutton ("showcal", choicewin, x, y, AUTO_SIZE,
1147:122:                 " Display Calibration Points ");
1148:123:    Cgethintpos (0, &y);
```

```
1149:124:    x = x1;
1150:125:    Cdrawbutton ("loadd", choicewin, x, y, AUTO_SIZE,
1151:126:                 " Load Previous Desktop ");
1152:127:    Cgethintpos (&x, 0);
1153:128:    Cdrawbutton ("saved", choicewin, x, y, AUTO_SIZE,
1154:129:                 " Save Previous Desktop ");
1155:130:    Cgethintpos (&x, 0);
1156:131:    Cdrawbutton ("finish", choicewin, x, y, AUTO_SIZE,
1157:132:                 " Quit ");
1158:133:    Cgethintpos (0, &y);
1159:134:    x = x1;
1160:135:    Cdrawbutton ("getpnt", choicewin, x, y, AUTO_SIZE,
1161:136:                 " Triangulate Point        ");
1162:137:    Cgethintpos (&x, 0);
1163:138:    Cdrawbutton ("getsurf", choicewin, x, y, AUTO_SIZE,
1164:139:                 " Triangulate Multiple Points (Surface)   ");
1165:140:    Cgethintpos (0, &y);
1166:141:    x = x1;
1167:142:    Cdrawbutton ("getline", choicewin, x, y, AUTO_SIZE,
1168:143:                 " Fit Line from Points     ");
1169:144:    Cgethintpos (&x, 0);
1170:145:    Cdrawbutton ("getlineedge", choicewin, x, y, AUTO_SIZE,
1171:146:                 " Fit Line from two Line Projections      ");
1172:147:    Cgethintpos (0, &y);
1173:148:    x = x1;
1174:149:    Cdrawbutton ("getcyl", choicewin, x, y, AUTO_SIZE,
1175:150:                 " Fit Cylinder from Points ");
1176:151:    Cgethintpos (&x, 0);
1177:152:    Cdrawbutton ("getcyle", choicewin, x, y, AUTO_SIZE,
1178:153:                 " Fit Cylinder from four Line Projections ");
1179:154:    Cgethintpos (0, &y);
1180:155:    x = x1;
1181:156:    Cdrawbutton ("getcirc", choicewin, x, y, AUTO_SIZE,
1182:157:                 " Fit Circle from Points   ");
1183:158:    Cgethintpos (&x, 0);
1184:159:    Cdrawbutton ("getcircedge", choicewin, x, y, AUTO_SIZE,
1185:160:                 " Fit Circle from Ellipses Projections    ");
1186:161:    Cgethintpos (0, &y);
1187:162:    x = x1;
1188:163:    Cdrawbutton ("getellipse", choicewin, x, y, AUTO_SIZE,
1189:164:                 " Fit Ellipse from Points  ");
1190:165:    Cgethintpos (&x, 0);
1191:166:
1192:167:    Csetsizehintpos ("choicewin");
1193:168:
1194:169:    editorwin = Cdrawwindow ("editorwin", CMain, 200, 100, 640 + 18, \
1195:170:                             450 + 40 + 18 + TEXT_PIX_PER_LINE + 11, "");
1196:171:    Cdraweditor ("editor", editorwin, 6, 6 + 40,
1197:172:              640, 450, "", 0, "/export/home/psheer/stereo/src/scrap/");
1198:173:
1199:174:    CDrawEditMenuButtons ("em", editorwin, Cwidget ("editor")->winid, 10, 10);
1200:175:
1201:176:    Caddcallback ("editor", cb_editor);
1202:177:
1203:178:
1204:179:    Cdraw3dobject ("3dview", win3d = Cdrawwindow ("3dplanewin",
1205:180:                CMain, 150, 150, WIDTH3D + 100, 16 + HEIGHT3D, ""), 6, 6,
1206:181:                   WIDTH3D, HEIGHT3D, 1, 256);
1207:182:
1208:183:    Cdrawbutton ("plshr", win3d, WIDTH3D + 20, 10, 70, 20, "Shrink");
1209:184:    Cdrawbutton ("plenl", win3d, WIDTH3D + 20, 40, 70, 20, "Enlarge");
1210:185:    Cdrawbutton ("newrender", win3d, WIDTH3D + 20, 70, 70, 20, "New render");
1211:186:    Cdrawbutton ("density", win3d, WIDTH3D + 20, 100, 70, 20, "Density");
1212:187:    Cdrawbutton ("flattri", win3d, WIDTH3D + 20, 130, 70, 20, "Flat Triangle");
1213:188:    Cdrawbutton ("savewin", win3d, WIDTH3D + 20, 160, 70, 20, "Save Window");
1214:189:
1215:190:
1216:191:    Caddcallback ("getpnt", cb_getpoint);
1217:192:    Caddcallback ("getellipse", cb_getellipse);
1218:193:    Caddcallback ("getcircedge", cb_getcircleedge);
1219:194:
1220:195:    Caddcallback ("load", cb_newimage);
1221:196:
1222:197:    Caddcallback ("3dview", cb_3dplane);
1223:198:    Caddcallback ("newrender", cb_newrender);
1224:199:
1225:200:
1226:201:    Caddcallback ("getsurf", cb_getsurface);
1227:202:    Caddcallback ("getline", cb_getline);
1228:203:    Caddcallback ("getcirc", cb_getcircle);
```

```
1229:204:    Caddcallback ("getcyle", cb_getcylinderedge);
1230:205:    Caddcallback ("getcyl", cb_getcylinder);
1231:206:    Caddcallback ("getlineedge", cb_getlineedge);
1232:207:
1233:208:    Caddcallback ("loadcal", cb_loadcal);
1234:209:    Caddcallback ("showcal", cb_showcal);
1235:210:    Caddcallback ("saved", cb_save_desktop);
1236:211:    Caddcallback ("loadd", cb_load_desktop);
1237:212:
1238:213:    Caddcallback ("savewin", cb_save_window);
1239:214:
1240:215:    if (argc > 1)
1241:216:        if (*argv[1] != '-')
1242:217:            do_load_desktop (&desktop, argv[1]);
1243:218:
1244:219:    do {
1245:220:        CNextEvent (&xevent, &cwevent);
1246:221:
1247:222:        if (!strcmp (cwevent.ident, "plshr")) {
1248:223:            Cwidget ("3dview")->solid->distance += 2400;
1249:224:            Cwidget ("3dview")->solid->y_cam += 2400;
1250:225:            Credraw3dobject ("3dview", 0);
1251:226:        }
1252:227:        if (!strcmp (cwevent.ident, "plenl")) {
1253:228:            Cwidget ("3dview")->solid->distance -= 2400;
1254:229:            Cwidget ("3dview")->solid->y_cam -= 2400;
1255:230:            Credraw3dobject ("3dview", 0);
1256:231:        }
1257:232:        if (!strcmp (cwevent.ident, "flattri")) {
1258:233:            CWidget *w = Cwidget ("3dview");
1259:234:            if (w->solid->option_flags & TDOPTION_FLAT_TRIANGLE) {
1260:235:                w->solid->option_flags &= 0xFFFFFFFF - TDOPTION_FLAT_TRIANGLE;
1261:236:            } else {
1262:237:                w->solid->option_flags |= TDOPTION_FLAT_TRIANGLE;
1263:238:            }
1264:239:            Credraw3dobject ("3dview", 0);
1265:240:        }
1266:241:        if (!strcmp (cwevent.ident, "kill")) {
1267:242:            menuopen = 0;
1268:243:            Cundrawwidget ("win1");
1269:244:        }
1270:245:    } while (strcmp (cwevent.ident, "finish"));
1271:246:
1272:247:    Cundrawall ();
1273:248:    mad_finalize (__FILE__, __LINE__);
1274:249:
1275:250:}
1276:1:
1277:2:#ifndef _DISPLAY_H
1278:3:#define _DISPLAY_H
1279:4:
1280:5:#include <config.h>
1281:6:#include "global.h"
1282:7:#include <stdlib.h>
1283:8:#include <stdio.h>
1284:9:#include <sys/types.h>
1285:10:
1286:11:#ifdef HAVE_UNISTD_H
1287:12:#include <unistd.h>
1288:13:#endif
1289:14:
1290:15:#include <my_string.h>
1291:16:#include <sys/stat.h>
1292:17:
1293:18:#ifdef HAVE_FCNTL_H
1294:19:#include <fcntl.h>
1295:20:#endif
1296:21:
1297:22:#include <stdlib.h>
1298:23:#include <stdarg.h>
1299:24:
1300:25:#ifdef HAVE_SYS_TIME_H
1301:26:#include <sys/time.h>
1302:27:#endif
1303:28:
1304:29:#include "regex.h"
1305:30:
1306:31:#include <X11/Xlib.h>
1307:32:#include <X11/Xutil.h>
1308:33:#include <X11/keysym.h>
```

```
1309:34:#include "coolwidget.h"
1310:35:
1311:36:#include "camera.h"
1312:37:#include "fitline.h"
1313:38:
1314:39:
1315:40:struct picwithzoom {
1316:41:    CWidget *main_image;        /* thumbnail image */
1317:42:    CWidget *zoom_image;        /* zoombox image */
1318:43:    CWidget *main_win;          /* window of thumbnail image */
1319:44:    CWidget *zoom_win;          /* window of zoombox image */
1320:45:    CWidget *main_rect;         /* zoombox rectagle rectangle
1321:46:                                   in thumnail image */
1322:47:    CWidget *zoom_rect;         /* ..in zoom image */
1323:48:    CWidget *main_markers;      /* markers in thumbnail */
1324:49:    CWidget *zoom_markers;      /* markers in zoombox */
1325:50:
1326:51:
1327:52:    CWidget *Ttext;             /* text display of image info */
1328:53:    CWidget *Binfo;             /* display cam info button */
1329:54:    CWidget *Blast;             /* remove last marker button */
1330:55:    CWidget *Ball;              /* remove all markers button */
1331:56:    CWidget *Bcalibrate;        /* calibrate camera button */
1332:57:    CWidget *Bkill;             /* kill this window button */
1333:58:
1334:59:    CWidget *Bleave;            /* for later use */
1335:60:    CWidget *Tleave;
1336:61:    CWidget *Ssig;
1337:62:    CWidget *Isigma;
1338:63:    CWidget *B5;
1339:64:    int last_pointer;
1340:65:
1341:66:    long width, height;         /* width and height of the thumbnail image */
1342:67:    double real_width, real_height;  /* width and height of the original image */
1343:68:    double x0, y0;              /* image centre */
1344:69:    int zwidth, zheight;        /* width and height of the zoomed area */
1345:70:    long xzoom, yzoom;          /* position of the zoomed area */
1346:71:    int mag;                    /* zoombox magnification */
1347:72:};
1348:73:
1349:74:typedef struct picwithzoom Picture;
1350:75:
1351:76:#define MAX_NUM_VIEWS 32
1352:77:#define MAX_NUM_MARKS 256
1353:78:
1354:79:typedef struct marks Mark;
1355:80:
1356:81:struct view {
1357:82:    Picture pic;
1358:83:    char *filename;             /* a tiff or targa file. 0 if non-existant */
1359:84:    Camera cam;                 /* camera data structure */
1360:85:    int calibrated;             /* always 1 if calibrated */
1361:86:    Vec mark[MAX_NUM_MARKS];
1362:87:    int num_marks;
1363:88:};
1364:89:
1365:90:typedef struct view View;
1366:91:
1367:92:struct desktop {
1368:93:    View view[MAX_NUM_VIEWS];
1369:94:    Vec *cal_points;
1370:95:    int num_cal_points;
1371:96:    int leave_out_calibration_points;
1372:97:    int optimise_sigma;
1373:98:    char *cal_file;
1374:99:    int current_view;
1375:100:     int num_views;
1376:101:     char *temp_dir;
1377:102:     char *image_dir;
1378:103:     Vec centre_offset_for_3d;
1379:104:     double scale_units_for_3d;
1380:105:};
1381:106:
1382:107:typedef struct desktop Desktop;
1383:108:
1384:109:
1385:110:int load_calibration_points (Desktop * d);
1386:111:int save_calibration_points (Desktop * d);
1387:112:int load_new_view (Desktop * d);
1388:113:/* int save_view(Desktop *d); */
```

```
1389:114:
1390:115:/* fits an object (see fitline.h) to the markers */
1391:116:int fit_object (Desktop * d, Object * object, int type);
1392:117:
1393:118:#define ZOOMSIZE (192*2)
1394:119:
1395:120:#define THESE_2D_PROJECTIONS_HAVE_BEEN_ADJUSTED_FOR_LENS_DISTORTION YES
1396:121:
1397:122:#define MAX_CAP 1024
1398:123:
1399:124:#define MONITOR_GAMMA 1.2
1400:125:
1401:126:#define MAX_CAL_FILE_SIZE 65536
1402:127:
1403:128:void Dcoordtotext (char *widgetevent, char *strx, char *stry, \
1404:129:                   struct picwithzoom *image);
1405:130:
1406:131:int Cundrawrectangle (const char *ident);
1407:132:
1408:133:int Cdrawrectangle (const char *ident, int x, int y, int w, \
1409:134:                    int h, unsigned long color);
1410:135:
1411:136:void Dcheckinsertcoordfromimage (CEvent * cwevent, const char *ident, \
1412:137:                                 struct picwithzoom *image, int p);
1413:138:
1414:139:void Dimagecoord (CEvent * cwevent, double *x, double *y, struct picwithzoom *image);
1415:140:
1416:141:#endif                          /* _DISPLAY_H */
1417:1:/*****************************************************************************************/
1418:2:/* displaycam.c - draw a list of the camera data                                         */
1419:3:/*****************************************************************************************/
1420:4:
1421:5:#include "display.h"
1422:6:#include "main/imagefit.h"
1423:7:#include "app_glob.c"
1424:8:#include "hugeimage.h"
1425:9:#include "widget3d.h"
1426:10:#include "main/marker.h"
1427:11:#include "main/displaycam.h"
1428:12:#include "picsetup.h"
1429:13:#include "dialog.h"
1430:14:#include "stringtools.h"
1431:15:#include "callback.h"
1432:16:#include "matrix.h"
1433:17:
1434:18:
1435:19:#include "bitmap/camera.bitmap"
1436:20:#include "bitmap/camphi.bitmap"
1437:21:#include "bitmap/camtheta.bitmap"
1438:22:#include "bitmap/campsi.bitmap"
1439:23:#include "bitmap/camf.bitmap"
1440:24:#include "bitmap/camsig.bitmap"
1441:25:#include "bitmap/camxy.bitmap"
1442:26:#include "bitmap/camz.bitmap"
1443:27:
1444:28:/*


 */
1445:29:
1446:30:void draw_camera_data_text (Camera * c, Window win, int x, int y)
1447:31:{
1448:32:    if (!Ci ("caldatabitmaps")) {
1449:33:        Cdrawbitmap ("caldatabitmaps", win, x, y,
1450:34:                     40, 32, Ccolor (1), Ccolor (25), camphi_bits);
1451:35:        Cdrawbitmap ("", win, x, y + 50,
1452:36:                     40, 32, Ccolor (1), Ccolor (25), camtheta_bits);
1453:37:        Cdrawbitmap ("", win, x, y + 100,
1454:38:                     40, 32, Ccolor (1), Ccolor (25), campsi_bits);
1455:39:        Cdrawbitmap ("", win, x, y + 150,
1456:40:                     40, 32, Ccolor (1), Ccolor (25), camf_bits);
1457:41:        Cdrawbitmap ("", win, x, y + 200,
1458:42:                     40, 40, Ccolor (1), Ccolor (25), camxy_bits);
1459:43:        Cdrawbitmap ("", win, x, y + 258,
1460:44:                     40, 20, Ccolor (1), Ccolor (25), camz_bits);
1461:45:        Cdrawbitmap ("", win, x, y + 296,
1462:46:                     40, 32, Ccolor (1), Ccolor (25), camsig_bits);
1463:47:
1464:48:        y += 10;
1465:49:        x += 55;
```

```
1466:50:
1467:51:        Cdrawtext ("tcalphi", win, x, y, " %15.2f ", c->phi * 180 / PI);
1468:52:        Cdrawtext ("tcaltheta", win, x, y + 50, " %15.2f ", c->theta * 180 / PI);
1469:53:        Cdrawtext ("tcaltsi", win, x, y + 100, " %15.2f ", c->tsi * 180 / PI);
1470:54:        Cdrawtext ("tcalf", win, x, y + 150, " %15.8g ", c->f);
1471:55:        Cdrawtext ("tcalx", win, x, y + 192, " %15.8g ", c->x.x);
1472:56:        Cdrawtext ("tcaly", win, x, y + 216, " %15.8g ", c->x.y);
1473:57:        Cdrawtext ("tcalz", win, x, y + 252, " %15.8g ", c->x.z);
1474:58:        Cdrawtext ("tcalsig", win, x, y + 296, " %15.5g ", c->sig);
1475:59:        Cdrawtext ("tcale", win, 10, y + 340, \
1476:60:                   " Sum squared error in pixels: \n %15.5g ", c->e);
1477:61:
1478:62:    } else {                    /*merely update the text */
1479:63:        y += 10;
1480:64:        x += 55;
1481:65:
1482:66:        Credrawtext ("tcalphi", " %15.2f ", c->phi * 180 / PI);
1483:67:        Credrawtext ("tcaltheta", " %15.2f ", c->theta * 180 / PI);
1484:68:        Credrawtext ("tcaltsi", " %15.2f ", c->tsi * 180 / PI);
1485:69:        Credrawtext ("tcalf", " %15.8g ", c->f);
1486:70:        Credrawtext ("tcalx", " %15.8g ", c->x.x);
1487:71:        Credrawtext ("tcaly", " %15.8g ", c->x.y);
1488:72:        Credrawtext ("tcalz", " %15.8g ", c->x.z);
1489:73:        Credrawtext ("tcalsig", " %15.5g ", c->sig);
1490:74:    }
1491:75:}
1492:76:
1493:77:
1494:78:void draw_calibrarion_data_text (Desktop * d, Window win, int x, int y)
1495:79:{
1496:80:    int i;
1497:81:    if (d->num_cal_points)
1498:82:        for (i = 0; i < d->num_cal_points; i++)
1499:83:            Cdrawtext (catstrs ("calpoint", itoa (i), 0), win, x, y + 30 * i, \
1500:84:                       " %3d: %15f %15f %15f ", i + 1, \
1501:85:             d->cal_points[i].x, d->cal_points[i].y, d->cal_points[i].z);
1502:86:    else
1503:87:        Cdrawtext ("calpoint", win, x, y, " Data not loaded ");
1504:88:}
1505:89:
1506:90:
1507:91:void draw_camera_data_window (Camera * c)
1508:92:{
1509:93:    Window win;
1510:94:    static isopen = 0;
1511:95:
1512:96:    if (isopen) {
1513:97:        Cundrawwidget ("camdatawin");
1514:98:        isopen = 0;
1515:99:    } else {
1516:100:        win = Cdrawwindow ("camdatawin", CMain, 20, 20, 300, 410, "caldatawin");
1517:101:        draw_camera_data_text (c, win, 10, 10);
1518:102:        isopen = 1;
1519:103:    }
1520:104:}
1521:105:
1522:106:void draw_camera_data (Desktop * d, CEvent * e)
1523:107:{
1524:108:    set_current_from_pointer (d, e);
1525:109:    if (d->num_views) {
1526:110:        draw_camera_data_window (&d->view[d->current_view].cam);
1527:111:    }
1528:112:}
1529:113:
1530:114:void show_cal_points (Desktop * d, CEvent * e)
1531:115:{
1532:116:    Window win;
1533:117:    static int isopen = 0;
1534:118:
1535:119:    if (isopen) {
1536:120:        Cundrawwidget ("caldatawin");
1537:121:        isopen = 0;
1538:122:    } else {
1539:123:        int x, y;
1540:124:        win = Cdrawheadedwindow ("caldatawin", CMain, 20, 20, 460, \
1541:125:                (d->num_cal_points ? d->num_cal_points : 1) * 30 + 10, \
1542:126:                                 " Calibration Points ");
1543:127:        Cgethintpos (&x, &y);
1544:128:        draw_calibrarion_data_text (d, win, x, y);
1545:129:        Csetsizehintpos ("caldatawin");
```

```
1546:130:        isopen = 1;
1547:131:    }
1548:132:}
1549:1:
1550:2:#ifndef _DISPLAY_CAM_H
1551:3:#define _DISPLAY_CAM_H
1552:4:
1553:5:void draw_camera_data_window (Camera * c);
1554:6:void draw_camera_data (Desktop * d, CEvent * e);
1555:7:void show_cal_points (Desktop * d, CEvent * ce);
1556:8:
1557:9:#endif
1558:1:/*****************************************************************************************/
1559:2:/* fitline.c - line, circle, ellipse, plane, and cylinder fitting routines               */
1560:3:/*****************************************************************************************/
1561:4:
1562:5:#include <config.h>
1563:6:#include "global.h"
1564:7:#include <stdlib.h>
1565:8:#include <string.h>
1566:9:#include <stdio.h>
1567:10:#include <math.h>
1568:11:#include "matrix.h"
1569:12:#include "camera.h"
1570:13:#include "simplex.h"
1571:14:#include "fitline.h"
1572:15:#include "quickmath.h"
1573:16:
1574:17:/*


 */
1575:18:
1576:19:/* fast gauss-jordan elimination to diagonalise a 3x3 matrix
1577:20:   with partial pivotting */
1578:21:Vec solve3x3 (double a[4][3])
1579:22:{
1580:23:    Vec v =
1581:24:    {10e20, 10e20, 10e20};
1582:25:    int i, j, k, max;
1583:26:    double temp, r;
1584:27:    for (i = 0; i < 3; i++) {
1585:28:        max = i;
1586:29:        for (j = i + 1; j < 3; j++) {
1587:30:            if (fabs (a[i][j]) >= fabs (a[i][max]))
1588:31:                max = j;
1589:32:        }
1590:33:        for (k = i; k < 4; k++) {
1591:34:            temp = a[k][i];
1592:35:            a[k][i] = a[k][max];
1593:36:            a[k][max] = temp;
1594:37:        }
1595:38:        for (j = i + 1; j < 3; j++) {
1596:39:            if (!a[i][i])
1597:40:                return v;
1598:41:            r = a[i][j] / a[i][i];
1599:42:            for (k = 3; k >= i; k--)
1600:43:                a[k][j] -= a[k][i] * r;
1601:44:        }
1602:45:    }
1603:46:    for (i = 2; i > 0; i--) {
1604:47:        if (!a[i][i])
1605:48:            return v;
1606:49:        r = 1 / a[i][i] * a[3][i];
1607:50:        for (j = i - 1; j >= 0; j--) {
1608:51:            a[3][j] -= a[i][j] * r;
1609:52:            a[i][j] = 0;
1610:53:        }
1611:54:    }
1612:55:    if (!a[0][0] || !a[1][1] || !a[2][2])
1613:56:        return v;
1614:57:    v.x = a[3][0] / a[0][0];
1615:58:    v.y = a[3][1] / a[1][1];
1616:59:    v.z = a[3][2] / a[2][2];
1617:60:    return v;
1618:61:}
1619:62:
1620:63:
1621:64:static inline Vec screen (Camera * cam, Vec p)
1622:65:{
```

```
1623:66:    Vec v;
1624:67:    v.z = 0;
1625:68:    phystocamera (cam, p, v.x, v.y);
1626:69:    return v;
1627:70:}
1628:71:
1629:72:
1630:73:/* fitline fits a 2D line to 'numpoints' points whose x and y values
1631:74:   are given as arrays xvals and yvals.
1632:75:
1633:76:   returns a, b and c of the line
1634:77:   a*x + b*y = c
1635:78:   in aa, ab, and ac.
1636:79:
1637:80:   return rules:
1638:81:   a^2 + b^2 = 1
1639:82:   b >= 0
1640:83:
1641:84:   errors are minimised normal to the line no-matter what the line's
1642:85:   orientation.
1643:86:   Fitline returns error.
1644:87: */
1645:88:double fitline (double *xvals, double *yvals, int numpoints, \
1646:89:                double *aa, double *ab, double *ac)
1647:90:{
1648:91:    double e1, e2, a, b, c, p, q, r, s, t, m1, m2, c1, c2;
1649:92:    double sumx = 0, sumy = 0, sumxy = 0, sumxx = 0, sumyy = 0;
1650:93:    int i;
1651:94:
1652:95:    for (i = 0; i < numpoints; i++) {
1653:96:        sumx += xvals[i];
1654:97:        sumy += yvals[i];
1655:98:        sumxy += xvals[i] * yvals[i];
1656:99:        sumxx += xvals[i] * xvals[i];
1657:100:        sumyy += yvals[i] * yvals[i];
1658:101:    }
1659:102:
1660:103:/*Need to divide by 'numpoints' because of an algeabraic error. Usually 'numpoints' */
1661:104:/*would be included in the minimisation expression. */
1662:105:
1663:106:    p = sumx / numpoints;
1664:107:    q = sumy / numpoints;
1665:108:    r = sumxx / numpoints;
1666:109:    s = sumyy / numpoints;
1667:110:    t = sumxy / numpoints;
1668:111:
1669:112:/*coefficents of quadratic in m, (a m^2 + b m + c = 0): */
1670:113:
1671:114:    a = (double) p *q - t;
1672:115:    b = (double) p *p - (q * q + r - s);
1673:116:    c = (double) -a;
1674:117:
1675:118:    if (fabs (b) < fabs (10000 * a)) {
1676:119:        /*Indicating a not to steep slope.
1677:120:          Otherwise minimisation must be handled differently. */
1678:121:
1679:122:/*There are two solutions; one for the maximum and one for minimum: */
1680:123:        m1 = (-b - sqrt (b * b - 4 * a * c)) / (2 * a);
1681:124:        m2 = (-b + sqrt (b * b - 4 * a * c)) / (2 * a);
1682:125:        c1 = q - m1 * p;
1683:126:        c2 = q - m2 * p;
1684:127:
1685:128:/*To determine the min from the max, check the actually error: */
1686:129:        e1 = (double) ((double) c1 * c1 + 2 * m1 * c1 * p + \
1687:130:             m1 * m1 * r - 2 * c1 * q - 2 * m1 * t + s) / (m1 * m1 + 1);
1688:131:        e2 = (double) ((double) c2 * c2 + 2 * m2 * c2 * p + \
1689:132:             m2 * m2 * r - 2 * c2 * q - 2 * m2 * t + s) / (m2 * m2 + 1);
1690:133:
1691:134:        if (e1 < e2) {
1692:135:            m2 = m1;
1693:136:            c2 = c1;
1694:137:            e2 = e1;
1695:138:        }
1696:139:/*Return coefficients (a*x + b*y = c): */
1697:140:        if (fabs (m2) > 1) {
1698:141:            *ab = (double) 1 / m2;
1699:142:            *aa = (double) -1;
1700:143:            *ac = (double) c2 / m2;
1701:144:        } else {
1702:145:            *ab = 1;
```

```
1703:146:            *aa = -m2;
1704:147:            *ac = c2;
1705:148:        }
1706:149:    } else {
1707:150:
1708:151:/*For very steep lines (either close to vertical or close to horizontal), */
1709:152:/*use usual least squares fit, reversing x and y axis if */
1710:153:/*vertical. */
1711:154:
1712:155:        if ((sumxx - numpoints * p * p) > (sumyy - numpoints * q * q)) {
1713:156:            /* gives an indication of vertical or horizontal. */
1714:157:            *ac = ((double) sumxx * sumy - sumxy * sumx) / \
1715:158:                (numpoints * sumxx - sumx * sumx);
1716:159:            *aa = -((double) numpoints * sumxy - sumx * sumy) / \
1717:160:                (numpoints * sumxx - sumx * sumx);
1718:161:            *ab = 1;
1719:162:            e2 = sumyy + (*aa) * (*aa) * sumxx + 2 * (*ac) * (*aa) * sumx
1720:163:                + (*ac) * (*ac) + 2 * (*aa) * sumxy + 2 * (*ac) * sumy;
1721:164:            /*almost horizontal line */
1722:165:        } else {
1723:166:            *ac = ((double) sumyy * sumx - (double) sumxy * sumy) / \
1724:167:                ((double) numpoints * sumyy - (double) sumy * sumy);
1725:168:            *ab = -((double) numpoints * sumxy - (double) sumy * sumx) / \
1726:169:                ((double) numpoints * sumyy - (double) sumy * sumy);
1727:170:            *aa = 1;
1728:171:/*****check this error value as it seldom happens*/
1729:172:            e2 = sumxx + (*ab) * (*ab) * sumyy + 2 * (*ac) * (*ab) * sumy
1730:173:                + (*ac) * (*ac) + 2 * (*ab) * sumxy + 2 * (*ac) * sumx;
1731:174:            /*almost vertical line */
1732:175:        }
1733:176:    }
1734:177:
1735:178:    r = sqrt (sqr (*aa) + sqr (*ab));
1736:179:    if (*ab < 0)
1737:180:        r = -r;
1738:181:    *aa /= r;
1739:182:    *ab /= r;
1740:183:    *ac /= r;
1741:184:
1742:185:    return e2;
1743:186:}
1744:187:
1745:188:
1746:189:
1747:190:/*
1748:191:   Given 3D point coords this calcs the SS error with the least 3D line in
1749:192:   the direction (u,v,w). Returns the offset of where the first point is
1750:193:   closest to the line in (a,b,c). (u,v,w) is the vector
1751:194:   between where the first and last points are closest
1752:195:   to the line. The first and last points must
1753:196:   not coincide.
1754:197: */
1755:198:
1756:199:double error3dline (double *xp, double *yp, double *zp,
1757:200:                    int num_points,
1758:201:                    double u, double v, double w,
1759:202:                    double *a, double *b, double *c)
1760:203:{
1761:204:    Matrix *A;
1762:205:    double d = 0, e = 0, f = 0, g = 0, h = 0, i = 0, j = 0, k = 0, l = 0;
1763:206:    double m, n, o, x, y, z;
1764:207:    double r;
1765:208:    int t;
1766:209:
1767:210:    r = u * u + v * v + w * w;
1768:211:
1769:212:    if (r) {
1770:213:        if (num_points > 2) {
1771:214:            for (t = 1; t < num_points - 1; t++) {
1772:215:                d += xp[t] * yp[t];
1773:216:                e += yp[t] * zp[t];
1774:217:                f += xp[t] * zp[t];
1775:218:                g += xp[t];
1776:219:                h += yp[t];
1777:220:                i += zp[t];
1778:221:                j += xp[t] * xp[t];
1779:222:                k += yp[t] * yp[t];
1780:223:                l += zp[t] * zp[t];
1781:224:            }
1782:225:        }
```

```
1783:226:        m = xp[num_points - 1];
1784:227:        n = yp[num_points - 1];
1785:228:        o = zp[num_points - 1];
1786:229:        x = xp[0];
1787:230:        y = yp[0];
1788:231:        z = zp[0];
1789:232:
1790:233:        num_points -= 2;
1791:234:        A = Madoublestomatrix (3, 4, (v * v + w * w) * num_points + 2 * r,
1792:235:                                -u * v * num_points,
1793:236:                                -u * w * num_points,
1794:237:           h * u * v + i * u * w - g * (v * v + w * w) - (m + x - u) * r,
1795:238:                                -u * v * num_points,
1796:239:                                (u * u + w * w) * num_points + 2 * r,
1797:240:                                -v * w * num_points,
1798:241:           g * u * v + i * v * w - h * (u * u + w * w) - (n + y - v) * r,
1799:242:                                -u * w * num_points,
1800:243:                                -v * w * num_points,
1801:244:                                (u * u + v * v) * num_points + 2 * r,
1802:245:         g * u * w + h * v * w - i * (u * u + v * v) - (o + z - w) * r);
1803:246:
1804:247:        if (Madiag (A)) {
1805:248:
1806:249:            (*a) = -Mard ((*A), 0, 3);
1807:250:            (*b) = -Mard ((*A), 1, 3);
1808:251:            (*c) = -Mard ((*A), 2, 3);
1809:252:
1810:253:            Mafreematrix (A);
1811:254:
1812:255:            e = sqr (u + (*a) - m) + sqr (v + (*b) - n) + sqr (w + (*c) - o)
1813:256:                + sqr ((*a) - x) + sqr ((*b) - y) + sqr ((*c) - z)
1814:257:                - (sqr (u) * j + 2 * u * v * d + 2 * u * w * f - 2 * \
1815:258:                   u * g * ((*a) * u + (*b) * v + (*c) * w)
1816:259:                   + sqr (v) * k + 2 * v * w * e - 2 * v * h * \
1817:260:                   ((*a) * u + (*b) * v + (*c) * w)
1818:261:                + sqr (w) * l - 2 * w * i * ((*a) * u + (*b) * v + (*c) * w)
1819:262:                + num_points * (pow ((*a) * u + (*b) * v + (*c) * w, 2))) / r
1820:263:                + j - 2 * (*a) * g + k - 2 * (*b) * h + l - 2 * (*c) * i
1821:264:                + (sqr (*a) + sqr (*b) + sqr (*c)) * num_points;
1822:265:
1823:266:            return e;
1824:267:        }
1825:268:        Mafreematrix (A);
1826:269:    }
1827:270:    fprintf (stderr, "stereo:%s:%d: in call to error3dline()\n", __FILE__, __LINE__);
1828:271:    return 10e90;
1829:272:}
1830:273:
1831:274:
1832:275:
1833:276:static double *x_points;
1834:277:static double *y_points;
1835:278:static double *z_points;
1836:279:static int num_3d_points;
1837:280:static double c_x, c_y, c_z;
1838:281:
1839:282:
1840:283:static double tominimise3dline (double *u)
1841:284:{
1842:285:    return error3dline (x_points, y_points, z_points, num_3d_points,
1843:286:                        u[0], u[1], u[2], &c_x, &c_y, &c_z);
1844:287:}
1845:288:
1846:289:
1847:290:LineSegment linesegment (Vec p1, Vec p2)
1848:291:{
1849:292:    LineSegment l;
1850:293:    memset (&l, 0, sizeof (LineSegment));
1851:294:    l.p1 = p1;
1852:295:    l.p2 = p2;
1853:296:    l.e = 0;
1854:297:    l.u = minus (p2, p1);
1855:298:    l.l = norm (l.u);
1856:299:    l.u = times (l.u, 1 / l.l);
1857:300:    l.psi = atan2 (l.u.y, l.u.x);
1858:301:    l.beta = atan2 (l.u.z, sqrt (sqr (l.u.x) + sqr (l.u.y)));
1859:302:    l.m = times (plus (l.p1, l.p2), 0.5);
1860:303:    l.a = -l.u.y;
1861:304:    l.b = l.u.x;
1862:305:    l.c = l.a * l.p1.x + l.b * l.p1.y;
```

```
1863:306:    l.type = LINE_SEGMENT;
1864:307:    return l;
1865:308:}
1866:309:
1867:310:
1868:311:
1869:312:
1870:313:/* this fits a line to numpoints 2D or 3D points and returns a lineseg structure */
1871:314:LineSegment linesegmentfrompoints (double *xvals, double *yvals, \
1872:315:                                    double *zvals, int numpoints)
1873:316:{
1874:317:    LineSegment l;
1875:318:    int i;
1876:319:    double vmax = -10e90, vmin = 10e90, len;
1877:320:    Vec v0, vi;
1878:321:    memset (&l, 0, sizeof (LineSegment));
1879:322:    l.n = numpoints;
1880:323:
1881:324:    if (zvals) {
1882:325:        double result[3];
1883:326:
1884:327:        num_3d_points = numpoints;
1885:328:        x_points = xvals;
1886:329:        y_points = yvals;
1887:330:        z_points = zvals;
1888:331:
1889:332:        result[0] = 1;          /* initial guess */
1890:333:        result[1] = 1;
1891:334:        result[2] = 1;
1892:335:
1893:336:        simplex_optimise (result, 3, 10e-7, 1, tominimise3dline, NULL);
1894:337:
1895:338:        l.e = tominimise3dline (result);
1896:339:        l.u.x = result[0];
1897:340:        l.u.y = result[1];
1898:341:        l.u.z = result[2];
1899:342:        l.u = times (l.u, 1 / norm (l.u));
1900:343:        v0.x = c_x;
1901:344:        v0.y = c_y;
1902:345:        v0.z = c_z;
1903:346:        for (i = 0; i < l.n; i++) {
1904:347:            vi.x = xvals[i];
1905:348:            vi.y = yvals[i];
1906:349:            vi.z = zvals[i];
1907:350:            len = dot (l.u, minus (vi, v0));
1908:351:            if (len > vmax)
1909:352:                vmax = len;
1910:353:            if (len < vmin)
1911:354:                vmin = len;
1912:355:        }
1913:356:        l.p1 = plus (times (l.u, vmin), v0);
1914:357:        l.p2 = plus (times (l.u, vmax), v0);
1915:358:    } else {
1916:359:        l.e = fitline (xvals, yvals, numpoints, &l.a, &l.b, &l.c);
1917:360:        /* returns normalised values */
1918:361:        l.u.x = -l.b;
1919:362:        l.u.y = l.a;
1920:363:        v0.x = l.a * l.c;
1921:364:        v0.y = l.b * l.c;
1922:365:        v0.z = 0;
1923:366:/* v0 is now a point on the line */
1924:367:        vi.z = 0;
1925:368:        for (i = 0; i < l.n; i++) {
1926:369:            vi.x = xvals[i];
1927:370:            vi.y = yvals[i];
1928:371:            len = dot (l.u, minus (vi, v0));
1929:372:            if (len > vmax)
1930:373:                vmax = len;
1931:374:            if (len < vmin)
1932:375:                vmin = len;
1933:376:        }
1934:377:        l.p1 = plus (times (l.u, vmin), v0);
1935:378:        l.p2 = plus (times (l.u, vmax), v0);
1936:379:    }
1937:380:    l.psi = atan2 (l.u.y, l.u.x);
1938:381:    l.beta = atan2 (l.u.z, sqrt (sqr (l.u.x) + sqr (l.u.y)));
1939:382:    l.l = norm (minus (l.p2, l.p1));
1940:383:    l.m = times (plus (l.p1, l.p2), 0.5);
1941:384:    l.type = LINE_SEGMENT;
1942:385:    return l;
```

```
1943:386:}
1944:387:
1945:388:
1946:389:/* [requires l->p1, l->u] */
1947:390:double distancetoline (Vec x, LineSegment * l)
1948:391:{
1949:392:    double r;
1950:393:    x = minus (x, l->p1);
1951:394:    r = fsqr (norm (x)) - fsqr (dot (x, l->u));
1952:395:    if (r <= 0)
1953:396:        return 0;
1954:397:    return sqrt (r);
1955:398:}
1956:399:
1957:400:#define Distancetoplane(x,u,d) fabs(dot(minus(x, times(u,d)),u))
1958:401:
1959:402:/*  [requires p->u and p->d] */
1960:403:double distancetoplane (Vec x, PlaneSegment * p)
1961:404:{
1962:405:    return Distancetoplane (x, p->u, p->d);
1963:406:}
1964:407:
1965:408:
1966:409:/* [requires p1, u] */
1967:410:Vec pointonline (Vec x, LineSegment * l)
1968:411:{
1969:412:    return plus (l->p1, times (l->u, dot (minus (x, l->p1), l->u)));
1970:413:}
1971:414:
1972:415:/* returns the point on the plane closest to x */
1973:416:/*  [requires p->n and p->d] */
1974:417:Vec pointonplane (Vec x, PlaneSegment * p)
1975:418:{
1976:419:    Vec c = times (p->u, p->d);
1977:420:    x = minus (x, c);
1978:421:    return plus (minus (x, times (p->u, dot (x, p->u))), c);
1979:422:}
1980:423:
1981:424:
1982:425:/* returns half way between the two lines where they are closest, \
1983:426:   [requires p1, p2 and u for both lines] */
1984:427:Vec interceptionbetweentwolines (LineSegment * l1, LineSegment * l2)
1985:428:{
1986:429:    Matrix *A = Madoublestomatrix (2, 3, dot (l1->u, l1->u), \
1987:430:                -dot (l1->u, l2->u), dot (minus (l2->p1, l1->p1), l1->u),
1988:431:                                dot (l1->u, l2->u), -dot (l2->u, l2->u), \
1989:432:                                    dot (minus (l2->p1, l1->p1), l2->u));
1990:433:    Vec v;
1991:434:    if (Madiag (A)) {
1992:435:        v = times (plus (plus (l1->p1, times (l1->u, Mard (*A, 0, 2))),
1993:436:                    plus (l2->p1, times (l2->u, Mard (*A, 1, 2)))), 0.5);
1994:437:    } else {
1995:438:        v = times (plus (plus (l1->p1, l1->p2), plus (l2->p1, l2->p2)), 0.25);
1996:439:    }
1997:440:    Mafreematrix (A);
1998:441:    return v;
1999:442:}
2000:443:
2001:444:
2002:445:double shortestdistancebetweentwolines (LineSegment * l1, LineSegment * l2)
2003:446:{
2004:447:    Vec c = cross (l1->u, l2->u);
2005:448:    double n = norm (c);
2006:449:    if (n) {
2007:450:        return fabs (dot (minus (l1->p1, l2->p1), times (c, 1 / n)));
2008:451:    } else {                    /* the lines are parallel */
2009:452:        Vec m = minus (l1->p1, l2->p1);
2010:453:        return sqrt (fsqr (norm (m)) - fsqr (dot (m, l1->u)));
2011:454:    }
2012:455:}
2013:456:
2014:457:
2015:458:
2016:459:/* returns 1 if on the line, zero otherwise, [requires p1, u, and l] */
2017:460:int ispointonline (Vec x, LineSegment * l)
2018:461:{
2019:462:    if (l->l) {
2020:463:        double d = dot (minus (x, l->p1), l->u);
2021:464:        if (d > l->l || d < 0)
2022:465:            return 0;
```

```
2023:466:        return 1;
2024:467:    }
2025:468:    return 0;
2026:469:}
2027:470:
2028:471:
2029:472:
2030:473:
2031:474:/* returns only .p1 and .u of the LineSegement */
2032:475:LineSegment lineinterceptionoftwoplanes (PlaneSegment * p1, PlaneSegment * p2)
2033:476:{
2034:477:    LineSegment l;
2035:478:    Vec r = cross (p1->u, p2->u);
2036:479:    Matrix *A;
2037:480:
2038:481:    memset (&l, 0, sizeof (LineSegment));
2039:482:    l.u = times (r, 1 / norm (r));
2040:483:    A = Madoublestomatrix (3, 4, p1->u.x, p1->u.y, p1->u.z, p1->d,
2041:484:                           p2->u.x, p2->u.y, p2->u.z, p2->d,
2042:485:                           l.u.x, l.u.y, l.u.z, NULL);
2043:486:    Madiag (A);
2044:487:    l.p1.x = Mard (*A, 0, 3);
2045:488:    l.p1.y = Mard (*A, 1, 3);
2046:489:    l.p1.z = Mard (*A, 2, 3);
2047:490:    Mafreematrix (A);
2048:491:    return l;
2049:492:}
2050:493:
2051:494:
2052:495:
2053:496:/* returns only ->c and ->u of the LineSegement */
2054:497:LineSegment segmentinterceptionoftwoplanes (PlaneSegment * p1, PlaneSegment * p2)
2055:498:{
2056:499:    LineSegment l = lineinterceptionoftwoplanes (p1, p2);
2057:500:    Vec limits[16];
2058:501:    int nlimits = 0;
2059:502:    int i, nmax = 0, nmin = 0;
2060:503:    double vmax = -10e90, vmin = 10e90, len;
2061:504:    double d;
2062:505:
2063:506:    for (i = 0; i < p1->num; i++) {
2064:507:        limits[nlimits] = interceptionbetweentwolines (&l, &p1->l[i]);
2065:508:        d = shortestdistancebetweentwolines (&l, &p1->l[i]);
2066:509:        if (ispointonline (limits[nlimits], &p1->l[i]))
2067:510:            nlimits++;
2068:511:    }
2069:512:
2070:513:    for (i = 0; i < p2->num; i++) {
2071:514:        limits[nlimits] = interceptionbetweentwolines (&l, &p2->l[i]);
2072:515:        d = shortestdistancebetweentwolines (&l, &p2->l[i]);
2073:516:        if (ispointonline (limits[nlimits], &p2->l[i]))
2074:517:            nlimits++;
2075:518:    }
2076:519:
2077:520:    if (nlimits) {
2078:521:        for (i = 0; i < nlimits; i++) {
2079:522:            len = dot (l.u, minus (l.p1, limits[i]));
2080:523:            if (len > vmax) {
2081:524:                nmax = i;
2082:525:                vmax = len;
2083:526:            }
2084:527:            if (len < vmin) {
2085:528:                nmin = i;
2086:529:                vmin = len;
2087:530:            }
2088:531:        }
2089:532:        l.p1 = limits[nmin];
2090:533:        l.p2 = limits[nmax];
2091:534:    }
2092:535:    return linesegment (l.p1, l.p2);
2093:536:}
2094:537:
2095:538:
2096:539:/* returns the vector pointing from the camera out to a point */
2097:540:LineSegment linethroughpointandcamcentre (double x, double y, Camera * cam)
2098:541:{
2099:542:    PlaneSegment p1, p2;
2100:543:    LineSegment l;
2101:544:    p1.u = plus (times (cam->m_s, x), times (cam->m_x, cam->f));
2102:545:    p1.d = dot (p1.u, cam->x);
```

```
2103:546:    p2.u = plus (times (cam->m_s, y), times (cam->m_y, cam->f));
2104:547:    p2.d = dot (p2.u, cam->x);
2105:548:    l = lineinterceptionoftwoplanes (&p1, &p2);
2106:549:    l.u = times (l.u, fsgn (dot (cam->m_s, l.u)));
2107:550:    /* direction is arbitrary, so make it point outward from the camera */
2108:551:    l.p1 = cam->x;
2109:552:    l.p2 = plus (l.p1, times (l.u, 10e50));
2110:553:    return linesegment (l.p1, l.p2);
2111:554:}
2112:555:
2113:556:
2114:557:
2115:558:/*
2116:559:   Calculates:
2117:560:   [ a*f*m_x + b*f*m_y + c*m_s : (a*f*m_x + b*f*m_y + c*m_s).X_0 ] = [ abc : d ]
2118:561:   whence
2119:562:   ax + by + cz = d
2120:563:   (a,b,c) is a unit vector normal to the plane.
2121:564: */
2122:565:PlaneSegment planethroughlineandcamcentre (LineSegment * l, Camera * cam)
2123:566:{
2124:567:    PlaneSegment plane;
2125:568:    Vec abc;
2126:569:    double r;
2127:570:    memset (&plane, 0, sizeof (PlaneSegment));
2128:571:
2129:572:    abc = plus (plus (times (cam->m_x, cam->f * l->a), \
2130:573:                times (cam->m_y, cam->f * l->b)), times (cam->m_s, l->c));
2131:574:    plane.d = dot (abc, cam->x);
2132:575:    r = norm (abc);
2133:576:    abc = times (abc, 1 / r);
2134:577:    plane.u = abc;
2135:578:    plane.d /= r;
2136:579:
2137:580:/* now find limits of the plane */
2138:581:    plane.l[0] = linethroughpointandcamcentre (l->p1.x, l->p1.y, cam);
2139:582:    plane.l[1] = linethroughpointandcamcentre (l->p2.x, l->p2.y, cam);
2140:583:    plane.num = 2;
2141:584:
2142:585:    return plane;
2143:586:}
2144:587:
2145:588:
2146:589:
2147:590:
2148:591:LineSegment fit3dlinetoprojections (LineSegment * line1, LineSegment * line2,
2149:592:                                    Camera * cam1, Camera * cam2)
2150:593:{
2151:594:    PlaneSegment plane1 = planethroughlineandcamcentre (line1, cam1);
2152:595:    PlaneSegment plane2 = planethroughlineandcamcentre (line2, cam2);
2153:596:    return segmentinterceptionoftwoplanes (&plane1, &plane2);
2154:597:}
2155:598:
2156:599:
2157:600:/* this undestorts x and y values */
2158:601:LineSegment linefromstereoedge (double *x1, double *y1, int n1, double *x2, \
2159:602:                          double *y2, int n2, Camera * cam1, Camera * cam2)
2160:603:{
2161:604:    LineSegment l1, l2;
2162:605:
2163:606:    l1 = linesegmentfrompoints (x1, y1, 0, n1);
2164:607:    l2 = linesegmentfrompoints (x2, y2, 0, n2);
2165:608:
2166:609:    return fit3dlinetoprojections (&l1, &l2, cam1, cam2);
2167:610:}
2168:611:
2169:612:
2170:613:
2171:614:
2172:615:
2173:616:
2174:617:/* how much do the line segments exactly match ? */
2175:618:double comparelinesegments (LineSegment * l1, LineSegment * l2)
2176:619:{
2177:620:    return fmin (
2178:621:            norm (minus (l1->p1, l2->p1)) + norm (minus (l1->p2, l2->p2))
2179:622:                ,
2180:623:            norm (minus (l1->p2, l2->p1)) + norm (minus (l1->p1, l2->p2))
2181:624:        );
2182:625:}
```

```
2183:626:
2184:627:/* how much do the line equations exactly match in the region of the lines ? */
2185:628:double comparelines (LineSegment * l1, LineSegment * l2)
2186:629:{
2187:630:    return
2188:631:        fmax (distancetoline (l1->p1, l2) +
2189:632:              distancetoline (l1->p2, l2),
2190:633:              distancetoline (l2->p1, l1) +
2191:634:              distancetoline (l2->p2, l1));
2192:635:/* fmax is so that if one of the lines is short, it does not appear
2193:636:   to have a small error */
2194:637:}
2195:638:
2196:639:
2197:640:
2198:641:
2199:642:void findcylinderedges (Camera * cam, LineSegment * axis, double r, \
2200:643:                        LineSegment * one_edge, LineSegment * other_edge)
2201:644:{
2202:645:    double a, b, c, d, e, f;
2203:646:    double A, B, C;
2204:647:    Vec r1, r2;
2205:648:    double alpha1, alpha2;
2206:649:    Vec p = minus (screen (cam, axis->p2), screen (cam, axis->p1));
2207:650:
2208:651:    fswap (p.x, p.y);
2209:652:    p.x = -p.x;                 /* p now normal to line of projected axis */
2210:653:
2211:654:    orth_vectors (axis->u, &r1, &r2, r);
2212:655:
2213:656:    a = (dot (axis->p1, cam->m_x) - dot (cam->x, cam->m_x)) * p.x + \
2214:657:        (dot (axis->p1, cam->m_y) - dot (cam->x, cam->m_y)) * p.y;
2215:658:    b = (dot (r1, cam->m_x)) * p.x + (dot (r1, cam->m_y)) * p.y;
2216:659:    c = (dot (r2, cam->m_x)) * p.x + (dot (r2, cam->m_y)) * p.y;
2217:660:
2218:661:    d = dot (axis->p1, cam->m_s) - dot (cam->x, cam->m_s);
2219:662:    e = dot (r1, cam->m_s);
2220:663:    f = dot (r2, cam->m_s);
2221:664:
2222:665:/* (b d - a e) COS (t) + (a f - c d) SIN (t) + b f - c e  =  0 */
2223:666:
2224:667:    A = a * f - c * d;
2225:668:    B = b * d - a * e;
2226:669:    C = b * f - c * e;
2227:670:
2228:671:    a = sqr (A) + sqr (B);
2229:672:    b = 2 * B * C;
2230:673:    c = sqr (C) - sqr (A);
2231:674:    d = sqrt (sqr (b) - 4 * a * c);
2232:675:    alpha1 = acos ((-b + d) / (2 * a));
2233:676:    alpha2 = acos ((-b - d) / (2 * a));
2234:677:
2235:678:    *one_edge = linesegment (
2236:679:                    screen (cam, plus (plus (times (r1, sin (alpha1)), \
2237:680:                                        times (r2, cos (alpha1))), axis->p1)), \
2238:681:                    screen (cam, plus (plus (times (r1, sin (alpha1)), \
2239:682:                                        times (r2, cos (alpha1))), axis->p2)));
2240:683:    \
2241:684:        *other_edge = linesegment (
2242:685:                    screen (cam, plus (plus (times (r1, sin (alpha2)), \
2243:686:                                        times (r2, cos (alpha2))), axis->p1)), \
2244:687:                    screen (cam, plus (plus (times (r1, sin (alpha2)), \
2245:688:                                        times (r2, cos (alpha2))), axis->p2)));
2246:689:}
2247:690:
2248:691:
2249:692:
2250:693:
2251:694:
2252:695:
2253:696:Cylinder cylinderfromstereoedge (double *x1l, double *y1l, int n1l, \
2254:697:           double *x2l, double *y2l, int n2l, double *x1r, double *y1r, \
2255:698:int n1r, double *x2r, double *y2r, int n2r, Camera * caml, Camera * camr)
2256:699:{
2257:700:    double t, d, dp;
2258:701:    LineSegment l1r, l2r, l1l, l2l;
2259:702:    LineSegment ll, lr, edge1l, edge2l, edge1r, edge2r;
2260:703:    Cylinder cyl;
2261:704:    double p[4];
2262:705:    Vec v, u;
```

```
2263:706:    int i;
2264:707:    double mx, mn;
2265:708:
2266:709:    l1l = linesegmentfrompoints (x1l, y1l, 0, n1l);
2267:710:    l2l = linesegmentfrompoints (x2l, y2l, 0, n2l);
2268:711:    u = times (plus (l1l.u, l2l.u), 0.5);
2269:712:    u = times (u, 1 / norm (u));
2270:713:
2271:714:/* is z value zero here ? */
2272:715:    v = interceptionbetweentwolines (&l1l, &l2l);
2273:716:    p[0] = dot (u, minus (l1l.p1, v));
2274:717:    p[1] = dot (u, minus (l1l.p2, v));
2275:718:    p[2] = dot (u, minus (l2l.p1, v));
2276:719:    p[3] = dot (u, minus (l2l.p2, v));
2277:720:
2278:721:    mx = -9e90;
2279:722:    mn = 9e90;
2280:723:    for (i = 0; i < 4; i++) {
2281:724:        if (p[i] > mx)
2282:725:            mx = p[i];
2283:726:        if (p[i] < mn)
2284:727:            mn = p[i];
2285:728:    }
2286:729:    ll.p1 = plus (v, times (u, mn));
2287:730:    ll.p2 = plus (v, times (u, mx));
2288:731:
2289:732:    ll = linesegment (ll.p1, ll.p2);
2290:733:
2291:734:    l1r = linesegmentfrompoints (x1r, y1r, 0, n1r);
2292:735:    l2r = linesegmentfrompoints (x2r, y2r, 0, n2r);
2293:736:    u = times (plus (l1r.u, l2r.u), 0.5);
2294:737:    u = times (u, 1 / norm (u));
2295:738:
2296:739:/* is z value zero here ? */
2297:740:    v = interceptionbetweentwolines (&l1r, &l2r);
2298:741:    p[0] = dot (u, minus (l1r.p1, v));
2299:742:    p[1] = dot (u, minus (l1r.p2, v));
2300:743:    p[2] = dot (u, minus (l2r.p1, v));
2301:744:    p[3] = dot (u, minus (l2r.p2, v));
2302:745:
2303:746:    mx = -9e90;
2304:747:    mn = 9e90;
2305:748:    for (i = 0; i < 4; i++) {
2306:749:        if (p[i] > mx)
2307:750:            mx = p[i];
2308:751:        if (p[i] < mn)
2309:752:            mn = p[i];
2310:753:    }
2311:754:    lr.p1 = plus (v, times (u, mn));
2312:755:    lr.p2 = plus (v, times (u, mx));
2313:756:
2314:757:    lr = linesegment (lr.p1, lr.p2);
2315:758:
2316:759:
2317:760:    cyl.l = fit3dlinetoprojections (&ll, &lr, caml, camr);
2318:761:    edgel1 = fit3dlinetoprojections (&l1r, &l1l, camr, caml);
2319:762:    cyl.r = distancetoline (edgel1.p1, &cyl.l);  /* estimate */
2320:763:
2321:764:    t = cyl.r / 10;
2322:765:    d = 1e90;
2323:766:    do {
2324:767:        do {
2325:768:            cyl.r += t;
2326:769:            dp = d;
2327:770:            findcylinderedges (caml, &cyl.l, cyl.r, &edgel1, &edge2l);
2328:771:            findcylinderedges (camr, &cyl.l, cyl.r, &edgelr, &edge2r);
2329:772:            d = fmin (comparelines (&edgel1, &l1l), comparelines (&edge2l, &l1l));
2330:773:            /* we don't know which line is which */
2331:774:            d += fmin (comparelines (&edgelr, &l1r), comparelines (&edge2r, &l1r));
2332:775:        } while (d < dp);
2333:776:        t *= -0.5;
2334:777:    } while (fabs (t) > 0.00000001);
2335:778:
2336:779:    cyl.e = l1l.e + l1r.e + l2l.e + l2r.e + d;
2337:780:
2338:781:    cyl.type = CYLINDER;
2339:782:    return cyl;
2340:783:}
2341:784:
2342:785:double fitcirc (double *xvals, double *yvals, int numpoints, \
```

```
2343:786:                double *ax, double *ay, double *ar);
2344:787:
2345:788:
2346:789:double circ_x, circ_y, circ_r;
2347:790:Vec cyl_r1, cyl_r2, cyl_n;
2348:791:
2349:792:static double tominimise3dcylinder (double *u)
2350:793:{
2351:794:    double e, *x, *y;
2352:795:    Vec v;
2353:796:    int i;
2354:797:
2355:798:    cyl_n.x = u[0];
2356:799:    cyl_n.y = u[1];
2357:800:    cyl_n.z = u[2];
2358:801:    if (norm (cyl_n) == 0) {
2359:802:        cyl_n.z = 1;
2360:803:    }
2361:804:    orth_vectors (cyl_n, &cyl_r1, &cyl_r2, 1);
2362:805:
2363:806:    x = malloc (num_3d_points * sizeof (double));
2364:807:    y = malloc (num_3d_points * sizeof (double));
2365:808:
2366:809:    for (i = 0; i < num_3d_points; i++) {
2367:810:        v.x = x_points[i];
2368:811:        v.y = y_points[i];
2369:812:        v.z = z_points[i];
2370:813:        x[i] = dot (cyl_r1, v);
2371:814:        y[i] = dot (cyl_r2, v);
2372:815:    }
2373:816:
2374:817:    e = fitcirc (x, y, num_3d_points, &circ_x, &circ_y, &circ_r);
2375:818:
2376:819:    free (y);
2377:820:    free (x);
2378:821:
2379:822:    return e;
2380:823:}
2381:824:
2382:825:
2383:826:
2384:827:
2385:828:
2386:829:Cylinder cylinderfrompoints (double *xvals, double *yvals, \
2387:830:                              double *zvals, int numpoints)
2388:831:{
2389:832:    double result[3];
2390:833:    Cylinder c;
2391:834:    Vec v0, vi;
2392:835:    double len, e, emin = 9e90;
2393:836:    int i, imin = 0;
2394:837:    double vmax = -9e90;
2395:838:    double vmin = 9e90;
2396:839:
2397:840:    num_3d_points = numpoints;
2398:841:    x_points = xvals;
2399:842:    y_points = yvals;
2400:843:    z_points = zvals;
2401:844:
2402:845:/* try with various initial guesses */
2403:846:    for (i = 0; i < 3; i++) {
2404:847:        result[0] = 0;
2405:848:        result[1] = 0;
2406:849:        result[2] = 0;
2407:850:        result[i] = 1;
2408:851:        simplex_optimise (result, 3, 10e-3, 1, tominimise3dcylinder, NULL);
2409:852:        e = tominimise3dcylinder (result);
2410:853:        if (e < emin) {
2411:854:            emin = e;
2412:855:            imin = i;
2413:856:        }
2414:857:    }
2415:858:
2416:859:    result[0] = 0;
2417:860:    result[1] = 0;
2418:861:    result[2] = 0;
2419:862:    result[imin] = 1;
2420:863:    simplex_optimise (result, 3, 10e-7, 1, tominimise3dcylinder, NULL);
2421:864:
2422:865:    c.e = emin;
```

```
2423:866:    c.r = circ_r;
2424:867:    cyl_n = times (cyl_n, 1 / norm (cyl_n));
2425:868:    c.l.u = cyl_n;
2426:869:    v0 = plus (times (cyl_r1, circ_x), times (cyl_r2, circ_y));
2427:870:
2428:871:    for (i = 0; i < numpoints; i++) {
2429:872:        vi.x = xvals[i];
2430:873:        vi.y = yvals[i];
2431:874:        vi.z = zvals[i];
2432:875:        len = dot (c.l.u, minus (vi, v0));
2433:876:        if (len > vmax)
2434:877:            vmax = len;
2435:878:        if (len < vmin)
2436:879:            vmin = len;
2437:880:    }
2438:881:    c.l = linesegment (plus (times (c.l.u, vmin), v0), \
2439:882:                       plus (times (c.l.u, vmax), v0));
2440:883:    c.type = CYLINDER;
2441:884:    c.l.n = numpoints;
2442:885:    return c;
2443:886:}
2444:887:
2445:888:
2446:889:
2447:890:
2448:891:
2449:892:/* fits a circle numpoints points (xvals, yvals). Places result in (ax,bx) radius r */
2450:893:double fitcirc (double *xvals, double *yvals, int numpoints, double *ax, \
2451:894:                double *ay, double *ar)
2452:895:{
2453:896:    double s = 0, sx = 0, sy = 0, sxxy = 0, sxyy = 0, sxy = 0, sxx = 0, sxxx = 0,
2454:897:     syy = 0, syyy = 0, e = 0;
2455:898:    Matrix *A;
2456:899:    int i;
2457:900:    double xs, ys;
2458:901:
2459:902:    for (i = 0; i < numpoints; i++) {
2460:903:        s++;
2461:904:        xs = xvals[i];
2462:905:        ys = yvals[i];
2463:906:
2464:907:        sx += xs;
2465:908:        sy += ys;
2466:909:        sxy += xs * ys;
2467:910:        sxxy += ys * xs * xs;
2468:911:        sxyy += ys * ys * xs;
2469:912:        sxx += xs * xs;
2470:913:        sxxx += xs * xs * xs;
2471:914:        syy += ys * ys;
2472:915:        syyy += ys * ys * ys;
2473:916:    }
2474:917:
2475:918:    A = Madoublestomatrix (3, 4, sxx, sxy, sx, sxxx + sxyy,
2476:919:                           sxy, syy, sy, syyy + sxxy,
2477:920:                           sx, sy, s, sxx + syy);
2478:921:
2479:922:    if (Madiag (A)) {
2480:923:        *ax = Mard (*A, 0, 3) / 2;
2481:924:        *ay = Mard (*A, 1, 3) / 2;
2482:925:        *ar = sqrt (Mard (*A, 2, 3) + *ax * *ax + *ay * *ay);
2483:926:        for (i = 0; i < numpoints; i++)  /* calculate error */
2484:927:            e += fsqr (*ar - sqrt (fsqr (*ax - xvals[i]) + fsqr (*ay - yvals[i])));
2485:928:    } else
2486:929:        e = 1e90;                /* singular matrix */
2487:930:
2488:931:    Mafreematrix (A);
2489:932:    return e;
2490:933:}
2491:934:
2492:935:
2493:936:
2494:937:
2495:938:struct sums {
2496:939:    double c;
2497:940:    double d;
2498:941:    double e;
2499:942:    double f;
2500:943:    double g;
2501:944:    double h;
2502:945:    double i;
```

```
2503:946:    double j;
2504:947:    double k;
2505:948:    double l;
2506:949:    double m;
2507:950:    double n;
2508:951:    double o;
2509:952:    double p;
2510:953:};
2511:954:
2512:955:Vec solve3x3 (double a[4][3]);
2513:956:
2514:957:double ellipfunc (double a, double b, struct sums *s, \
2515:958:                  double *xu, double *yu, double *zu)
2516:959:{
2517:960:    double matrix[4][3];
2518:961:    double x, y, z;
2519:962:    double funcf, funcg;
2520:963:
2521:964:/* Reduce five non-linear equations to two non-linear equations */
2522:965:/* by solving for three of them: */
2523:966:    matrix[0][0] = (8 * pow (a, 4) * s->f + 16 * pow (a, 3) * b * s->e + \
2524:967:                    sqr (a) * (8 * sqr (b) * s->g + 16 * s->f) + 16 * a * b * s->e + 8 * s->f);
2525:968:    matrix[1][0] = (8 * pow (a, 3) * b * s->f + sqr (a) * (16 * sqr (b) * \
2526:969:        s->e + 8 * s->e) + a * (8 * pow (b, 3) * s->g + b * (8 * s->f + \
2527:970:                                8 * s->g)) + 8 * sqr (b) * s->e + 8 * s->e);
2528:971:    matrix[2][0] = (-4 * sqr (a) * s->c - 4 * a * b * s->d - 4 * s->c);
2529:972:    matrix[3][0] = -4 * pow (a, 4) * s->j - 12 * pow (a, 3) * b * s->h \
2530:973:        +sqr (a) * (-12 * sqr (b) * s->i - 4 * s->i - 8 * s->j) + \
2531:974:        a * (b * (-12 * s->h - 4 * s->k) - 4 * pow (b, 3) * s->k) - \
2532:975:        4 * sqr (b) * s->i - 4 * s->i - 4 * s->j;
2533:976:
2534:977:    matrix[0][1] = (8 * pow (a, 3) * b * s->f + sqr (a) * (16 * sqr (b) * \
2535:978:        s->e + 8 * s->e) + a * (8 * pow (b, 3) * s->g + b * (8 * s->f + \
2536:979:                                8 * s->g)) + 8 * sqr (b) * s->e + 8 * s->e);
2537:980:    matrix[1][1] = (8 * sqr (a) * sqr (b) * s->f + a * (16 * pow (b, 3) * \
2538:981:         s->e + 16 * b * s->e) + 8 * pow (b, 4) * s->g + 16 * sqr (b) * \
2539:982:                   s->g + 8 * s->g);
2540:983:    matrix[2][1] = (-4 * a * b * s->c - 4 * sqr (b) * s->d - 4 * s->d);
2541:984:    matrix[3][1] = -4 * pow (a, 3) * b * s->j + sqr (a) * (-12 * sqr (b) \
2542:985:             *s->h - 4 * s->h) + a * (b * (-12 * s->i - 4 * s->j) - 12 * \
2543:986:           pow (b, 3) * s->i) - 4 * pow (b, 4) * s->k + sqr (b) * (-4 * \
2544:987:                                  s->h - 8 * s->k) - 4 * s->h - 4 * s->k;
2545:988:
2546:989:    matrix[0][2] = (-4 * sqr (a) * s->c - 4 * a * b * s->d - 4 * s->c);
2547:990:    matrix[1][2] = (-4 * a * b * s->c - 4 * sqr (b) * s->d - 4 * s->d);
2548:991:    matrix[2][2] = 2;
2549:992:    matrix[3][2] = 2 * sqr (a) * s->f + 4 * a * b * s->e + 2 * sqr (b) \
2550:993:        *s->g + 2 * s->f + 2 * s->g;
2551:994:
2552:995:    solve3x3 (matrix);
2553:996:    x = -matrix[3][0];
2554:997:    y = -matrix[3][1];
2555:998:    z = -matrix[3][2];
2556:999:
2557:1000:#define SQR(x) ((x)*(x))
2558:1001:
2559:1002:    /* Return value of two non-linears: */
2560:1003:    funcf = 16 * SQR (x) * pow (a, 3) * s->f + 24 * SQR (x) * SQR (a) * b * \
2561:1004:        s->e + 8 * SQR (x) * a * SQR (b) * s->g + 16 * SQR (x) * a * s->f + 8 * \
2562:1005:        SQR (x) * b * s->e + 24 * x * y * SQR (a) * b * s->f + 32 * x * y * a * SQR (b) * \
2563:1006:        s->e + 16 * x * y * a * s->e + 8 * x * y * pow (b, 3) * s->g + 8 * x * y * b * s->f \
2564:1007:        +8 * x * y * b * s->g - 8 * x * z * a * s->c - 4 * x * z * b * s->d - 16 * x * pow \
2565:1008:        (a, 3) * s->j - 36 * x * SQR (a) * b * s->h - 24 * x * a * SQR (b) * s->i - 16 * x * \
2566:1009:        a * s->j - 8 * x * a * s->i - 4 * x * pow (b, 3) * s->k - 12 * x * b * s->h - 4 * x.* \
2567:1010:        b * s->k + 8 * SQR (y) * a * SQR (b) * s->f + 8 * SQR (y) * pow (b, 3) * s->e + 8 * \
2568:1011:        SQR (y) * b * s->e - 4 * y * z * b * s->c - 12 * y * SQR (a) * b * s->j - 24 * y * a \
2569:1012:        *SQR (b) * s->h - 8 * y * a * s->h - 12 * y * pow (b, 3) * s->i - 4 * y * b * s->j \
2570:1013:        -12 * y * b * s->i + 4 * z * a * s->f + 4 * z * b * s->e + 4 * pow (a, 3) * s->o + \
2571:1014:        12 * SQR (a) * b * s->n + 12 * a * SQR (b) * s->m + 4 * a * s->o + 4 * a * s->m + 4 \
2572:1015:        *pow (b, 3) * s->l + 4 * b * s->n + 4 * b * s->l;
2573:1016:    funcg = 8 * SQR (x) * pow (a, 3) * s->e + 8 * SQR (x) * SQR (a) * b * s->g \
2574:1017:        +8 * SQR (x) * a * s->e + 8 * x * y * pow (a, 3) * s->f + 32 * x * y * SQR (a) * b \
2575:1018:        *s->e + 24 * x * y * a * SQR (b) * s->g + 8 * x * y * a * s->f + 8 * x * y * a * \
2576:1019:        s->g + 16 * x * y * b * s->e - 4 * x * z * a * s->d - 12 * x * pow (a, 3) * s->h - \
2577:1020:        24 * x * SQR (a) * b * s->i - 12 * x * a * SQR (b) * s->k - 12 * x * a * s->h - 4 * \
2578:1021:        x * a * s->k - 8 * x * b * s->i + 8 * SQR (y) * SQR (a) * b * s->f + 24 * SQR (y) \
2579:1022:        *a * SQR (b) * s->e + 8 * SQR (y) * a * s->e + 16 * SQR (y) * pow (b, 3) * s->g \
2580:1023:        +16 * SQR (y) * b * s->g - 4 * y * z * a * s->c - 8 * y * z * b * s->d - 4 * y * \
2581:1024:        pow (a, 3) * s->j - 24 * y * SQR (a) * b * s->h - 36 * y * a * SQR (b) * s->i - \
2582:1025:        4 * y * a * s->j - 12 * y * a * s->i - 16 * y * pow (b, 3) * s->k - 8 * y * b * \
```

```
2583:1026:        s->h - 16 * y * b * s->k + 4 * z * a * s->e + 4 * z * b * s->g + 4 * pow (a, 3) \
2584:1027:        *s->n + 12 * SQR (a) * b * s->m + 12 * a * SQR (b) * s->l + 4 * a * s->n + 4 * \
2585:1028:        a * s->l + 4 * pow (b, 3) * s->p + 4 * b * s->m + 4 * b * s->p;
2586:1029:
2587:1030:    /* Return other variables: */
2588:1031:    *xu = x;
2589:1032:    *yu = y;
2590:1033:    *zu = z;
2591:1034:    return funcf * funcf + funcg * funcg;
2592:1035:}
2593:1036:
2594:1037:
2595:1038:double ellipse_x;
2596:1039:double ellipse_y;
2597:1040:double ellipse_z;
2598:1041:struct sums ellipse_sums;
2599:1042:
2600:1043:double tominimiseellipse (double *r)
2601:1044:{
2602:1045:    return ellipfunc (r[0], r[1], &ellipse_sums,
2603:1046:                      &ellipse_x, &ellipse_y, &ellipse_z);
2604:1047:}
2605:1048:
2606:1049:
2607:1050:double fitellip (double *xvals, double *yvals, int numpoints, \
2608:1051:                 double *ax, double *ay, double *aa, double *ab, double *ar)
2609:1052:{
2610:1053:    int i;
2611:1054:    double s = 0, sx = 0, sy = 0, sxxy = 0, sxyy = 0;
2612:1055:    double sxy = 0, sxx = 0, sxxx = 0, syy = 0, syyy = 0;
2613:1056:    double syyyy = 0, sxyyy = 0, sxxyy = 0, sxxxy = 0, sxxxx = 0;
2614:1057:    double result[2];
2615:1058:    struct sums sig;
2616:1059:    float xs, ys;
2617:1060:
2618:1061:    for (i = 0; i < numpoints; i++) {
2619:1062:        xs = xvals[i];
2620:1063:        ys = yvals[i];
2621:1064:
2622:1065:        s++;
2623:1066:        sx += (double) xs;
2624:1067:        sy += (double) ys;
2625:1068:        sxy += (double) xs *ys;
2626:1069:        sxxy += (double) ys *xs * xs;
2627:1070:        sxyy += (double) ys *ys * xs;
2628:1071:        sxx += (double) xs *xs;
2629:1072:        sxxx += (double) xs *xs * xs;
2630:1073:        syy += (double) ys *ys;
2631:1074:        syyy += (double) ys *ys * ys;
2632:1075:        sxyyy += (double) ys *ys * ys * xs;
2633:1076:        sxxyy += (double) xs *xs * ys * ys;
2634:1077:        sxxxy += (double) xs *xs * xs * ys;
2635:1078:        sxxxx += (double) xs *xs * xs * xs;
2636:1079:        syyyy += (double) ys *ys * ys * ys;
2637:1080:    }
2638:1081:
2639:1082:/* Normalise all sums: */
2640:1083:    sig.c = sx / s;
2641:1084:    sig.d = sy / s;
2642:1085:    sig.e = sxy / s;
2643:1086:    sig.f = sxx / s;
2644:1087:    sig.g = syy / s;
2645:1088:    sig.h = sxxy / s;
2646:1089:    sig.i = sxyy / s;
2647:1090:    sig.j = sxxx / s;
2648:1091:    sig.k = syyy / s;
2649:1092:    sig.l = sxyyy / s;
2650:1093:    sig.m = sxxyy / s;
2651:1094:    sig.n = sxxxy / s;
2652:1095:    sig.o = sxxxx / s;
2653:1096:    sig.p = syyyy / s;
2654:1097:
2655:1098:    result[0] = 1;
2656:1099:    result[1] = 1;
2657:1100:
2658:1101:    simplex_optimise (result, 2, 10e-7, 1, tominimiseellipse, NULL);
2659:1102:
2660:1103:    *ax = ellipse_x;
2661:1104:    *ay = ellipse_y;
2662:1105:    *aa = result[0];
```

```
2663:1106:    *ab = result[1];
2664:1107:    *ar = sqrt (-ellipse_z + ellipse_x * ellipse_x + ellipse_y * ellipse_y \
2665:1108:        +(result[0] * ellipse_x + result[1] * ellipse_y) * (result[0] * \
2666:1109:                                        ellipse_x + result[1] * ellipse_y));
2667:1110:    return tominimiseellipse (result);
2668:1111:}
2669:1112:
2670:1113:
2671:1114:
2672:1115:
2673:1116:
2674:1117:
2675:1118:
2676:1119:
2677:1120:
2678:1121:
2679:1122:
2680:1123:
2681:1124:
2682:1125:static double tominimise3dplane (double *u)
2683:1126:{
2684:1127:    Vec n, x;
2685:1128:    double d, e = 0;
2686:1129:    int i;
2687:1130:
2688:1131:    n.x = u[0];
2689:1132:    n.y = u[1];
2690:1133:    n.z = u[2];
2691:1134:
2692:1135:    d = norm (n);
2693:1136:    n = times (n, 1 / d);
2694:1137:    d = u[3] / d;
2695:1138:
2696:1139:    for (i = 0; i < num_3d_points; i++) {
2697:1140:        x.x = x_points[i];
2698:1141:        x.y = y_points[i];
2699:1142:        x.z = z_points[i];
2700:1143:        e += fsqr (Distancetoplane (x, n, d));
2701:1144:    }
2702:1145:    return e;
2703:1146:}
2704:1147:
2705:1148:
2706:1149:
2707:1150:
2708:1151:
2709:1152:
2710:1153:PlaneSegment planefrompoints (double *xvals, double *yvals, \
2711:1154:                                double *zvals, int numpoints)
2712:1155:{
2713:1156:    double result[4];
2714:1157:    PlaneSegment p;
2715:1158:    int i;
2716:1159:    double x = 0, y = 0, z = 0;
2717:1160:
2718:1161:    num_3d_points = numpoints;
2719:1162:    x_points = xvals;
2720:1163:    y_points = yvals;
2721:1164:    z_points = zvals;
2722:1165:
2723:1166:    result[0] = 1 / sqrt (3);   /* initial guess */
2724:1167:    result[1] = result[0];
2725:1168:    result[2] = result[0];
2726:1169:    result[3] = 0;
2727:1170:
2728:1171:    simplex_optimise (result, 4, 10e-7, 1, tominimise3dplane, NULL);
2729:1172:
2730:1173:    p.e = tominimise3dplane (result);
2731:1174:    p.u.x = result[0];
2732:1175:    p.u.y = result[1];
2733:1176:    p.u.z = result[2];
2734:1177:    p.u = times (p.u, p.d = 1 / norm (p.u));
2735:1178:    p.d *= result[3];
2736:1179:    p.num = numpoints;
2737:1180:    p.gx = p.u.x / p.u.z;
2738:1181:    p.gy = p.u.y / p.u.z;
2739:1182:    p.elev = sqrt (sqr (p.gx) + sqr (p.gy));
2740:1183:    for (i = 0; i < numpoints; i++) {
2741:1184:        x += xvals[i];
2742:1185:        y += yvals[i];
```

```
2743:1186:        z += zvals[i];
2744:1187:    }
2745:1188:    p.c.x = x / numpoints;
2746:1189:    p.c.y = y / numpoints;
2747:1190:    p.c.z = z / numpoints;
2748:1191:
2749:1192:    p.c = pointonplane (p.c, &p);
2750:1193:    return p;
2751:1194:}
2752:1195:
2753:1196:
2754:1197:
2755:1198:Circle circlefrompoints (double *xvals, double *yvals, double *zvals, int numpoints)
2756:1199:{
2757:1200:    Circle circ;
2758:1201:
2759:1202:    memset (&circ, 0, sizeof (Circle));
2760:1203:
2761:1204:    if (zvals) {                /* 3D fit */
2762:1205:        double rx, ry;
2763:1206:        PlaneSegment p;
2764:1207:        double *x = malloc (numpoints * sizeof (double));
2765:1208:        double *y = malloc (numpoints * sizeof (double));
2766:1209:        Vec r1, r2;
2767:1210:        int i;
2768:1211:        p = planefrompoints (xvals, yvals, zvals, numpoints);
2769:1212:        orth_vectors (p.u, &r1, &r2, 1);
2770:1213:        for (i = 0; i < numpoints; i++) {
2771:1214:/* we want to fit a 2D circle to points on the plane.
2772:1215:   So find positions w.r.t. r1 and r2 */
2773:1216:            x[i] = (xvals[i] - p.c.x) * r1.x + (yvals[i] - p.c.y) * \
2774:1217:                r1.y + (zvals[i] - p.c.z) * r1.z;  /* dot */
2775:1218:            y[i] = (xvals[i] - p.c.x) * r2.x + (yvals[i] - p.c.y) * \
2776:1219:                r2.y + (zvals[i] - p.c.z) * r2.z;
2777:1220:        }
2778:1221:        circ.e = fitcirc (x, y, numpoints, &rx, &ry, &circ.r) + p.e;
2779:1222:        circ.p = plus (plus (times (r1, rx), times (r2, ry)), p.c);
2780:1223:        circ.u = p.u;
2781:1224:        circ.n = numpoints;
2782:1225:    } else {                    /* 2D fit */
2783:1226:        circ.e = fitcirc (xvals, yvals, numpoints, &circ.p.x, &circ.p.y, &circ.r);
2784:1227:        circ.u.z = 1;
2785:1228:        circ.n = numpoints;
2786:1229:    }
2787:1230:    circ.type = CIRCLE;
2788:1231:    return circ;
2789:1232:}
2790:1233:
2791:1234:
2792:1235:Ellipse ellipsefrompoints (double *xvals, double *yvals, double *zvals, int numpoints)
2793:1236:{
2794:1237:    Ellipse ellip;
2795:1238:
2796:1239:    memset (&ellip, 0, sizeof (Ellipse));
2797:1240:
2798:1241:    if (zvals) {                /* 3D fit */
2799:1242:        double rx, ry, ra, rb;
2800:1243:        PlaneSegment p;
2801:1244:        double *x = malloc (numpoints * sizeof (double));
2802:1245:        double *y = malloc (numpoints * sizeof (double));
2803:1246:        Vec r1, r2;
2804:1247:        int i;
2805:1248:        p = planefrompoints (xvals, yvals, zvals, numpoints);
2806:1249:        orth_vectors (p.u, &r1, &r2, 1);
2807:1250:        for (i = 0; i < numpoints; i++) {
2808:1251:/* we want to fit a 2D ellipse to points on the plane.
2809:1252:   So find positions w.r.t. r1 and r2 */
2810:1253:            x[i] = (xvals[i] - p.c.x) * r1.x + (yvals[i] - p.c.y) * \
2811:1254:                r1.y + (zvals[i] - p.c.z) * r1.z;  /* dot */
2812:1255:            y[i] = (xvals[i] - p.c.x) * r2.x + (yvals[i] - p.c.y) * \
2813:1256:                r2.y + (zvals[i] - p.c.z) * r2.z;
2814:1257:        }
2815:1258:        ellip.e = fitellip (x, y, numpoints, &rx, &ry, &ra, &rb, &ellip.r) + p.e;
2816:1259:        ellip.p = plus (plus (times (r1, rx), times (r2, ry)), p.c);
2817:1260:        ellip.q = plus (plus (times (r1, ra), times (r2, rb)), p.c);
2818:1261:        ellip.u = p.u;
2819:1262:        ellip.n = numpoints;
2820:1263:    } else {                    /* 2D fit */
2821:1264:        ellip.e = fitellip (xvals, yvals, numpoints, \
2822:1265:              &ellip.p.x, &ellip.p.y, &ellip.q.x, &ellip.q.y, &ellip.r);
```

```
2823:1266:        ellip.u.z = 1;
2824:1267:        ellip.n = numpoints;
2825:1268:    }
2826:1269:    ellip.type = ELLIPSE;
2827:1270:    return ellip;
2828:1271:}
2829:1272:
2830:1273:
2831:1274:double *xvals1, *yvals1, *xvals2, *yvals2;
2832:1275:Camera *camera1, *camera2;
2833:1276:int num1points, num2points;
2834:1277:
2835:1278:
2836:1279:double to_minimise_stereo_circle (double *r)
2837:1280:{
2838:1281:    double error;
2839:1282:    double a, b, c, d, e, f, g, h, i, j, k, l;
2840:1283:    int ic;
2841:1284:    Vec u, r1, r2, p;
2842:1285:    double x, y;
2843:1286:    double theta, ex, ey, dtheta, etot, etotp;
2844:1287:
2845:1288:    u.x = r[0];
2846:1289:    u.y = r[1];
2847:1290:    u.z = r[2];
2848:1291:    p.x = r[3];
2849:1292:    p.y = r[4];
2850:1293:    p.z = r[5];
2851:1294:
2852:1295:    error = 0;
2853:1296:
2854:1297:    orth_vectors (u, &r1, &r2, norm (u));
2855:1298:
2856:1299:    a = dot (p, camera1->m_x) - dot (camera1->x, camera1->m_x);
2857:1300:    b = dot (r1, camera1->m_x);
2858:1301:    c = dot (r2, camera1->m_x);
2859:1302:
2860:1303:    j = d = dot (p, camera1->m_s) - dot (camera1->x, camera1->m_s);
2861:1304:    k = e = dot (r1, camera1->m_s);
2862:1305:    l = f = dot (r2, camera1->m_s);
2863:1306:
2864:1307:    g = dot (p, camera1->m_y) - dot (camera1->x, camera1->m_y);
2865:1308:    h = dot (r1, camera1->m_y);
2866:1309:    i = dot (r2, camera1->m_y);
2867:1310:
2868:1311:    for (ic = 0; ic < num1points; ic++) {
2869:1312:        x = xvals1[ic];
2870:1313:        y = yvals1[ic];
2871:1314:
2872:1315:        dtheta = 20;
2873:1316:        theta = 0;
2874:1317:        etot = 1e90;
2875:1318:        do {
2876:1319:            do {
2877:1320:                theta += dtheta;
2878:1321:                etotp = etot;
2879:1322:                ex = x + camera1->f * (a + b * sin (theta) + c * cos (theta)) / \
2880:1323:                    (d + e * sin (theta) + f * cos (theta));
2881:1324:                ey = y + camera1->f * (g + h * sin (theta) + i * cos (theta)) / \
2882:1325:                    (j + k * sin (theta) + l * cos (theta));
2883:1326:                etot = ex * ex + ey * ey;
2884:1327:            } while (etot < etotp);
2885:1328:            dtheta *= -0.5;
2886:1329:        } while (fabs (dtheta) > 0.00001);
2887:1330:
2888:1331:        error += etotp;
2889:1332:    }
2890:1333:
2891:1334:    a = dot (p, camera2->m_x) - dot (camera2->x, camera2->m_x);
2892:1335:    b = dot (r1, camera2->m_x);
2893:1336:    c = dot (r2, camera2->m_x);
2894:1337:
2895:1338:    j = d = dot (p, camera2->m_s) - dot (camera2->x, camera2->m_s);
2896:1339:    k = e = dot (r1, camera2->m_s);
2897:1340:    l = f = dot (r2, camera2->m_s);
2898:1341:
2899:1342:    g = dot (p, camera2->m_y) - dot (camera2->x, camera2->m_y);
2900:1343:    h = dot (r1, camera2->m_y);
2901:1344:    i = dot (r2, camera2->m_y);
2902:1345:
```

```
2903:1346:    for (ic = 0; ic < num2points; ic++) {
2904:1347:        x = xvals2[ic];
2905:1348:        y = yvals2[ic];
2906:1349:
2907:1350:        dtheta = 20;
2908:1351:        theta = 0;
2909:1352:        etot = 1e90;
2910:1353:        do {
2911:1354:            do {
2912:1355:                theta += dtheta;
2913:1356:                etotp = etot;
2914:1357:                ex = x + camera2->f * (a + b * sin (theta) + c * cos (theta)) / \
2915:1358:                    (d + e * sin (theta) + f * cos (theta));
2916:1359:                ey = y + camera2->f * (g + h * sin (theta) + i * cos (theta)) / \
2917:1360:                    (j + k * sin (theta) + l * cos (theta));
2918:1361:                etot = ex * ex + ey * ey;
2919:1362:            } while (etot < etotp);
2920:1363:            dtheta *= -0.5;
2921:1364:        } while (fabs (dtheta) > 0.00001);
2922:1365:
2923:1366:        error += etotp;
2924:1367:    }
2925:1368:
2926:1369:    return error;
2927:1370:}
2928:1371:
2929:1372:
2930:1373:
2931:1374:Circle circlefromstereoedge (double *x1, double *y1, int n1, \
2932:1375:            double *x2, double *y2, int n2, Camera * cam1, Camera * cam2)
2933:1376:{
2934:1377:    Circle circ;
2935:1378:    double p[6];
2936:1379:
2937:1380:/* This is a test only */
2938:1381:    p[0] = 0;
2939:1382:    p[1] = 250;
2940:1383:    p[2] = 0;
2941:1384:    p[3] = 375;
2942:1385:    p[4] = 0;
2943:1386:    p[5] = 375;
2944:1387:
2945:1388:    camera1 = cam1;
2946:1389:    camera2 = cam2;
2947:1390:    xvals1 = x1;
2948:1391:    yvals1 = y1;
2949:1392:    xvals2 = x2;
2950:1393:    yvals2 = y2;
2951:1394:    num1points = n1;
2952:1395:    num2points = n2;
2953:1396:
2954:1397:    simplex_optimise (p, 6, 10e-7, 1, to_minimise_stereo_circle, NULL);
2955:1398:
2956:1399:    circ.u.x = p[0];
2957:1400:    circ.u.y = p[1];
2958:1401:    circ.u.z = p[2];
2959:1402:    circ.r = norm (circ.u);
2960:1403:    circ.u = times (circ.u, 1 / circ.r);
2961:1404:    circ.p.x = p[3];
2962:1405:    circ.p.y = p[4];
2963:1406:    circ.p.z = p[5];
2964:1407:    circ.type = CIRCLE;
2965:1408:    circ.e = to_minimise_stereo_circle (p);
2966:1409:    return circ;
2967:1410:}
2968:1:
2969:2:
2970:3:#ifndef FITLINE_H
2971:4:#define FITLINE_H
2972:5:
2973:6:#include "quickmath.h"
2974:7:
2975:8:/* type is one of */
2976:9:enum {
2977:10:    NO_TYPE, VECTOR, POINT, LINE_SEGMENT, CYLINDER, CIRCLE, PLANE_SEGMENT, SURFACE, ELLIPSE
2978:11:};
2979:12:
2980:13:typedef struct vector {
2981:14:    int type;
2982:15:    Vec p;
```

```
2983:16:    Vec u;
2984:17:} Vector;
2985:18:
2986:19:typedef struct point {
2987:20:    int type;
2988:21:    Vec p;
2989:22:} Point;
2990:23:
2991:24:typedef struct lineseg {
2992:25:    int type;
2993:26:    Vec p1;                    /* begin point */
2994:27:    Vec p2;                    /* end point */
2995:28:    double l;                  /* length x1,y1,z1 -> x2,y2,z2 */
2996:29:    double psi;                /* psi = atan2(uy, ux) is the azimuth */
2997:30:    double beta;               /* the elivation = 0 for 2D lines */
2998:31:    double a, b, c;            /* ax + by = c 2d lines only */
2999:32:    Vec u;                     /* unit vector x1,y1 -> x2,y2 */
3000:33:    Vec m;                     /* position of centre */
3001:34:    double e;                  /* error in fit */
3002:35:    int n;                     /* number of points used in fit */
3003:36:} LineSegment;
3004:37:
3005:38:typedef struct cylseg {
3006:39:    int type;
3007:40:    LineSegment l;             /* axis */
3008:41:    double r, e;               /* radius, error */
3009:42:} Cylinder;
3010:43:
3011:44:typedef struct circle {
3012:45:    int type;
3013:46:    Vec p;                     /* centre */
3014:47:    double r;                  /* length x1,y1,z1 -> x2,y2,z2 */
3015:48:    Vec u;                     /* unit vector normal to plane */
3016:49:    double e;                  /* error in fit */
3017:50:    int n;                     /* number of points used in fit */
3018:51:} Circle;
3019:52:
3020:53:typedef struct ellipse {
3021:54:    int type;
3022:55:    Vec p;                     /* centre */
3023:56:    Vec q;                     /* direction of minor axis */
3024:57:    double r;                  /* length x1,y1,z1 -> x2,y2,z2 */
3025:58:    Vec u;                     /* unit vector normal to plane */
3026:59:    double e;                  /* error in fit */
3027:60:    int n;                     /* number of points used in fit */
3028:61:} Ellipse;
3029:62:
3030:63:
3031:64:/* dot(n, (x,y,z)) = d */
3032:65:typedef struct planeseg {
3033:66:    int type;
3034:67:    LineSegment l[8];          /* border */
3035:68:    int num;                   /* number of LineSegments used to border */
3036:69:    Vec u;                     /* normal (unit vector) */
3037:70:    double d;                  /* least distance from origin */
3038:71:    Vec c;                     /* centre point */
3039:72:    double elev;               /* max elivation gradient */
3040:73:    double gx, gy;             /* gradient in the x and y directions */
3041:74:    double e;
3042:75:    int n;
3043:76:} PlaneSegment;
3044:77:
3045:78:typedef struct surface {
3046:79:    int type;
3047:80:    int w, h;
3048:81:    Vec *p;
3049:82:} Surface;
3050:83:
3051:84:typedef union object {
3052:85:    int type;
3053:86:    Point point;
3054:87:    Vector vector;
3055:88:    LineSegment line;
3056:89:    Cylinder cylinder;
3057:90:    Circle circle;
3058:91:    Ellipse ellipse;
3059:92:    PlaneSegment plane;
3060:93:    Surface surface;
3061:94:} Object;
3062:95:
```

```
3063:96:
3064:97:Vec solve3x3 (double a[4][3]);
3065:98:
3066:99:double fitline (double *xvals, double *yvals, int numpoints, double *aa, \
3067:100:                double *ab, double *ac);
3068:101:LineSegment linesegment (Vec p1, Vec p2);
3069:102:LineSegment linesegmentfrompoints (double *xvals, double *yvals, \
3070:103:                                     double *zvals, int numpoints);
3071:104:double distancetoline (Vec x, LineSegment * l);
3072:105:Vec pointonline (Vec x, LineSegment * l);
3073:106:Vec interceptionbetweentwolines (LineSegment * l1, LineSegment * l2);
3074:107:int ispointonline (Vec x, LineSegment * l);
3075:108:LineSegment lineinterceptionoftwoplanes (PlaneSegment * p1, PlaneSegment * p2);
3076:109:LineSegment segmentinterceptionoftwoplanes (PlaneSegment * p1, PlaneSegment * p2);
3077:110:LineSegment linethroughpointandcamcentre (double x, double y, Camera * cam);
3078:111:PlaneSegment planethroughlineandcamcentre (LineSegment * l, Camera * cam);
3079:112:LineSegment fit3dlinetoprojections (LineSegment * line1, LineSegment * line2,
3080:113:                                     Camera * cam1, Camera * cam2);
3081:114:LineSegment linefromstereoedge (double *x1, double *y1, int n1, double *x2, \
3082:115:                          double *y2, int n2, Camera * cam1, Camera * cam2);
3083:116:void linetest3d (void);
3084:117:void get_perp_vec (Vec X, Vec * r1, Vec * r2, double r);
3085:118:double comparelinesegments (LineSegment * l1, LineSegment * l2);
3086:119:double comparelines (LineSegment * l1, LineSegment * l2);
3087:120:void findcylinderedges (Camera * c, LineSegment * axis, double r, \
3088:121:                         LineSegment * one_edge, LineSegment * other_edge);
3089:122:Cylinder cylinderfromstereoedge (double *x1l, double *y1l, int n1l, \
3090:123:                          double *x2l, double *y2l, int n2l, double *x1r, \
3091:124:                double *y1r, int n1r, double *x2r, double *y2r, int n2r, \
3092:125:                                    Camera * camr, Camera * caml);
3093:126:Circle circlefrompoints (double *xvals, double *yvals, double *zvals, int numpoints);
3094:127:Cylinder cylinderfrompoints (double *xvals, double *yvals, double *zvals, \
3095:128:                                int numpoints);
3096:129:Ellipse ellipsefrompoints (double *xvals, double *yvals, double *zvals, int numpoints);
3097:130:Circle circlefromstereoedge (double *x1, double *y1, int n1, \
3098:131:           double *x2, double *y2, int n2, Camera * cam1, Camera * cam2);
3099:132:
3100:133:#endif
3101:1:/*****************************************************************************************/
3102:2:/* hugeimage.c - widget to draw the tiff file, zoom box and do caching of image data     */
3103:3:/*****************************************************************************************/
3104:4:
3105:5:#include <config.h>
3106:6:#include "global.h"
3107:7:#include <stdlib.h>
3108:8:#include <stdio.h>
3109:9:
3110:10:#include <X11/Xlib.h>
3111:11:#include <X11/Xutil.h>
3112:12:
3113:13:#include "stringtools.h"
3114:14:#include "app_glob.c"
3115:15:
3116:16:#include "coolwidget.h"
3117:17:#include "hugeimage.h"
3118:18:#include <sys/types.h>
3119:19:#include <sys/stat.h>
3120:20:#include <fcntl.h>
3121:21:#include <unistd.h>
3122:22:#include <string.h>
3123:23:#include "display.h"
3124:24:#include "loadtiff.h"
3125:25:#include "dialog.h"
3126:26:
3127:27:#include "mad.h"
3128:28:
3129:29:int hugegetpixel (HugeImage * image, unsigned long x, unsigned long y)
3130:30:{
3131:31:    if (y < 0 || y >= image->height || x < 0 || x >= image->width)
3132:32:        return 0;
3133:33:    else {
3134:34:        int i = x >> COLUMNSHIFT;
3135:35:
3136:36:/*check if the point has not already been read: */
3137:37:        if (y < image->linestart[i] || y >= image->lineend[i])
3138:38:            hugeloadnextbuf (image, x, y, i);
3139:39:
3140:40:        return (image->buffers[i])[((y - image->linestart[i]) << COLUMNSHIFT)
3141:41:                                   + (x & COLUMNMASK)];
3142:42:    }
```

```
3143:43:}
3144:44:
3145:45:
3146:46:void hugeloadnextbuf (HugeImage * image, long x, long y, int i)
3147:47:{
3148:48:/*try not to seek if possible, rather just read the next
3149:49:   adjacent LINESCACHED : */
3150:50:    if (y >= image->lineend[i] + LINESCACHED || y < image->lineend[i]) {
3151:51:        lseek (image->temp[i], y * image->columnwidth, SEEK_SET);
3152:52:        image->linestart[i] = y;
3153:53:        image->lineend[i] = y + LINESCACHED;
3154:54:    } else {
3155:55:        image->linestart[i] += LINESCACHED;
3156:56:        image->lineend[i] += LINESCACHED;
3157:57:    }
3158:58:    read (image->temp[i], image->buffers[i],
3159:59:          image->columnwidth * LINESCACHED);
3160:60:}
3161:61:
3162:62:
3163:63:int loadhugeimage (HugeImage * image, const char *fname)
3164:64:{
3165:65:    Window progresswin;
3166:66:    unsigned char *blockofrows; /*holds the block of hieght BLOCKHEIGHT */
3167:67:    long width, height;         /*of the whole image */
3168:68:    long numblocks, numcolumns, i, j, k, bheight;
3169:69:    char tempstr[256];
3170:70:    int fileformat = 0;
3171:71:    unsigned char *cache;
3172:72:    int createtempfiles = 0;
3173:73:    char *imagefile = strdup (fname);
3174:74:
3175:75:    memset (image, 0, sizeof (HugeImage));
3176:76:    image->has_been_allocated = 314159265;
3177:77:    /* this is just so we don't free anything that hasn't been allocated */
3178:78:
3179:79:/*The first thing is to do create a whole lot of temp files,
3180:80:   hence setting up the HugeImage structure.
3181:81:   Each temp file is a columnwidth-pixels-wide column of the image.
3182:82:   By dividing the image up into columns like this, we can speed
3183:83:   up access to the image. The data in the temp file is contigous
3184:84:   scan lines with no header.
3185:85: */
3186:86:
3187:87:/*first get the image width and height and find the file format */
3188:88:
3189:89:    if ((blockofrows = loadgreytiff (imagefile, &width, &height, \
3190:90:                                     0, 1, MONITOR_GAMMA)))
3191:91:        fileformat = 1;
3192:92:    else if ((blockofrows = loadtarga2grey (imagefile, &width, &height, 0, 1)))
3193:93:        fileformat = 2;
3194:94:    else {
3195:95:        Cerrordialog (CMain, 20, 20, " Load Image ", \
3196:96:                      " Unsupported image file format ");
3197:97:        goto error;
3198:98:    }
3199:99:    free (blockofrows);
3200:100:
3201:101:    image->width = width;
3202:102:    image->height = height;
3203:103:
3204:104:    numcolumns = (width + COLUMNWIDTH - 1) / COLUMNWIDTH;
3205:105:    numblocks = (height + BLOCKHEIGHT - 1) / BLOCKHEIGHT;
3206:106:
3207:107:    image->columnwidth = COLUMNWIDTH;
3208:108:    image->numcolumns = numcolumns;
3209:109:
3210:110:    image->temp = Cmalloc (numcolumns * sizeof (int));
3211:111:    image->buffers = Cmalloc (numcolumns * sizeof (unsigned char *));
3212:112:    image->linestart = Cmalloc (numcolumns * sizeof (long));
3213:113:    image->lineend = Cmalloc (numcolumns * sizeof (long));
3214:114:
3215:115:    printf ("loading image: res=(%ldx%ld)\n", width, height);
3216:116:
3217:117:/*open files and set up caches */
3218:118:    for (i = 0; i < numcolumns; i++) {
3219:119:        for (j = strlen (imagefile); j > 0; j--)
3220:120:            if (imagefile[j] == '/')
3221:121:                break;
3222:122:        if (imagefile[j] != '/') {
```

```
3223:123:            memmove (imagefile + 1, imagefile + j, strlen (imagefile + j) + 1);
3224:124:            imagefile[0] = '/';
3225:125:            j = 0;
3226:126:        }
3227:127:        sprintf (tempstr, "%s%s%ld.temp", TEMPDIR, imagefile + j, i);
3228:128:        if ((image->temp[i] = open (tempstr, O_RDWR)) == -1) {
3229:129:            createtempfiles = 1;
3230:130:            if ((image->temp[i] = open (tempstr, O_RDWR | O_CREAT | O_TRUNC,
3231:131:                        S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP)) == -1) {
3232:132:                printf ("%s\n", tempstr);
3233:133:                Cerror (" cannot open temporary file %s ");
3234:134:            }
3235:135:        } else {
3236:136:/********** ask user if they want to use temp files, or create new ones.*/
3237:137:            /*tempfiles exist this flag says don't need to write them out */
3238:138:        }
3239:139:/*********/
3240:140:        image->buffers[i] = Cmalloc (LINESCACHED * COLUMNWIDTH);
3241:141:        image->linestart[i] = 0;
3242:142:        /* linestart = lineend = 0 will force a cache */
3243:143:        image->lineend[i] = 0;  /*load on the first call to hugegetpixel */
3244:144:    }
3245:145:
3246:146:    if (createtempfiles)
3247:147:        printf ("Creating new temp column files.\n");
3248:148:    else
3249:149:        printf ("Using existing temp column files.\n");
3250:150:
3251:151:    progresswin = Cdrawwindow ("loadprogw", CMain, 50, 50, 300, 110, "");
3252:152:
3253:153:    if (createtempfiles) {
3254:154:        cache = Cmalloc (BLOCKHEIGHT * COLUMNWIDTH);
3255:155:        for (i = 0; i < numblocks; i++) {
3256:156:            bheight = height - i * BLOCKHEIGHT;
3257:157:            if (bheight > BLOCKHEIGHT)
3258:158:                bheight = BLOCKHEIGHT;
3259:159:            blockofrows = NULL;
3260:160:
3261:161:
3262:162:            Cdrawprogress ("loadprogbar", progresswin, 20, 20, 260, 25,
3263:163:                           i * 65535 / numblocks);
3264:164:            Ccheckifevent (NULL, NULL);
3265:165:
3266:166:            switch (fileformat) {
3267:167:            case 1:
3268:168:                blockofrows = loadgreytiff (imagefile, &width, &height,
3269:169:                                            i * BLOCKHEIGHT, \
3270:170:                                i * BLOCKHEIGHT + bheight, MONITOR_GAMMA);
3271:171:                break;
3272:172:            case 2:
3273:173:                blockofrows = loadtarga2grey (imagefile, &width, &height,
3274:174:                                i * BLOCKHEIGHT, i * BLOCKHEIGHT + bheight);
3275:175:                break;
3276:176:            }
3277:177:
3278:178:            for (k = 0; k < numcolumns; k++) {
3279:179:                for (j = 0; j < bheight; j++)
3280:180:                    memcpy (cache + j * COLUMNWIDTH,
3281:181:                            blockofrows + j * width + k * COLUMNWIDTH, COLUMNWIDTH);
3282:182:                write (image->temp[k], cache, COLUMNWIDTH * bheight);
3283:183:            }
3284:184:            if (blockofrows)
3285:185:                free (blockofrows);
3286:186:        }
3287:187:        free (cache);
3288:188:    }
3289:189:    Cundrawwidget ("loadprogw");
3290:190:
3291:191:    free (imagefile);
3292:192:    return 0;
3293:193:  error:;
3294:194:    free (imagefile);
3295:195:    return 1;
3296:196:}
3297:197:
3298:198:/* This just frees all the stuff allocated in loadhugeimage */
3299:199:void freehugeimage (HugeImage * image)
3300:200:{
3301:201:    int i;
3302:202:
```

```
3303:203:    if (image->has_been_allocated != 314159265)
3304:204:        return;
3305:205:
3306:206:    if (image->temp)
3307:207:        for (i = 0; i < image->numcolumns; i++)
3308:208:            if (image->temp[i] >= 0)  /*do you get zero file handles? */
3309:209:                close (image->temp[i]);
3310:210:
3311:211:    if (image->buffers)
3312:212:        for (i = 0; i < image->numcolumns; i++)
3313:213:            if (image->buffers[i])
3314:214:                free (image->buffers[i]);
3315:215:
3316:216:    if (image->temp)
3317:217:        free (image->temp);
3318:218:    if (image->buffers)
3319:219:        free (image->buffers);
3320:220:    if (image->linestart)
3321:221:        free (image->linestart);
3322:222:    if (image->lineend)
3323:223:        free (image->lineend);
3324:224:    memset (image, 0, sizeof (HugeImage));
3325:225:}
3326:226:
3327:227:
3328:228:void destroy_huge (CWidget * w)
3329:229:{
3330:230:    if (CUserOf (w)) {
3331:231:        freehugeimage (CUserOf (w));
3332:232:        free (CUserOf (w));
3333:233:        CUserOf (w) = 0;
3334:234:    }
3335:235:}
3336:236:
3337:237:
3338:238:
3339:239:/*
3340:240:   Draw a huge image. The image is scaled (i.e. scaled by proper interpolation
3341:241:   NOT by just leaving out scanlines) to the size specified (width, height).
3342:242:   if width is zero image is scaled to height and width set adjusted to
3343:243:   preserve aspect ratio. Similarly if height is zero.
3344:244:
3345:245:   At the moment this handles only tiff files as discribed in imagewidget.c
3346:246:   loadgreytiff(...) **********
3347:247: */
3348:248:/*, HugeImage * returnimage */
3349:249:
3350:250:CWidget *Cdrawhugebwimage (const char *identifier, Window parent, int x, int y,
3351:251:                           long *width, long *height, const char *imagefile)
3352:252:{
3353:253:    Window progresswin;
3354:254:    CWidget *wdt;
3355:255:    unsigned char *data;
3356:256:/* check what happens to 'data' memory for mallocing and freeing***** */
3357:257:    HugeImage *image;
3358:258:    long i, j;
3359:259:    long pixel;
3360:260:    unsigned long xfrac, yfrac;
3361:261:    unsigned long xs1, ys1, xs2, ys2, xp, yp, area;
3362:262:    char fstr[256];
3363:263:    long w, h;
3364:264:
3365:265:    image = Cmalloc (sizeof (HugeImage));
3366:266:
3367:267:    if (loadhugeimage (image, imagefile)) {
3368:268:        free (image);
3369:269:        return NULL;
3370:270:    }
3371:271:    if (!(*width | *height)) {
3372:272:        *width = image->width;
3373:273:        *height = image->height;
3374:274:
3375:275:        data = Cmalloc (*width * *height);
3376:276:
3377:277:        for (j = 0; j < *height; j++)
3378:278:            for (i = 0; i < *width; i++)
3379:279:                data[i + j * *width] = hugegetpixel (image, i, j);
3380:280:
3381:281:    } else {
3382:282:        sprintf (fstr, "%s.thumb.tga", imagefile);
```

```
3383:283:        data = loadtarga2grey (fstr, &w, &h, 0, 1 << 30);
3384:284:
3385:285:        if (!data) {            /*a previously created thumbnail image does not exist, so... */
3386:286:            printf (\
3387:287:                    "No thumbnail image exists, creating thumbnail image from original.\n");
3388:288:            if (!*width)
3389:289:                *width = image->width * (*height) / image->height;
3390:290:            if (!*height)
3391:291:                *height = image->height * (*width) / image->width;
3392:292:
3393:293:            data = Cmalloc ((*width + 20) * *height);
3394:294:
3395:295:#define PFRAC 256
3396:296:#define PSH 8
3397:297:
3398:298:            area = (double) image->width * image->height * PFRAC * PFRAC / \
3399:299:                (*height * *width);
3400:300:
3401:301:            progresswin = Cdrawwindow ("loadprogw", CMain, 50, 50, 300, 110, "");
3402:302:
3403:303:            for (j = 0; j < *height; j++) {
3404:304:
3405:305:                Cdrawprogress ("loadprogbar", progresswin, 20, 20, 260, 25,
3406:306:                               j * 65535 / (*height));
3407:307:                Ccheckifevent (NULL, NULL);
3408:308:                for (i = 0; i < *width; i++) {
3409:309:                    pixel = 0;  /*pixel holds the accumulated sum */
3410:310:
3411:311:/*the following rectangle must be averaged; it bounds an unscaled pixel: */
3412:312:                    xs1 = ((i * image->width) << PSH) / (*width);
3413:313:                    xs2 = (((i + 1) * image->width) << PSH) / (*width);
3414:314:                    ys1 = ((j * image->height) << PSH) / (*height);
3415:315:                    ys2 = (((j + 1) * image->height) << PSH) / (*height);
3416:316:
3417:317:/*
3418:318:   yfrac is the proportion of the pixel that can be seen if
3419:319:   the pixel crosses the rectangle border. It is out of 1 << PSH = PFRAC
3420:320:   and is therefore only less than PFRAC for border pixels.
3421:321: */
3422:322:
3423:323:                    yfrac = PFRAC - (ys1 & (PFRAC - 1));
3424:324:
3425:325:                    for (yp = (ys1 >> PSH); yp < (ys2 >> PSH); yp++) {
3426:326:                        xfrac = PFRAC - (xs1 & (PFRAC - 1));
3427:327:                        for (xp = (xs1 >> PSH); xp < (xs2 >> PSH); xp++) {
3428:328:                            pixel += xfrac * yfrac * hugegetpixel (image, xp, yp);
3429:329:                            xfrac = PFRAC;
3430:330:                            /* inner pixels can all be seen entirely */
3431:331:                        }
3432:332:/*do last pixel in row */
3433:333:                        if ((xfrac = (xs2 & (PFRAC - 1))))
3434:334:                            pixel += xfrac * yfrac * hugegetpixel (image, xp, yp);
3435:335:                        yfrac = PFRAC;
3436:336:                    }
3437:337:
3438:338:/*now do the last row of the rectangle */
3439:339:                    if ((yfrac = (ys2 & (PFRAC - 1)))) {
3440:340:                        xfrac = PFRAC - (xs1 & (PFRAC - 1));
3441:341:                        for (xp = (xs1 >> PSH); xp < (xs2 >> PSH); xp++) {
3442:342:                            pixel += xfrac * yfrac * hugegetpixel (image, xp, yp);
3443:343:                            xfrac = PFRAC;
3444:344:                        }
3445:345:/*do last pixel in row */
3446:346:                        xfrac = (xs2 & (PFRAC - 1));
3447:347:                        pixel += xfrac * yfrac * hugegetpixel (image, xp, yp);
3448:348:                    }
3449:349:                    data[i + j * *width] = (long) pixel / area;
3450:350:                }
3451:351:            }
3452:352:
3453:353:            Cundrawwidget ("loadprogw");
3454:354:            writetarga (data, fstr, *width, *height, 1);
3455:355:            /* save the thumbnail image for later */
3456:356:        } else {                /* thumbnail image exists and is stored in data so do nothing */
3457:357:            printf ("Thumbnail image exists, loading.\n");
3458:358:            *width = w;
3459:359:            *height = h;
3460:360:        }
3461:361:    }
3462:362:
```

```
3463:363:    wdt = Cdrawbwimage (identifier, parent, x, y,
3464:364:                        *width, *height, data);
3465:365:
3466:366:    CUserOf (wdt) = (void *) image;
3467:367:    wdt->destroy = destroy_huge;
3468:368:    return wdt;
3469:369:}
3470:370:
3471:371:
3472:372:void Crenderzoombox (CWidget * wdt, int x, int y, int rendw, int rendh)
3473:373:{
3474:374:    int w = wdt->width;
3475:375:    int h = wdt->height;
3476:376:    Window win = wdt->winid;
3477:377:    int xim, yim, xwin, ywin;
3478:378:
3479:379:    xim = x - 2;
3480:380:    yim = y - 2;
3481:381:    xwin = x;
3482:382:    ywin = y;
3483:383:    if (xim < 0) {
3484:384:        rendw += xim;
3485:385:        xim = 0;
3486:386:        xwin = 2;
3487:387:    }
3488:388:    if (yim < 0) {
3489:389:        rendh += yim;
3490:390:        yim = 0;
3491:391:        ywin = 2;
3492:392:    }
3493:393:    XPutImage (CDisplay, win, CGC, wdt->ximage,
3494:394:               xim, yim, xwin, ywin, rendw, rendh);
3495:395:
3496:396:    Crenderbevel (win, 0, 0, w - 1, h - 1, 2, 1);
3497:397:
3498:398:}
3499:399:
3500:400:
3501:401:
3502:402:/*
3503:403:   x, y is position in huge image. w, h is data size, zoom is
3504:404:   pixel width of a hugeimage pixel i.e. the enlargement factor.
3505:405: */
3506:406:void extractfromhugeimage (HugeImage * image, unsigned char *data,
3507:407:                           long x, long y, int width, int height, int zoom)
3508:408:{                                 /*this must be tested */
3509:409:    long i, j;
3510:410:    long c, l, m, k;
3511:411:    unsigned char *q = data;
3512:412:
3513:413:/*wanna optimise more than this? I'm sure there's room. */
3514:414:    if (zoom)
3515:415:        if (zoom == 1) {
3516:416:            for (j = y; j < height + y; j++)
3517:417:                for (i = x; i < width + x; i++) {
3518:418:                    *(q++) = hugegetpixel (image, i, j);
3519:419:                }
3520:420:        } else if (zoom == 2) {
3521:421:            for (j = 0; j < height; j += 2)
3522:422:                for (i = 0; i < width; i += 2) {
3523:423:                    data[k = (i + j * width)] = c = \
3524:424:                        hugegetpixel (image, (i >> 1) + x, (j >> 1) + y);
3525:425:                    data[k + 1] = c;
3526:426:                    data[(k++) + width] = c;
3527:427:                    data[k + width] = c;
3528:428:                }
3529:429:        } else if (zoom == 3) {
3530:430:            for (j = 0; j < height; j += 3)
3531:431:                for (i = 0; i < width; i += 3) {
3532:432:                    data[k = (i + j * width)] = c = \
3533:433:                        hugegetpixel (image, i / 3 + x, j / 3 + y);
3534:434:                    data[k + 1] = c;
3535:435:                    data[k + 2] = c;
3536:436:                    k += width;
3537:437:                    data[k] = c;
3538:438:                    data[k + 1] = c;
3539:439:                    data[k + 2] = c;
3540:440:                    k += width;
3541:441:                    data[k] = c;
3542:442:                    data[k + 1] = c;
```

```
3543:443:                    data[k + 2] = c;
3544:444:                }
3545:445:        } else if (zoom == 4) {
3546:446:            for (j = 0; j < height; j += 4)
3547:447:                for (i = 0; i < width; i += 4) {
3548:448:                    data[k = (i + j * width)] = c = \
3549:449:                        hugegetpixel (image, (i >> 2) + x, (j >> 2) + y);
3550:450:                    data[k + 1] = c;
3551:451:                    data[k + 2] = c;
3552:452:                    data[k + 3] = c;
3553:453:                    k += width;
3554:454:                    data[k] = c;
3555:455:                    data[k + 1] = c;
3556:456:                    data[k + 2] = c;
3557:457:                    data[k + 3] = c;
3558:458:                    k += width;
3559:459:                    data[k] = c;
3560:460:                    data[k + 1] = c;
3561:461:                    data[k + 2] = c;
3562:462:                    data[k + 3] = c;
3563:463:                    k += width;
3564:464:                    data[k] = c;
3565:465:                    data[k + 1] = c;
3566:466:                    data[k + 2] = c;
3567:467:                    data[k + 3] = c;
3568:468:                }
3569:469:        } else {
3570:470:            for (j = 0; j < height; j += zoom)
3571:471:                for (i = 0; i < width; i += zoom) {
3572:472:                    c = hugegetpixel (image, i / zoom + x, j / zoom + y);
3573:473:                    k = i + j * width;
3574:474:                    for (m = 0; m < zoom; m++, k += width)
3575:475:                        for (l = 0; l < zoom; l++)
3576:476:                            data[k + l] = c;
3577:477:                }
3578:478:        }
3579:479:}
3580:480:
3581:481:CWidget *Cdrawzoombox (const char *identifier, const char *main, \
3582:482:                    Window parent, int x, int y, long width, long height, \
3583:483:                         long posx, long posy, int zoom)
3584:484:{
3585:485:    CWidget *w;
3586:486:    unsigned char *data = Cmalloc (width * height);
3587:487:    HugeImage *hi = (HugeImage *) CUserOf (Cwidget (main));
3588:488:    extractfromhugeimage (hi, data, posx, posy, width, height, zoom);
3589:489:
3590:490:    w = Cdrawbwimage (identifier, parent, x, y,
3591:491:                      width, height, data);
3592:492:    CUserOf (w) = hi;
3593:493:/*    w->destroy = destroy_huge; */
3594:494:    free (data);
3595:495:    return w;
3596:496:}
3597:497:
3598:498:
3599:499:CWidget *Cupdatezoombox (const char *identifier, long posx, long posy, int zoom)
3600:500:{
3601:501:    Window win;
3602:502:    unsigned char *data;
3603:503:    long width, height;
3604:504:    HugeImage *hi;
3605:505:
3606:506:    CWidget *w = Cwidget (identifier);
3607:507:    hi = (HugeImage *) CUserOf (w);
3608:508:
3609:509:    win = w->winid;
3610:510:    width = w->width;
3611:511:    height = w->height;
3612:512:
3613:513:    data = Cmalloc (width * height);
3614:514:
3615:515:    extractfromhugeimage (hi, data, posx, posy, width - 4, height - 4, zoom);
3616:516:
3617:517:/* as with Cdrawbwimage: */
3618:518:    greyscaletopix (w->ximage->data, data, width - 4, height - 4, \
3619:519:                    w->ximage->bits_per_pixel / 8);
3620:520:
3621:521:    Cexpose (identifier);
3622:522:
```

```
3623:523:    free (data);
3624:524:    return w;
3625:525:}
3626:526:
3627:527:
3628:528:
3629:529:long CHugeImageRealWidth (const char *ident)
3630:530:{
3631:531:    return ((HugeImage *) CUserOf (Cwidget (ident)))->width;
3632:532:}
3633:533:
3634:534:
3635:535:long CHugeImageRealHeight (const char *ident)
3636:536:{
3637:537:    return ((HugeImage *) CUserOf (Cwidget (ident)))->height;
3638:538:}
3639:1:#ifndef HUG_IMAGE_H
3640:2:#define HUG_IMAGE_H
3641:3:
3642:4:typedef struct {
3643:5:    long width;
3644:6:    long height;
3645:7:    int *temp;
3646:8:    long *linestart;            /* first line cached in buffer (inclusive of this line) */
3647:9:    long *lineend;              /* last line cached in buffer (exclusive of this line) */
3648:10:    unsigned char **buffers;    /* buffer */
3649:11:    int columnwidth;
3650:12:    long numcolumns;
3651:13:    unsigned long has_been_allocated;
3652:14:} HugeImage;
3653:15:
3654:16:#define BLOCKHEIGHT 64
3655:17:/*the number of rows we are going to read from the file
3656:18:   at a time. Small to reduce memory requirements, large to be
3657:19:   faster */
3658:20:#define COLUMNSHIFT 7
3659:21:#define COLUMNWIDTH 128
3660:22:#define COLUMNMASK 127
3661:23:
3662:24:/*This must be about the size as the maximum mask you
3663:25:   might want to use on the image. i.e. if you are going to
3664:26:   readily access pixel(i,j) followed by pixel(i, j+4) then
3665:27:   choose 5. If you are only going to readily access
3666:28:   pixel(i+1,j) after pixel(i,j) the choose 1.
3667:29:   In any even, mask must also be accessed from left to right. */
3668:30:#define LINESCACHED 128
3669:31:
3670:32:void hugeloadnextbuf (HugeImage * image, long x, long y, int i);
3671:33:
3672:34:/*getpixel gets a pixel grey value (0-255) from a huge image.
3673:35:   It exploits a caching mechanism to improve speed. This is a little
3674:36:   faster than the caching mechanism used by the FILE type.
3675:37:   The caching mechanism works best when pixels are accessed from left to
3676:38:   right in the image. See LINESCACHED above. */
3677:39:
3678:40:CWidget *Cdrawhugebwimage (const char *identifier, Window parent, int x, int y,
3679:41:                           long *width, long *height, const char *imagefile);
3680:42:void Crenderzoombox (CWidget * w, int x, int y, int rendw, int rendh);
3681:43:void extractfromhugeimage (HugeImage * image, unsigned char *data,
3682:44:                            long x, long y, int width, int height, int zoom);
3683:45:CWidget *Cdrawzoombox (const char *identifier, const char *main, \
3684:46:                           Window parent, int x, int y, \
3685:47:                   long width, long height, long posx, long posy, int zoom);
3686:48:CWidget *Cupdatezoombox (const char *identifier, long posx, long posy, int zoom);
3687:49:long CHugeImageRealHeight (const char *ident);
3688:50:long CHugeImageRealWidth (const char *ident);
3689:51:
3690:52:#endif
3691:1:/*****************************************************************************************/
3692:2:/* imagefit.c - commands to fit objects to desktop markers                                */
3693:3:/*****************************************************************************************/
3694:4:
3695:5:#include "display.h"
3696:6:#include "main/imagefit.h"
3697:7:#include "app_glob.c"
3698:8:#include "hugeimage.h"
3699:9:#include "widget3d.h"
3700:10:#include "main/marker.h"
3701:11:#include "main/displaycam.h"
3702:12:#include "picsetup.h"
```

```
3703:13:#include "dialog.h"
3704:14:#include "stringtools.h"
3705:15:#include "callback.h"
3706:16:#include "matrix.h"
3707:17:#include "output.h"
3708:18:
3709:19:/*


 */
3710:20:
3711:21:extern Desktop desktop;
3712:22:
3713:23:Matrix *get_3d_points (Desktop * d)
3714:24:{
3715:25:    return 0;
3716:26:}
3717:27:
3718:28:
3719:29:#define LOTS 30000
3720:30:
3721:31:static int get_min_points (Marker * m)
3722:32:{
3723:33:    int i, min = LOTS;
3724:34:    for (i = 0; m[i].v; i++)
3725:35:        if (m[i].n < min)
3726:36:            min = m[i].n;
3727:37:    return min;
3728:38:}
3729:39:
3730:40:
3731:41:/* minimise an overconstrained nx3 matrix */
3732:42:Vec vec_invert (Vec * s, double *r, int n)
3733:43:{
3734:44:    int i;
3735:45:    double a[4][3] =
3736:46:    {
3737:47:        {0, 0, 0},
3738:48:        {0, 0, 0},
3739:49:        {0, 0, 0},
3740:50:        {0, 0, 0}};
3741:51:    if (n < 3) {
3742:52:        fprintf (stderr, \
3743:53:        "stereo:%s:%d: call to vec_invert with less than three rows\n", \
3744:54:                 __FILE__, __LINE__);
3745:55:        abort ();
3746:56:    }
3747:57:    if (n > 3) {
3748:58:        for (i = 0; i < n; i++) {  /* s transpose multiplied by s */
3749:59:            a[0][0] += s[i].x * s[i].x;
3750:60:            a[0][1] += s[i].y * s[i].x;
3751:61:            a[0][2] += s[i].z * s[i].x;
3752:62:            a[1][0] += s[i].x * s[i].y;
3753:63:            a[1][1] += s[i].y * s[i].y;
3754:64:            a[1][2] += s[i].z * s[i].y;
3755:65:            a[2][0] += s[i].x * s[i].z;
3756:66:            a[2][1] += s[i].y * s[i].z;
3757:67:            a[2][2] += s[i].z * s[i].z;
3758:68:            a[3][0] += r[i] * s[i].x;
3759:69:            a[3][1] += r[i] * s[i].y;
3760:70:            a[3][2] += r[i] * s[i].z;
3761:71:        }
3762:72:        return solve3x3 (a);
3763:73:    }
3764:74:    a[0][0] = s[0].x;
3765:75:    a[0][1] = s[1].x;
3766:76:    a[0][2] = s[2].x;
3767:77:    a[1][0] = s[0].y;
3768:78:    a[1][1] = s[1].y;
3769:79:    a[1][2] = s[2].y;
3770:80:    a[2][0] = s[0].z;
3771:81:    a[2][1] = s[1].z;
3772:82:    a[2][2] = s[2].z;
3773:83:    a[3][0] = r[0];
3774:84:    a[3][1] = r[1];
3775:85:    a[3][2] = r[2];
3776:86:    return solve3x3 (a);
3777:87:}
3778:88:
3779:89:
```

```
3780:90:
3781:91:
3782:92:Vec triangulate_camera_point (double *x, double *y, Camera ** c, int n)
3783:93:{
3784:94:/* Here we find the best estimate for the points position: a linear problem. */
3785:95:    int i, j;
3786:96:    Vec *s, v;
3787:97:    double *r;
3788:98:    r = Cmalloc ((n * 2) * sizeof (double));
3789:99:    s = Cmalloc ((n * 2) * sizeof (Vec));
3790:100:    j = 0;
3791:101:    for (i = 0; i < n; i++) {
3792:102:        s[j] = plus (times (c[i]->m_s, x[i]), times (c[i]->m_x, c[i]->f));
3793:103:        r[j] = dot (s[j], c[i]->x);
3794:104:        j++;
3795:105:        s[j] = plus (times (c[i]->m_s, y[i]), times (c[i]->m_y, c[i]->f));
3796:106:        r[j] = dot (s[j], c[i]->x);
3797:107:        j++;
3798:108:    }
3799:109:    v = vec_invert (s, r, j);
3800:110:    free (s);
3801:111:    free (r);
3802:112:    return v;
3803:113:}
3804:114:
3805:115:
3806:116:
3807:117:Vec *triangulate_image_points (Marker * m, int *n)
3808:118:{
3809:119:    Vec *v;
3810:120:    int num_sets = 0, i, j;
3811:121:    double *x, *y;
3812:122:    Camera **c;
3813:123:    *n = get_min_points (m);
3814:124:    while (m[num_sets].v)
3815:125:        num_sets++;
3816:126:
3817:127:    c = Cmalloc (num_sets * sizeof (Camera *));
3818:128:    v = Cmalloc (*n * sizeof (Vec));
3819:129:    x = Cmalloc (num_sets * sizeof (double));
3820:130:    y = Cmalloc (num_sets * sizeof (double));
3821:131:
3822:132:    for (i = 0; i < num_sets; i++)
3823:133:        c[i] = m[i].cam;
3824:134:
3825:135:    for (j = 0; j < *n; j++) {
3826:136:        for (i = 0; i < num_sets; i++) {
3827:137:            x[i] = m[i].v[j].x;
3828:138:            y[i] = m[i].v[j].y;
3829:139:            imagetocamera (c[i], x[i], y[i]);
3830:140:        }
3831:141:        v[j] = triangulate_camera_point (x, y, c, num_sets);
3832:142:    }
3833:143:    free (y);
3834:144:    free (x);
3835:145:    free (c);
3836:146:    return v;
3837:147:}
3838:148:
3839:149:
3840:150:/* alignment **** */
3841:151:/* this object may be free'd with just free */
3842:152:Object *get_surface (Marker * m, int w, int h)
3843:153:{
3844:154:    Vec *v;
3845:155:    int n;
3846:156:    Object *o;
3847:157:
3848:158:    v = triangulate_image_points (m, &n);
3849:159:    if (n != w * h) {
3850:160:        free (v);
3851:161:        Cerrordialog (0, 0, 0, " Get surface ", \
3852:162:                      " width * height not the same as the number of points triangulated \n" \
3853:163:          " check that you have a representative marker in each image ");
3854:164:        return 0;
3855:165:    }
3856:166:    o = Cmalloc (sizeof (Object) + n * sizeof (Vec));
3857:167:    o->surface.p = (Vec *) ((char *) o + sizeof (Object));
3858:168:    memcpy (o->surface.p, v, n * sizeof (Vec));
3859:169:    free (v);
```

```
3860:170:    o->surface.w = w;
3861:171:    o->surface.h = h;
3862:172:    o->type = SURFACE;
3863:173:    return o;
3864:174:}
3865:175:
3866:176:void width_and_height_query (int *w, int *h)
3867:177:{
3868:178:    CEvent e;
3869:179:    Window size_query = Cdrawwindow ("surfsizeQ", CMain, 20, 20, 320, \
3870:180:                                     145, "surfsizeQ");
3871:181:
3872:182:    Cdrawbitmapbutton ("surfsizeQ.close", size_query, 136, 93,
3873:183:                       40, 40, Ccolor (6), C_FLAT, tick_bits);
3874:184:
3875:185:    Cdrawtext ("", size_query, 4, 4, "Enter width\n");
3876:186:    Cdrawtext ("", size_query, 4, 32, "Enter height\n");
3877:187:    Cdrawtextinput ("surfsizeQ.width", size_query, 120, 4,
3878:188:                    50, 20, 20, "");
3879:189:    Cdrawtextinput ("surfsizeQ.height", size_query, 120, 32,
3880:190:                    50, 20, 20, "");
3881:191:
3882:192:    do {
3883:193:        CNextEvent (NULL, &e);
3884:194:    } while (strcmp (e.ident, "surfsizeQ.close"));
3885:195:
3886:196:    *w = atoi (Cgettext ("surfsizeQ.width"));
3887:197:    *h = atoi (Cgettext ("surfsizeQ.height"));
3888:198:
3889:199:    Cundrawwidget ("surfsizeQ");
3890:200:}
3891:201:
3892:202:
3893:203:Object *fit_surface (Desktop * d)
3894:204:{
3895:205:    int w, h;
3896:206:    Marker *m = 0;
3897:207:    Object *o;
3898:208:    if (!check_markers (d, 2, LOTS, 4, LOTS, " Fit Surface "))
3899:209:        return 0;
3900:210:    width_and_height_query (&w, &h);
3901:211:    if ((w | h)) {
3902:212:        m = get_all_markers (d);
3903:213:        o = get_surface (m, w, h);
3904:214:        free (m);
3905:215:        return o;
3906:216:    }
3907:217:    free (m);
3908:218:    return 0;
3909:219:}
3910:220:
3911:221:
3912:222:void output_surface (Desktop * d)
3913:223:{
3914:224:    Object *o;
3915:225:    o = fit_surface (d);
3916:226:    if (o) {
3917:227:        output_object (o);
3918:228:        free (o);
3919:229:    }
3920:230:}
3921:231:
3922:232:Circle circle_from_points (Vec * v, int n)
3923:233:{
3924:234:    Circle c;
3925:235:    double *x, *y, *z;
3926:236:    int i;
3927:237:    x = Cmalloc (sizeof (double) * n * 3);
3928:238:    y = x + n;
3929:239:    z = y + n;
3930:240:    for (i = 0; i < n; i++) {
3931:241:        x[i] = v[i].x;
3932:242:        y[i] = v[i].y;
3933:243:        z[i] = v[i].z;
3934:244:    }
3935:245:
3936:246:    c = circlefrompoints (x, y, z, n);
3937:247:    free (x);
3938:248:    return c;
3939:249:}
```

```
3940:250:
3941:251:Cylinder cylinder_from_points (Vec * v, int n)
3942:252:{
3943:253:    Cylinder c;
3944:254:    double *x, *y, *z;
3945:255:    int i;
3946:256:    x = Cmalloc (sizeof (double) * n * 3);
3947:257:    y = x + n;
3948:258:    z = y + n;
3949:259:    for (i = 0; i < n; i++) {
3950:260:        x[i] = v[i].x;
3951:261:        y[i] = v[i].y;
3952:262:        z[i] = v[i].z;
3953:263:    }
3954:264:
3955:265:    c = cylinderfrompoints (x, y, z, n);
3956:266:    free (x);
3957:267:    return c;
3958:268:}
3959:269:
3960:270:
3961:271:LineSegment line_from_points (Vec * v, int n)
3962:272:{
3963:273:    LineSegment l;
3964:274:    double *x, *y, *z;
3965:275:    int i;
3966:276:    x = Cmalloc (sizeof (double) * n * 3);
3967:277:    y = x + n;
3968:278:    z = y + n;
3969:279:    for (i = 0; i < n; i++) {
3970:280:        x[i] = v[i].x;
3971:281:        y[i] = v[i].y;
3972:282:        z[i] = v[i].z;
3973:283:    }
3974:284:
3975:285:    l = linesegmentfrompoints (x, y, z, n);
3976:286:    free (x);
3977:287:    return l;
3978:288:}
3979:289:
3980:290:Ellipse ellipse_from_points (Vec * v, int n)
3981:291:{
3982:292:    Ellipse e;
3983:293:    double *x, *y, *z;
3984:294:    int i;
3985:295:    x = Cmalloc (sizeof (double) * n * 3);
3986:296:    y = x + n;
3987:297:    z = y + n;
3988:298:    for (i = 0; i < n; i++) {
3989:299:        x[i] = v[i].x;
3990:300:        y[i] = v[i].y;
3991:301:        z[i] = v[i].z;
3992:302:    }
3993:303:
3994:304:    e = ellipsefrompoints (x, y, z, n);
3995:305:    free (x);
3996:306:    return e;
3997:307:}
3998:308:
3999:309:
4000:310:Object *fit_circle (Desktop * d)
4001:311:{
4002:312:    int n;
4003:313:    Vec *v;
4004:314:    Marker *m;
4005:315:    Object *o;
4006:316:    if (!check_markers (d, 2, LOTS, 3, LOTS, " Fit Circle "))
4007:317:        return 0;
4008:318:    m = get_all_markers (d);
4009:319:    v = triangulate_image_points (m, &n);
4010:320:    o = Cmalloc (sizeof (Object));
4011:321:    o->circle = circle_from_points (v, n);
4012:322:    if (o->type != CIRCLE) {
4013:323:        free (o);
4014:324:        o = 0;
4015:325:    }
4016:326:    free (v);
4017:327:    free (m);
4018:328:    return o;
4019:329:}
```

```
4020:330:
4021:331:
4022:332:Object *fit_ellipse (Desktop * d)
4023:333:{
4024:334:    int n;
4025:335:    Vec *v;
4026:336:    Marker *m;
4027:337:    Object *o;
4028:338:    if (!check_markers (d, 2, LOTS, 3, LOTS, " Fit Circle "))
4029:339:        return 0;
4030:340:    m = get_all_markers (d);
4031:341:    v = triangulate_image_points (m, &n);
4032:342:    o = Cmalloc (sizeof (Object));
4033:343:    o->ellipse = ellipse_from_points (v, n);
4034:344:    if (o->type != ELLIPSE) {
4035:345:        free (o);
4036:346:        o = 0;
4037:347:    }
4038:348:    free (v);
4039:349:    free (m);
4040:350:    return o;
4041:351:}
4042:352:
4043:353:Object *fit_cylinder (Desktop * d)
4044:354:{
4045:355:    int n;
4046:356:    Vec *v;
4047:357:    Marker *m;
4048:358:    Object *o;
4049:359:    if (!check_markers (d, 2, LOTS, 3, LOTS, " Fit Circle "))
4050:360:        return 0;
4051:361:    m = get_all_markers (d);
4052:362:    v = triangulate_image_points (m, &n);
4053:363:    o = Cmalloc (sizeof (Object));
4054:364:    o->cylinder = cylinder_from_points (v, n);
4055:365:    if (o->type != CYLINDER) {
4056:366:        free (o);
4057:367:        o = 0;
4058:368:    }
4059:369:    free (v);
4060:370:    free (m);
4061:371:    return o;
4062:372:}
4063:373:
4064:374:
4065:375:void output_circle (Desktop * d)
4066:376:{
4067:377:    Object *o;
4068:378:    o = fit_circle (d);
4069:379:    if (o) {
4070:380:        output_object (o);
4071:381:        free (o);
4072:382:    }
4073:383:}
4074:384:
4075:385:
4076:386:void output_ellipse (Desktop * d)
4077:387:{
4078:388:    Object *o;
4079:389:    o = fit_ellipse (d);
4080:390:    if (o) {
4081:391:        output_object (o);
4082:392:        free (o);
4083:393:    }
4084:394:}
4085:395:
4086:396:/* this object may be free'd with just free */
4087:397:Object *triangulate_single_point (Desktop * d)
4088:398:{
4089:399:    Vec *v;
4090:400:    Object *o;
4091:401:    Marker *m;
4092:402:    int n;
4093:403:
4094:404:    o = Cmalloc (sizeof (Object));
4095:405:    if (!check_markers (d, 2, LOTS, 1, 1, " Get Point "))
4096:406:        return 0;
4097:407:    m = get_all_markers (d);
4098:408:    v = triangulate_image_points (m, &n);
4099:409:    free (m);
```

```
4100:410:    o->point.p = v[0];
4101:411:    free (v);
4102:412:    o->type = POINT;
4103:413:    return o;
4104:414:}
4105:415:
4106:416:void output_point (Desktop * d)
4107:417:{
4108:418:    Object *o;
4109:419:    o = triangulate_single_point (d);
4110:420:    if (o) {
4111:421:        output_object (o);
4112:422:        free (o);
4113:423:    }
4114:424:}
4115:425:
4116:426:void output_cylinder (Desktop * d)
4117:427:{
4118:428:    Object *o;
4119:429:    o = fit_cylinder (d);
4120:430:    if (o) {
4121:431:        output_object (o);
4122:432:        free (o);
4123:433:    }
4124:434:}
4125:435:
4126:436:
4127:437:Object *fit_line (Desktop * d)
4128:438:{
4129:439:    int n;
4130:440:    Vec *v;
4131:441:    Marker *m;
4132:442:    Object *o;
4133:443:    if (!check_markers (d, 2, LOTS, 2, LOTS, " Fit Circle "))
4134:444:        return 0;
4135:445:    m = get_all_markers (d);
4136:446:    v = triangulate_image_points (m, &n);
4137:447:    o = Cmalloc (sizeof (Object));
4138:448:    o->line = line_from_points (v, n);
4139:449:    if (o->type != LINE_SEGMENT) {
4140:450:        free (o);
4141:451:        o = 0;
4142:452:    }
4143:453:    free (v);
4144:454:    free (m);
4145:455:    return o;
4146:456:}
4147:457:
4148:458:
4149:459:void output_line (Desktop * d)
4150:460:{
4151:461:    Object *o;
4152:462:    o = fit_line (d);
4153:463:    if (o) {
4154:464:        output_object (o);
4155:465:        free (o);
4156:466:    }
4157:467:}
4158:468:
4159:469:
4160:470:
4161:471:static int dividepointsintotwolines (double *x, double *y, \
4162:472:                                     int n, double **x1, double **y1, int *n1, double **x2, double **
4163:473:{
4164:474:    LineSegment l;
4165:475:    int i = 2;
4166:476:    Vec v =
4167:477:    {0, 0, 0};
4168:478:
4169:479:    *x1 = NULL;
4170:480:    *y1 = NULL;
4171:481:    *x2 = NULL;
4172:482:    *y2 = NULL;
4173:483:    *n1 = 0;
4174:484:    *n2 = 0;
4175:485:
4176:486:    while (i < n) {
4177:487:        l = linesegmentfrompoints (x, y, NULL, i);
4178:488:        v.x = x[i];
4179:489:        v.y = y[i];
```

```
4180:490:        if (distancetoline (v, &l) > DIVIDE_THRESHOLD) {
4181:491:            *n1 = i;
4182:492:            *x1 = Cmalloc (i * sizeof (double));
4183:493:            memcpy (*x1, x, i * sizeof (double));
4184:494:            *y1 = Cmalloc (i * sizeof (double));
4185:495:            memcpy (*y1, y, i * sizeof (double));
4186:496:            *n2 = n - i;
4187:497:            *x2 = Cmalloc ((*n2) * sizeof (double));
4188:498:            memcpy (*x2, x + i, (*n2) * sizeof (double));
4189:499:            *y2 = Cmalloc ((*n2) * sizeof (double));
4190:500:            memcpy (*y2, y + i, (*n2) * sizeof (double));
4191:501:            return 1;
4192:502:        }
4193:503:        i++;
4194:504:    }
4195:505:    return 0;
4196:506:}
4197:507:
4198:508:
4199:509:/*
4200:510:   gets markers from the first two marked images only
4201:511:   and returns them as array x1,... length n1, n2. Also removes
4202:512:   camera distortion
4203:513: */
4204:514:void get_two_sets_of_markers (Marker * m, double **x1, \
4205:515:                 double **y1, int *n1, double **x2, double **y2, int *n2)
4206:516:{
4207:517:    int i;
4208:518:    *n1 = m[0].n;
4209:519:    *n2 = m[1].n;
4210:520:    *x1 = Cmalloc (*n1 * sizeof (double));
4211:521:    *y1 = Cmalloc (*n1 * sizeof (double));
4212:522:    *x2 = Cmalloc (*n2 * sizeof (double));
4213:523:    *y2 = Cmalloc (*n2 * sizeof (double));
4214:524:
4215:525:    for (i = 0; i < *n1; i++) {
4216:526:        (*x1)[i] = m[0].v[i].x;
4217:527:        (*y1)[i] = m[0].v[i].y;
4218:528:        imagetocamera (m[0].cam, (*x1)[i], (*y1)[i]);
4219:529:    }
4220:530:
4221:531:    for (i = 0; i < *n2; i++) {
4222:532:        (*x2)[i] = m[1].v[i].x;
4223:533:        (*y2)[i] = m[1].v[i].y;
4224:534:        imagetocamera (m[1].cam, (*x2)[i], (*y2)[i]);
4225:535:    }
4226:536:}
4227:537:
4228:538:
4229:539:Object *fit_edge_line (Desktop * d)
4230:540:{
4231:541:    double *x1, *y1, *y2, *x2;
4232:542:    int n1, n2;
4233:543:    Marker *m;
4234:544:    Object *o;
4235:545:
4236:546:    if (!check_markers (d, 2, 2, 2, LOTS, " Fit Line "))
4237:547:        return 0;
4238:548:    o = Cmalloc (sizeof (Object));
4239:549:
4240:550:    m = get_all_markers (d);
4241:551:    get_two_sets_of_markers (m, &x1, &y1, &n1, &x2, &y2, &n2);
4242:552:    o->line = linefromstereoedge (x1, y1, n1, x2, y2, n2, m[0].cam, m[1].cam);
4243:553:    free (x1);
4244:554:    free (y1);
4245:555:    free (x2);
4246:556:    free (y2);
4247:557:
4248:558:    if (o->type != LINE_SEGMENT) {
4249:559:        free (o);
4250:560:        o = 0;
4251:561:    }
4252:562:    free (m);
4253:563:    return o;
4254:564:}
4255:565:
4256:566:Object *fit_edge_circle (Desktop * d)
4257:567:{
4258:568:    double *x1, *y1, *y2, *x2;
4259:569:    int n1, n2;
```

```
4260:570:    Marker *m;
4261:571:    Object *o;
4262:572:
4263:573:    if (!check_markers (d, 2, 2, 3, LOTS, " Fit Circle "))
4264:574:        return 0;
4265:575:    o = Cmalloc (sizeof (Object));
4266:576:
4267:577:    m = get_all_markers (d);
4268:578:    get_two_sets_of_markers (m, &x1, &y1, &n1, &x2, &y2, &n2);
4269:579:    o->circle = circlefromstereoedge (x1, y1, n1, x2, y2, n2, m[0].cam, m[1].cam);
4270:580:    free (x1);
4271:581:    free (y1);
4272:582:    free (x2);
4273:583:    free (y2);
4274:584:
4275:585:    if (o->type != CIRCLE) {
4276:586:        free (o);
4277:587:        o = 0;
4278:588:    }
4279:589:    free (m);
4280:590:    return o;
4281:591:}
4282:592:
4283:593:
4284:594:Object *fit_edge_cylinder (Desktop * d)
4285:595:{
4286:596:    double *xl, *yl, *yr, *xr;
4287:597:    int nl, nr;
4288:598:    Marker *m;
4289:599:    Object *o;
4290:600:    double *x1l, *y1l, *x2l, *y2l, *x1r, *y1r, *x2r, *y2r;
4291:601:    int n1l, n2l, n1r, n2r;
4292:602:
4293:603:    if (!check_markers (d, 2, 2, 2, LOTS, " Fit Circle "))
4294:604:        return 0;
4295:605:    o = Cmalloc (sizeof (Object));
4296:606:    clear (o, Object);
4297:607:
4298:608:    m = get_all_markers (d);
4299:609:    get_two_sets_of_markers (m, &xl, &yl, &nl, &xr, &yr, &nr);
4300:610:
4301:611:    if (dividepointsintotwolines (xl, yl, nl, &x1l, &y1l, &n1l, &x2l, &y2l, &n2l)) {
4302:612:        if (dividepointsintotwolines (xr, yr, nr, &x1r, &y1r, &n1r, &x2r, &y2r, \
4303:613:                                      &n2r)) {
4304:614:            o->cylinder = cylinderfromstereoedge (x1l, y1l, n1l, x2l, y2l, n2l,
4305:615:                        x1r, y1r, n1r, x2r, y2r, n2r, m[0].cam, m[1].cam);
4306:616:            free (x1l);
4307:617:            free (y1l);
4308:618:            free (x2l);
4309:619:            free (y2l);
4310:620:        }
4311:621:        free (x1r);
4312:622:        free (y1r);
4313:623:        free (x2r);
4314:624:        free (y2r);
4315:625:    }
4316:626:    free (xl);
4317:627:    free (yl);
4318:628:    free (xr);
4319:629:    free (yr);
4320:630:
4321:631:    if (o->type != CYLINDER) {
4322:632:        free (o);
4323:633:        o = 0;
4324:634:    }
4325:635:    free (m);
4326:636:    return o;
4327:637:}
4328:638:
4329:639:void output_line_edge (Desktop * d)
4330:640:{
4331:641:    Object *o;
4332:642:    o = fit_edge_line (d);
4333:643:    if (o) {
4334:644:        output_object (o);
4335:645:        free (o);
4336:646:    }
4337:647:}
4338:648:
4339:649:void output_circle_edge (Desktop * d)
```

```
4340:650:{
4341:651:    Object *o;
4342:652:    o = fit_edge_circle (d);
4343:653:    if (o) {
4344:654:        output_object (o);
4345:655:        free (o);
4346:656:    }
4347:657:}
4348:658:
4349:659:void output_cylinder_edge (Desktop * d)
4350:660:{
4351:661:    Object *o;
4352:662:    o = fit_edge_cylinder (d);
4353:663:    if (o) {
4354:664:        output_object (o);
4355:665:        free (o);
4356:666:    }
4357:667:}
4358:1:#ifndef _IMAGE_FIT
4359:2:#define _IMAGE_FIT
4360:3:
4361:4:#define EDITOR Cwidget("editor")->editor
4362:5:#define DIVIDE_THRESHOLD 10
4363:6:#define LINE_THICKNES_3D (scale_units_for_3d / 1000)
4364:7:
4365:8:Vec vec_invert (Vec * s, double *r, int n);
4366:9:
4367:10:void output_surface (Desktop * d);
4368:11:void output_circle (Desktop * d);
4369:12:void output_line (Desktop * d);
4370:13:void output_line_edge (Desktop * d);
4371:14:void output_circle_edge (Desktop * d);
4372:15:void output_cylinder_edge (Desktop * d);
4373:16:void output_cylinder (Desktop * d);
4374:17:
4375:18:int cb_getcircle (CWidget * none, XEvent * xevent, CEvent * cwevent);
4376:19:int cb_getlinefrom3D (CWidget * none, XEvent * xevent, CEvent * cwevent);
4377:20:int cb_getpoint (CWidget * none, XEvent * xevent, CEvent * cwevent);
4378:21:int cb_getline (CWidget * none, XEvent * xevent, CEvent * cwevent);
4379:22:int cb_getcylinder (CWidget * none, XEvent * xevent, CEvent * cwevent);
4380:23:
4381:24:#endif                              /* _IMAGE_FIT */
4382:1:/*********************************************************************************************/
4383:2:/* imagehandler.c - handles events from the view windows                                     */
4384:3:/*********************************************************************************************/
4385:4:
4386:5:#include "display.h"
4387:6:#include "marker.h"
4388:7:#include "app_glob.c"
4389:8:#include "hugeimage.h"
4390:9:#include "widget3d.h"
4391:10:#include "main/marker.h"
4392:11:#include "main/displaycam.h"
4393:12:#include "picsetup.h"
4394:13:#include "dialog.h"
4395:14:#include "stringtools.h"
4396:15:#include "callback.h"
4397:16:
4398:17:
4399:18:/* check zoom box motion */
4400:19:static int check_motion (Picture * image, CEvent * cwevent)
4401:20:{
4402:21:    if (!CPending (CDisplay)) {
4403:22:        Csetdrawingtarget (image->zoom_rect->ident);
4404:23:        Cremovepp (0);
4405:24:        Cdrawpicrectangle ((cwevent->x - 1) - (cwevent->x - 1) % image->mag - 1, \
4406:25:                           (cwevent->y - 1) - (cwevent->y - 1) % image->mag - 1, image->mag + 1, \
4407:26:                           image->mag + 1, Ccolor (19));
4408:27:        return 1;
4409:28:    } else
4410:29:        return 0;
4411:30:}
4412:31:
4413:32:
4414:33:static int check_button_release (Picture * image, CEvent * cwevent)
4415:34:{
4416:35:    double x, y;
4417:36:    int xzoom, yzoom;
4418:37:    long p, q;
4419:38:    float oldmag = image->mag;
```

```
4420:39:    xzoom = cwevent->x / image->mag;
4421:40:    yzoom = cwevent->y / image->mag;
4422:41:
4423:42:    x = (double) (image->xzoom + xzoom) * image->width / image->real_width;
4424:43:    y = (double) (image->yzoom + yzoom) * image->width / image->real_width;
4425:44:
4426:45:    image->mag <<= 1;           /*cycle zooming with right mouse button */
4427:46:    if (image->mag > 16)
4428:47:        image->mag = 1;
4429:48:
4430:49:    q = ZOOMSIZE / (2 * image->mag);
4431:50:    p = q * image->width / image->real_width;
4432:51:
4433:52:    image->xzoom = (double) x *image->real_width / image->width - q;
4434:53:    image->yzoom = (double) y *image->real_width / image->width - q;
4435:54:
4436:55:    Csetdrawingtarget (image->main_rect->ident);
4437:56:    Cremovepp (0);
4438:57:    Cdrawpicrectangle (x - p - 2, y - p - 2, p * 2 + 1, p * 2 + 1, Ccolor (19));
4439:58:
4440:59:    Csetdrawingtarget (image->zoom_rect->ident);
4441:60:    Cremovepp (0);
4442:61:
4443:62:    Csetdrawingtarget (image->zoom_markers->ident);
4444:63:    Cscalepicture ((float) image->mag / oldmag);
4445:64:    Csetwidgetposition (image->zoom_markers->ident, \
4446:65:         -image->xzoom * image->mag + 2, -image->yzoom * image->mag + 2);
4447:66:    Cupdatezoombox (image->zoom_image->ident, image->xzoom, image->yzoom, \
4448:67:                    image->mag);
4449:68:    return 1;
4450:69:}
4451:70:
4452:71:static int check_button_drag (Picture * image, CEvent * cwevent)
4453:72:{
4454:73:    int xzoom, yzoom;
4455:74:    long p, q;
4456:75:    xzoom = cwevent->x;
4457:76:    yzoom = cwevent->y;
4458:77:    q = ZOOMSIZE / (2 * image->mag);
4459:78:    p = q * image->width / image->real_width;
4460:79:
4461:80:    image->xzoom = xzoom * image->real_width / image->width - q;
4462:81:    image->yzoom = yzoom * image->real_width / image->width - q;
4463:82:
4464:83:    if (!CPending (CDisplay) || !(cwevent->type == MotionNotify)) {
4465:84:        Csetdrawingtarget (image->zoom_markers->ident);
4466:85:        Csetwidgetposition (image->zoom_markers->ident, \
4467:86:         -image->xzoom * image->mag + 2, -image->yzoom * image->mag + 2);
4468:87:
4469:88:        Csetdrawingtarget (image->main_rect->ident);
4470:89:        Cremovepp (0);
4471:90:        Cdrawpicrectangle (xzoom - p - 2, yzoom - p - 2, p * 2 + 1, p * 2 + 1, \
4472:91:                           Ccolor (19));
4473:92:
4474:93:        Csetdrawingtarget (image->zoom_rect->ident);
4475:94:        Cremovepp (0);
4476:95:
4477:96:        Cupdatezoombox (image->zoom_image->ident, image->xzoom, image->yzoom, \
4478:97:                        image->mag);
4479:98:    }
4480:99:    return 1;
4481:100:}
4482:101:
4483:102:
4484:103:int handle_zoom_box (Desktop * d, CEvent * cwevent)
4485:104:{
4486:105:    Picture *p;
4487:106:
4488:107:    set_current_from_pointer (d, cwevent);
4489:108:    p = &(d->view[d->current_view].pic);
4490:109:
4491:110:    if (cwevent->state & ControlMask) {
4492:111:        if ((cwevent->button == Button1
4493:112:             && (cwevent->type == ButtonRelease
4494:113:                 || cwevent->type == ButtonPress))
4495:114:            || (cwevent->state & Button1Mask && cwevent->type == MotionNotify))
4496:115:            move_marker (p, cwevent);
4497:116:
4498:117:        if (cwevent->button == Button2 && cwevent->type == ButtonRelease)
4499:118:            remove_marker (p, cwevent);
```

```
4500:119:    } else {
4501:120:        if (cwevent->button == Button1 && cwevent->type == ButtonRelease)
4502:121:            insert_marker (p, cwevent);
4503:122:
4504:123:        if (cwevent->button == Button2 && cwevent->type == ButtonRelease)
4505:124:            check_button_release (p, cwevent);
4506:125:    }
4507:126:
4508:127:    if (cwevent->type == MotionNotify || cwevent->type == ButtonRelease)
4509:128:        return check_motion (p, cwevent);
4510:129:
4511:130:    return 0;
4512:131:}
4513:132:
4514:133:
4515:134:int handle_main_box (Desktop * d, CEvent * cwevent)
4516:135:{
4517:136:    Picture *p;
4518:137:
4519:138:    set_current_from_pointer (d, cwevent);
4520:139:    p = &(d->view[d->current_view].pic);
4521:140:
4522:141:    if (cwevent->button == Button1 || (cwevent->state & Button1Mask))
4523:142:        return check_button_drag (p, cwevent);
4524:143:
4525:144:    return 0;
4526:145:}
4527:1:#ifndef _IMAGE_HANDLER_H
4528:2:#define _IMAGE_HANDLER_H
4529:3:
4530:4:int handle_zoom_box (Desktop * d, CEvent * cwevent);
4531:5:int handle_main_box (Desktop * d, CEvent * cwevent);
4532:6:
4533:7:#endif  /* _IMAGE_HANDLER_H */
4534:1:/*********************************************************************************************/
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                                    */
4537:4:/*********************************************************************************************/
4538:5:
4539:6:#include <config.h>
4540:7:#include "global.h"
4541:8:#include "stdio.h"
4542:9:#include "display.h"
4543:10:#include "main/imagefit.h"
4544:11:#include "app_glob.c"
4545:12:#include "hugeimage.h"
4546:13:#include "widget3d.h"
4547:14:#include "main/marker.h"
4548:15:#include "main/displaycam.h"
4549:16:#include "picsetup.h"
4550:17:#include "dialog.h"
4551:18:#include "stringtools.h"
4552:19:#include "callback.h"
4553:20:#include "matrix.h"
4554:21:#include "output.h"
4555:22:#include "imagefit.h"
4556:23:
4557:24:#define NORMAL_LENGTH 256
4558:25:
4559:26:
4560:27:static inline double normal (Vec x)
4561:28:{
4562:29:    double l;
4563:30:    l = norm (x);
4564:31:    return l == 0 ? 1e-10 : l;
4565:32:}
4566:33:
4567:34:#define norm(x) normal(x)
4568:35:
4569:36:
4570:37:/*
4571:38:   This is an isolated file to do two things to an 3D object.
4572:39:
4573:40:   The first is to join together edges of a surface that are close.
4574:41:   So if two surfaces are adjacent, but not quite touching, this will
4575:42:   move the edges so that they touch exactly.
4576:43:
4577:44:   The other thing it does is to set all the normal vectors perfectly.
4578:45:   Adjacent surfaces, if their edges have normal that different by less
4579:46:   than AngleTheshold, then the normals are made the same at the edges.
```

```
4580:47:   This avoid bad shadowing on edges.
4581:48: */
4582:49:
4583:50:
4584:51:struct close_line {
4585:52:    TD_Surface *s;
4586:53:    int i, j, dir;
4587:54:    double d;
4588:55:};
4589:56:
4590:57:
4591:58:/* returns 1 if on the line, zero otherwise, [requires p1, u, and l] */
4592:59:int ispointonline (Vec x, LineSegment * l);
4593:60:
4594:61:/* [requires l->p1, l->u] */
4595:62:double distancetoline (Vec x, LineSegment * l);
4596:63:
4597:64:
4598:65:/*
4599:66:
4600:67:   width ---->
4601:68:
4602:69:   |
4603:70:   length  |
4604:71:   V
4605:72:
4606:73: */
4607:74:
4608:75:#define DIR_RIGHT 0
4609:76:#define DIR_LEFT 1
4610:77:#define DIR_DOWN 2
4611:78:#define DIR_UP 3
4612:79:
4613:80:Vec surfpoint (TD_Surface * s, int i, int j)
4614:81:{
4615:82:    Vec x;
4616:83:    TD_Point *p;
4617:84:
4618:85:    if (i < 0 || i >= s->w || j < 0 || j >= s->l) {
4619:86:        printf ("surfpoint aborting\n");
4620:87:        abort ();
4621:88:    }
4622:89:    p = s->point + i + j * s->w;
4623:90:    x.x = p->x;
4624:91:    x.y = p->y;
4625:92:    x.z = p->z;
4626:93:    return x;
4627:94:}
4628:95:
4629:96:void insertsurfpoint (Vec x, TD_Surface * s, int i, int j)
4630:97:{
4631:98:    TD_Point *p;
4632:99:
4633:100:    if (i < 0 || i >= s->w || j < 0 || j >= s->l) {
4634:101:        printf ("insertsurfpoint aborting\n");
4635:102:        abort ();
4636:103:    }
4637:104:    p = s->point + i + j * s->w;
4638:105:    p->x = x.x;
4639:106:    p->y = x.y;
4640:107:    p->z = x.z;
4641:108:}
4642:109:
4643:110:
4644:111:
4645:112:LineSegment getlinefromsurface (TD_Surface * s, int i, int j, int dir)
4646:113:{
4647:114:    LineSegment l;
4648:115:    TD_Point *p;
4649:116:
4650:117:    if (i < 0 || i >= s->w || j < 0 || j >= s->l) {
4651:118:        printf ("getlinefromsurface aborting\n");
4652:119:        abort ();
4653:120:    }
4654:121:    p = s->point + i + j * s->w;
4655:122:    l.p1.x = p->x;
4656:123:    l.p1.y = p->y;
4657:124:    l.p1.z = p->z;
4658:125:    switch (dir) {
4659:126:    case DIR_RIGHT:
```

```
4660:127:        i++;
4661:128:        p += 1;
4662:129:        break;
4663:130:    case DIR_LEFT:
4664:131:        i--;
4665:132:        p -= 1;
4666:133:        break;
4667:134:    case DIR_DOWN:
4668:135:        j++;
4669:136:        p += s->w;
4670:137:        break;
4671:138:    case DIR_UP:
4672:139:        j--;
4673:140:        p -= s->w;
4674:141:    }
4675:142:
4676:143:    if (i < 0 || i >= s->w || j < 0 || j >= s->l) {
4677:144:        printf ("getlinefromsurface aborting\n");
4678:145:        abort ();
4679:146:    }
4680:147:    l.p2.x = p->x;
4681:148:    l.p2.y = p->y;
4682:149:    l.p2.z = p->z;
4683:150:    l.u = minus (l.p2, l.p1);
4684:151:    if (norm (l.u) == 0) {
4685:152:        l.u.x = 10e-10;
4686:153:        l.u.y = 10e-10;
4687:154:        l.u.z = 10e-10;
4688:155:    }
4689:156:    l.l = norm (l.u);
4690:157:    l.u = times (l.u, 1 / l.l);
4691:158:    return l;
4692:159:}
4693:160:
4694:161:
4695:162:#define IS_MIN \
4696:163:    l = getlinefromsurface (s, i, j, dir); \
4697:164:    if (ispointonline (x, &l)) { \
4698:165:        d = distancetoline (x, &l); \
4699:166:        if (d < dmin) { \
4700:167:            dmin = d; \
4701:168:            imin = i; \
4702:169:            jmin = j; \
4703:170:            dirmin = dir; \
4704:171:        } \
4705:172:    } \
4706:173:
4707:174:
4708:175:void checksurf (TD_Surface * s, Vec x, struct close_line *cl)
4709:176:{
4710:177:    LineSegment l;
4711:178:    double d, dmin;
4712:179:    int i, j, dir, dirmin = 0, imin = 0, jmin = 0;
4713:180:
4714:181:    dmin = 9e90;
4715:182:
4716:183:    j = 0;
4717:184:    i = 0;
4718:185:    dir = DIR_RIGHT;
4719:186:    for (; i < s->w - 1; i++) {
4720:187:        l = getlinefromsurface (s, i, j, dir);
4721:188:        if (ispointonline (x, &l)) {
4722:189:            d = distancetoline (x, &l);
4723:190:            if (d < dmin) {
4724:191:                dmin = d;
4725:192:                imin = i;
4726:193:                jmin = j;
4727:194:                dirmin = dir;
4728:195:            }
4729:196:        }
4730:197:    }
4731:198:
4732:199:    j = 0;
4733:200:    i = s->w - 1;
4734:201:    dir = DIR_DOWN;
4735:202:    for (; j < s->l - 1; j++) {
4736:203:        l = getlinefromsurface (s, i, j, dir);
4737:204:        if (ispointonline (x, &l)) {
4738:205:            d = distancetoline (x, &l);
4739:206:            if (d < dmin) {
```

```
4740:207:                dmin = d;
4741:208:                imin = i;
4742:209:                jmin = j;
4743:210:                dirmin = dir;
4744:211:            }
4745:212:        }
4746:213:    }
4747:214:
4748:215:    j = s->l - 1;
4749:216:    i = 0;
4750:217:    dir = DIR_RIGHT;
4751:218:    for (; i < s->w - 1; i++) {
4752:219:        l = getlinefromsurface (s, i, j, dir);
4753:220:        if (ispointonline (x, &l)) {
4754:221:            d = distancetoline (x, &l);
4755:222:            if (d < dmin) {
4756:223:                dmin = d;
4757:224:                imin = i;
4758:225:                jmin = j;
4759:226:                dirmin = dir;
4760:227:            }
4761:228:        }
4762:229:    }
4763:230:
4764:231:    j = 0;
4765:232:    i = 0;
4766:233:    dir = DIR_DOWN;
4767:234:    for (; j < s->l - 1; j++) {
4768:235:        l = getlinefromsurface (s, i, j, dir);
4769:236:        if (ispointonline (x, &l)) {
4770:237:            d = distancetoline (x, &l);
4771:238:            if (d < dmin) {
4772:239:                dmin = d;
4773:240:                imin = i;
4774:241:                jmin = j;
4775:242:                dirmin = dir;
4776:243:            }
4777:244:        }
4778:245:    }
4779:246:
4780:247:    if (dmin < cl->d) {
4781:248:        cl->d = dmin;
4782:249:        cl->dir = dirmin;
4783:250:        cl->i = imin;
4784:251:        cl->j = jmin;
4785:252:        cl->s = s;
4786:253:    }
4787:254:}
4788:255:
4789:256:
4790:257:/* 'omit' is the surface that we must NOT check that x belongs to */
4791:258:void checkobject (TD_Solid * o, Vec x, int omit, struct close_line *cl)
4792:259:{
4793:260:    int i;
4794:261:    TD_Surface *s;
4795:262:
4796:263:    for (i = 0; i < o->num_surfaces; i++) {
4797:264:        s = o->surf + i;
4798:265:        if (i == omit || !s)
4799:266:            continue;
4800:267:        checksurf (s, x, cl);
4801:268:    }
4802:269:}
4803:270:
4804:271:/*
4805:272:
4806:273:   #define CHECK_OBJECT \
4807:274:   cl.s = 0; \
4808:275:   cl.d = limit; \
4809:276:   s = o->surf + k; \
4810:277:   if (s) { \
4811:278:   checkobject (o, surfpoint (s, i, j), k, &cl); \
4812:279:   if (cl.s) \
4813:280:   (*cb) (&cl, s->point + i + j * s->w); \
4814:281:   }
4815:282:
4816:283:
4817:284: */
4818:285:/*
4819:286:   this finds the closest line to each point in object 'o' and calls cb for each point.
```

```
4820:287:   The distance to the line must however fall within limit, or else cb won't be called.
4821:288:   Only lines along edges of surfaces are considered. Each close line and point is
4822:289:   passed to cn.
4823:290: */
4824:291:void find_closest_line (TD_Solid * o, double limit, \
4825:292:                        void (*cb) (struct close_line *, TD_Point *))
4826:293:{
4827:294:    int i, j, k;
4828:295:    TD_Surface *s;
4829:296:
4830:297:    struct close_line cl;
4831:298:
4832:299:    for (k = 0; k < o->num_surfaces; k++) {
4833:300:        s = o->surf + k;
4834:301:        if (!s)
4835:302:            continue;
4836:303:        j = 0;
4837:304:        i = 0;
4838:305:        for (; i < s->w - 1; i++) {
4839:306:            cl.s = 0;
4840:307:            cl.d = limit;
4841:308:            s = o->surf + k;
4842:309:            if (s) {
4843:310:                checkobject (o, surfpoint (s, i, j), k, &cl);
4844:311:                if (cl.s)
4845:312:                    (*cb) (&cl, s->point + i + j * s->w);
4846:313:            }
4847:314:        }
4848:315:        j = 1;
4849:316:        i = 0;
4850:317:        for (; j < s->l - 1; j++) {
4851:318:            cl.s = 0;
4852:319:            cl.d = limit;
4853:320:            s = o->surf + k;
4854:321:            if (s) {
4855:322:                checkobject (o, surfpoint (s, i, j), k, &cl);
4856:323:                if (cl.s)
4857:324:                    (*cb) (&cl, s->point + i + j * s->w);
4858:325:            }
4859:326:        }
4860:327:        j = 0;
4861:328:        i = s->w - 1;
4862:329:        for (; j < s->l; j++) {
4863:330:            cl.s = 0;
4864:331:            cl.d = limit;
4865:332:            s = o->surf + k;
4866:333:            if (s) {
4867:334:                checkobject (o, surfpoint (s, i, j), k, &cl);
4868:335:                if (cl.s)
4869:336:                    (*cb) (&cl, s->point + i + j * s->w);
4870:337:            }
4871:338:        }
4872:339:        j = s->l - 1;
4873:340:        i = 0;
4874:341:        for (; i < s->w; i++) {
4875:342:            cl.s = 0;
4876:343:            cl.d = limit;
4877:344:            s = o->surf + k;
4878:345:            if (s) {
4879:346:                checkobject (o, surfpoint (s, i, j), k, &cl);
4880:347:                if (cl.s)
4881:348:                    (*cb) (&cl, s->point + i + j * s->w);
4882:349:            }
4883:350:        }
4884:351:    }
4885:352:}
4886:353:
4887:354:
4888:355:/* this moves the point half way to the closest line */
4889:356:void cb_join (struct close_line *cl, TD_Point * p)
4890:357:{
4891:358:    LineSegment l;
4892:359:    Vec x, y;
4893:360:    x.x = p->x;
4894:361:    x.y = p->y;
4895:362:    x.z = p->z;
4896:363:    l = getlinefromsurface (cl->s, cl->i, cl->j, cl->dir);
4897:364:    y = pointonline (x, &l);
4898:365:    x = times (plus (x, y), 0.5);
4899:366:    p->x = x.x;
```

```
4900:367:    p->y = x.y;
4901:368:    p->z = x.z;
4902:369:}
4903:370:
4904:371:/*
4905:372:   this takes the point and its closest line from the edge of the nearest surface
4906:373:   and computes the mean normal. This smooths the rendering.
4907:374: */
4908:375:static double norm_threshold;
4909:376:void cb_norm (struct close_line *cl, TD_Point * p)
4910:377:{
4911:378:    int dir2;
4912:379:    int d = 0;
4913:380:    Vec x, y;
4914:381:    double a;
4915:382:    if (cl->dir == DIR_LEFT || cl->dir == DIR_RIGHT) {
4916:383:        if (cl->j)
4917:384:            dir2 = DIR_UP;
4918:385:        else
4919:386:            dir2 = DIR_DOWN;
4920:387:        x = getlinefromsurface (cl->s, cl->i, cl->j, cl->dir).u;
4921:388:        y = getlinefromsurface (cl->s, cl->i, cl->j, dir2).u;
4922:389:    } else {
4923:390:        d = 1;
4924:391:        if (cl->i)
4925:392:            dir2 = DIR_LEFT;
4926:393:        else
4927:394:            dir2 = DIR_RIGHT;
4928:395:        x = getlinefromsurface (cl->s, cl->i, cl->j, dir2).u;
4929:396:        y = getlinefromsurface (cl->s, cl->i, cl->j, cl->dir).u;
4930:397:    }
4931:398:
4932:399:    if (dir2 + cl->dir == 3)    /* quick way to check clockwise/anti-clockwise */
4933:400:        y = cross (y, x);
4934:401:    else
4935:402:        y = cross (x, y);
4936:403:    x.x = p->dirx;
4937:404:    x.y = p->diry;
4938:405:    x.z = p->dirz;
4939:406:    x = times (x, 1 / norm (x));
4940:407:    y = times (y, 1 / norm (y));
4941:408:    a = acos (norm (minus (x, y)) / 2);
4942:409:    if (a < norm_threshold) {
4943:410:        x = plus (x, y);        /* average them */
4944:411:        x = times (x, NORMAL_LENGTH / norm (x));
4945:412:        /* normalise to NORMAL_LENGTH */
4946:413:        p->dirx = x.x;          /* store the point */
4947:414:        p->diry = x.y;
4948:415:        p->dirz = x.z;
4949:416:    }
4950:417:}
4951:418:
4952:419:void cb_both (struct close_line *cl, TD_Point * p)
4953:420:{
4954:421:    cb_join (cl, p);
4955:422:    cb_norm (cl, p);
4956:423:}
4957:424:
4958:425:
4959:426:/*
4960:427:   t is the threshold. When surface edge points are less than t away from the closest
4961:428:   edge, they are deemed to be 'touching', and the surface edges are zipped together.
4962:429: */
4963:430:void join_up_surface_edges (TD_Solid * o, double t)
4964:431:{
4965:432:    find_closest_line (o, t, cb_join);
4966:433:}
4967:434:
4968:435:/*
4969:436:   t is the threshold. When surface edge points panels are more than an angle t
4970:437:   from the panel of the closest edge, they are deemed to be not 'touching',
4971:438:   and the surface normals are not averaged.
4972:439: */
4973:440:void join_up_normals (TD_Solid * o, double t, double a)
4974:441:{
4975:442:    norm_threshold = a;
4976:443:    find_closest_line (o, t, cb_norm);
4977:444:}
4978:445:
4979:446:void join_up_normals_and_surface_edges (TD_Solid * o, double t, double a)
```

```
4980:447:{
4981:448:    norm_threshold = a;
4982:449:    find_closest_line (o, t, cb_both);
4983:450:}
4984:451:
4985:452:static inline Vec unit (Vec v)
4986:453:{
4987:454:    return times (v, norm (v));
4988:455:}
4989:456:
4990:457:void calc_all_normals (TD_Solid * o)
4991:458:{
4992:459:    Vec n;
4993:460:    TD_Point *p;
4994:461:    int i, j, k;
4995:462:    TD_Surface *s;
4996:463:
4997:464:    for (k = 0; k < o->num_surfaces; k++) {
4998:465:        s = o->surf + k;
4999:466:        if (!s)
5000:467:            continue;
5001:468:        p = s->point;
5002:469:        for (j = 0; j < s->l; j++) {
5003:470:            for (i = 0; i < s->w; i++) {
5004:471:                if (i < s->w - 1 && j < s->l - 1)
5005:472:                    n = unit (cross (getlinefromsurface (s, i, j, DIR_RIGHT).u, \
5006:473:                                     getlinefromsurface (s, i, j, DIR_DOWN).u));
5007:474:                if (i < s->w - 1 && j > 0)
5008:475:                    n = plus (n, unit (cross (\
5009:476:                                       getlinefromsurface (s, i, j, DIR_UP).u, \
5010:477:                                   getlinefromsurface (s, i, j, DIR_RIGHT).u)));
5011:478:                if (i > 0 && j > 0)
5012:479:                    n = plus (n, unit (cross (\
5013:480:                                     getlinefromsurface (s, i, j, DIR_LEFT).u, \
5014:481:                                      getlinefromsurface (s, i, j, DIR_UP).u)));
5015:482:                if (i > 0 && j < s->l - 1)
5016:483:                    n = plus (n, unit (cross (\
5017:484:                                     getlinefromsurface (s, i, j, DIR_DOWN).u, \
5018:485:                                    getlinefromsurface (s, i, j, DIR_LEFT).u)));
5019:486:                n = times (n, NORMAL_LENGTH / norm (n));
5020:487:                p->dirx = n.x;
5021:488:                p->diry = n.y;
5022:489:                (p++)->dirz = n.z;
5023:490:            }
5024:491:        }
5025:492:    }
5026:493:}
5027:1:#ifndef _JOIN_H
5028:2:#define _JOIN_H
5029:3:
5030:4:void join_up_surface_edges (TD_Solid * o, double t);
5031:5:void join_up_normals (TD_Solid * o, double t, double a);
5032:6:void join_up_normals_and_surface_edges (TD_Solid * o, double t, double a);
5033:7:void calc_all_normals (TD_Solid * o);
5034:8:
5035:9:#endif
5036:1:
5037:2:/*****************************************************************************/
5038:3:/* loadcalfile.c - loads a calibration file into the desktop structure       */
5039:4:/*****************************************************************************/
5040:5:
5041:6:#include <config.h>
5042:7:#include "global.h"
5043:8:#include <stdlib.h>
5044:9:#include <stdio.h>
5045:10:#include <sys/types.h>
5046:11:
5047:12:#ifdef HAVE_UNISTD_H
5048:13:#include <unistd.h>
5049:14:#endif
5050:15:
5051:16:#include <my_string.h>
5052:17:#include <sys/stat.h>
5053:18:
5054:19:#ifdef HAVE_FCNTL_H
5055:20:#include <fcntl.h>
5056:21:#endif
5057:22:
5058:23:#include <stdlib.h>
5059:24:#include <stdarg.h>
```

```
5060:25:
5061:26:#ifdef HAVE_SYS_TIME_H
5062:27:#include <sys/time.h>
5063:28:#endif
5064:29:
5065:30:#include "regex.h"
5066:31:
5067:32:#include "display.h"
5068:33:#include "dialog.h"
5069:34:#include "my_string.h"
5070:35:#include "stringtools.h"
5071:36:
5072:37:#define MAX_CAL_POINTS 256
5073:38:
5074:39:int load_calibration_file (Desktop * d, const char *filename)
5075:40:{
5076:41:    FILE *f;
5077:42:    int n, i = 0;
5078:43:    Vec *v;
5079:44:
5080:45:    f = fopen (filename, "r");
5081:46:    if (!f) {
5082:47:        Cerrordialog (0, 0, 0, " Load Calibration File ", \
5083:48:                      get_sys_error (" Error trying to load file. "));
5084:49:        return 1;
5085:50:    }
5086:51:    v = Cmalloc (MAX_CAL_POINTS * sizeof (Vec));
5087:52:
5088:53:    while (!feof (f)) {
5089:54:        n = fscanf (f, "%lf %lf %lf ", &v[i].x, &v[i].y, &v[i].z);
5090:55:        if (n == 3) {
5091:56:            i++;
5092:57:        } else {
5093:58:            Cerrordialog (0, 0, 0, " Load Calibration File ", \
5094:59:                          " Format error in calibration file ");
5095:60:            i = 0;
5096:61:            break;
5097:62:        }
5098:63:    }
5099:64:    fclose (f);
5100:65:    if (i) {
5101:66:        d->num_cal_points = i;
5102:67:        destroy ((void *) &(d->cal_points));
5103:68:        d->cal_points = Cmalloc (i * sizeof (Vec));
5104:69:        memcpy (d->cal_points, v, i * sizeof (Vec));
5105:70:        destroy ((void *) &(d->cal_file));
5106:71:        d->cal_file = strdup (filename);
5107:72:    }
5108:73:/* else { cal points not changed */
5109:74:    destroy ((void *) &v);
5110:75:    fclose (f);
5111:76:    return !i;
5112:77:}
5113:78:
5114:79:int load_calibration (Desktop * d)
5115:80:{
5116:81:    char *s;
5117:82:    s = Cgetfile (0, 0, 0, d->image_dir, d->cal_file ? d->cal_file : "", \
5118:83:                  " Load Calibration File ");
5119:84:    if (s)
5120:85:        if (*s) {
5121:86:            load_calibration_file (d, s);
5122:87:            return 0;
5123:88:        }
5124:89:    return 1;
5125:90:}
5126:1:#ifndef _LOAD_CAL_FILE_H
5127:2:#define _LOAD_CAL_FILE_H
5128:3:
5129:4:int load_calibration (Desktop * d);
5130:5:int load_calibration_file (Desktop * d, const char *filename);
5131:6:
5132:7:#endif                          /*  _LOAD_CAL_FILE_H  */
5133:1:
5134:2:/*****************************************************************************/
5135:3:/* marker.c - marker handling and finding real pointer positions in images   */
5136:4:/*****************************************************************************/
5137:5:
5138:6:#include <config.h>
5139:7:#include "global.h"
```

```
5140:8:#include <stdlib.h>
5141:9:#include <stdio.h>
5142:10:#include <sys/types.h>
5143:11:
5144:12:#ifdef HAVE_UNISTD_H
5145:13:#include <unistd.h>
5146:14:#endif
5147:15:
5148:16:#include <my_string.h>
5149:17:#include <sys/stat.h>
5150:18:
5151:19:#ifdef HAVE_FCNTL_H
5152:20:#include <fcntl.h>
5153:21:#endif
5154:22:
5155:23:#include <stdlib.h>
5156:24:#include <stdarg.h>
5157:25:
5158:26:#ifdef HAVE_SYS_TIME_H
5159:27:#include <sys/time.h>
5160:28:#endif
5161:29:
5162:30:#include "regex.h"
5163:31:
5164:32:#include "display.h"
5165:33:#include "marker.h"
5166:34:#include "app_glob.c"
5167:35:#include "hugeimage.h"
5168:36:#include "widget3d.h"
5169:37:#include "main/marker.h"
5170:38:#include "main/displaycam.h"
5171:39:#include "picsetup.h"
5172:40:#include "dialog.h"
5173:41:#include "stringtools.h"
5174:42:#include "callback.h"
5175:43:
5176:44:#define LARGE 10e90
5177:45:
5178:46:/*


 */
5179:47:
5180:48:static int dialog_error (const char *head, const char *mess)
5181:49:{
5182:50:    Cerrordialog (0, 0, 0, head, mess);
5183:51:    return 1;
5184:52:}
5185:53:
5186:54:int num_views_with_markers (Desktop * d)
5187:55:{
5188:56:    int i, v = 0;
5189:57:    for (i = 0; i < d->num_views; i++)
5190:58:        if (exists (d, i))
5191:59:            if (d->view[i].num_marks)
5192:60:                v++;
5193:61:    return v;
5194:62:}
5195:63:
5196:64:/* return 0 if not satisifed */
5197:65:static int markers_in_bounds (Desktop * d, int min_views, int max_views, \
5198:66:                               int min_markers, int max_markers)
5199:67:{
5200:68:    int i, n, v;
5201:69:    for (i = 0; i < d->num_views; i++) {
5202:70:        if (exists (d, i)) {
5203:71:            n = d->view[i].num_marks;
5204:72:            if (n && (n < min_markers || n > max_markers))
5205:73:                return 0;
5206:74:        }
5207:75:    }
5208:76:    v = num_views_with_markers (d);
5209:77:    if (v < min_views || v > max_views)
5210:78:        return 0;
5211:79:    return 1;
5212:80:}
5213:81:
5214:82:/* returns the maximum number of markers in a view */
5215:83:int max_markers (Desktop * d)
5216:84:{
```

```
5217:85:    int i, max = 0;
5218:86:    for (i = 0; i < d->num_views; i++)
5219:87:        if (exists (d, i))
5220:88:            if (d->view[i].num_marks > max)
5221:89:                max = d->view[i].num_marks;
5222:90:    return max;
5223:91:}
5224:92:
5225:93:
5226:94:/* returned value must be freed with just free */
5227:95:Marker *get_all_markers (Desktop * d)
5228:96:{
5229:97:    int v, i, j;
5230:98:    Marker *m;
5231:99:
5232:100:    v = num_views_with_markers (d);
5233:101:    if (!v)
5234:102:        return 0;
5235:103:    m = Cmalloc ((v + 1) * sizeof (Marker));  /* allignment ? ***** */
5236:104:
5237:105:    i = 0;
5238:106:    for (j = 0; j < d->num_views; j++)
5239:107:        if (exists (d, j))
5240:108:            if (d->view[j].num_marks) {
5241:109:                m[i].cam = &d->view[j].cam;
5242:110:                m[i].v = d->view[j].mark;
5243:111:                m[i].n = d->view[j].num_marks;
5244:112:                i++;
5245:113:            }
5246:114:    m[i].v = 0;
5247:115:    m[i].n = 0;
5248:116:    return m;
5249:117:}
5250:118:
5251:119:
5252:120:
5253:121:/* returns 0 if any are not calibrated */
5254:122:int images_calibrated (Desktop * d)
5255:123:{
5256:124:    int i;
5257:125:    for (i = 0; i < d->num_views; i++)
5258:126:        if (exists (d, i))      /* exists */
5259:127:            if (d->view[i].num_marks)  /* has markers */
5260:128:                if (!d->view[i].calibrated)  /* is calibrated */
5261:129:                    return 0;
5262:130:    return 1;
5263:131:}
5264:132:
5265:133:/* reports an error message if there are not enough markers in enough views */
5266:134:/* returns 1 if all ok */
5267:135:int check_markers (Desktop * d, int min_views, int max_views, \
5268:136:                   int min_markers, int max_markers, const char *head)
5269:137:{
5270:138:    if (!d->cal_points) {
5271:139:        dialog_error (head, " First load calibration points ");
5272:140:        return 0;
5273:141:    }
5274:142:    if (!images_calibrated (d)) {
5275:143:        dialog_error (head, " Not all images are calibrated ");
5276:144:        return 0;
5277:145:    }
5278:146:    if (!d->num_views) {
5279:147:        dialog_error (head, " No views open, no markers ");
5280:148:        return 0;
5281:149:    }
5282:150:    if (!markers_in_bounds (d, min_views, max_views, min_markers, max_markers)) {
5283:151:        char *s;
5284:152:        s = sprintf_alloc (" Need %d-%d markers in each of %d-%d views ", \
5285:153:                           min_markers, max_markers, min_views, max_views);
5286:154:        dialog_error (head, s);
5287:155:        free (s);
5288:156:        return 0;
5289:157:    }
5290:158:    return 1;
5291:159:}
5292:160:
5293:161:
5294:162:/* returns 1 on error */
5295:163:int set_current_view (Desktop * d, int i)
5296:164:{
```

```
5297:165:    if (i < 0 || i >= d->num_views)
5298:166:        return 1;
5299:167:    if (!exists (d, i))
5300:168:        return 1;
5301:169:    d->current_view = i;
5302:170:    return 0;
5303:171:}
5304:172:
5305:173:void clear_markers (Desktop * d)
5306:174:{
5307:175:    d->view[d->current_view].num_marks = 0;
5308:176:}
5309:177:
5310:178:int new_marker (Desktop * d, Vec x)
5311:179:{
5312:180:    int i;
5313:181:    i = d->view[d->current_view].num_marks++;
5314:182:    d->view[d->current_view].mark[i] = x;
5315:183:    return i;
5316:184:}
5317:185:
5318:186:int remove_last_marker (Desktop * d)
5319:187:{
5320:188:    if (d->view[d->current_view].num_marks)
5321:189:        return --d->view[d->current_view].num_marks;
5322:190:    return 0;
5323:191:}
5324:192:
5325:193:void remove_vec (Vec * vec, int i, int n)
5326:194:{
5327:195:    if (i < n && i >= 0 && n > 0)
5328:196:        memmove (vec + i, vec + i + 1, (n - 1 - i) * sizeof (Vec));
5329:197:}
5330:198:
5331:199:void remove_a_marker (Desktop * d, int i)
5332:200:{
5333:201:    remove_vec (d->view[d->current_view].mark, i, \
5334:202:                d->view[d->current_view].num_marks--);
5335:203:}
5336:204:
5337:205:/* find the closest marker to x in the current view and returns its index */
5338:206:int find_closest_marker (Desktop * d, Vec x)
5339:207:{
5340:208:    int i;
5341:209:    double e = LARGE, f;
5342:210:    int min = 0;
5343:211:    if (!d->view[d->current_view].num_marks)
5344:212:        return -1;
5345:213:    for (i = 0; i < d->view[d->current_view].num_marks; i++) {
5346:214:        f = norm (minus (x, d->view[d->current_view].mark[i]));
5347:215:        if (f < e) {
5348:216:            min = i;
5349:217:            e = f;
5350:218:        }
5351:219:    }
5352:220:    return min;
5353:221:}
5354:222:
5355:223:void remove_closest_marker (Desktop * d, Vec x)
5356:224:{
5357:225:    int min;
5358:226:    min = find_closest_marker (d, x);
5359:227:    if (min >= 0)
5360:228:        remove_a_marker (d, min);
5361:229:}
5362:230:
5363:231:void move_closest_marker (Desktop * d, Vec x)
5364:232:{
5365:233:    int min;
5366:234:    min = find_closest_marker (d, x);
5367:235:    if (min >= 0)
5368:236:        d->view[d->current_view].mark[min] = x;
5369:237:}
5370:238:
5371:239:void undrawmarker (View * v)
5372:240:{
5373:241:    if (v->pic.main_markers) {
5374:242:        Csetdrawingtarget (v->pic.main_markers->ident);
5375:243:        Cclearpic ();
5376:244:        Csetdrawingtarget (v->pic.zoom_markers->ident);
```

```
5377:245:        Cclearpic ();
5378:246:    }
5379:247:}
5380:248:
5381:249:void draw_line_to_pic (Picture * p, double x1, double y1, \
5382:250:                       double x2, double y2, unsigned long c)
5383:251:{
5384:252:    imagetopic (p, x1, y1);
5385:253:    imagetopic (p, x2, y2);
5386:254:
5387:255:    Csetdrawingtarget (p->main_markers->ident);
5388:256:    Cdrawline ((float) x1 * p->width / p->real_width, \
5389:257:               (float) y1 * p->width / p->real_width, \
5390:258:               (float) x2 * p->width / p->real_width, \
5391:259:               (float) y2 * p->width / p->real_width, c);
5392:260:    Csetdrawingtarget (p->zoom_markers->ident);
5393:261:    Cdrawline (x1 * p->mag, y1 * p->mag, x2 * p->mag, y2 * p->mag, c);
5394:262:}
5395:263:
5396:264:/* draws two crosses, one in the zoom box, and on in the main image */
5397:265:static void draw_cross_to_pic (Picture * p, Vec x, int r, unsigned long c)
5398:266:{
5399:267:    draw_line_to_pic (p, x.x - r, x.y - r, x.x + r, x.y + r, c);
5400:268:    draw_line_to_pic (p, x.x + r, x.y - r, x.x - r, x.y + r, c);
5401:269:}
5402:270:
5403:271:/* draw the markers of one view */
5404:272:static void drawmarker (View * v)
5405:273:{
5406:274:    int i;
5407:275:    for (i = 0; i < v->num_marks; i++)
5408:276:        draw_cross_to_pic (&v->pic, v->mark[i], CROSS_SIZE, CROSS_COLOR);
5409:277:}
5410:278:
5411:279:/* redraw all the markers */
5412:280:void draw_markers (Desktop * d)
5413:281:{
5414:282:    int i;
5415:283:    for (i = 0; i < d->num_views; i++) {
5416:284:        if (exists (d, i)) {
5417:285:            undrawmarker (&d->view[i]);
5418:286:            drawmarker (&d->view[i]);
5419:287:        }
5420:288:    }
5421:289:}
5422:290:
5423:291:/* gets the view that the pointer is in. returns -1 if not found */
5424:292:int get_pointer_view (Desktop * d, CEvent * e)
5425:293:{
5426:294:    int i;
5427:295:    for (i = 0; i < d->num_views; i++) {
5428:296:        if (exists (d, i)) {
5429:297:            if (d->view[i].pic.main_image->winid == e->window)
5430:298:                return i;
5431:299:            if (d->view[i].pic.zoom_image->winid == e->window)
5432:300:                return i;
5433:301:            if (d->view[i].pic.main_win->winid == Cwidget (e->ident)->parentid)
5434:302:                return i;
5435:303:        }
5436:304:    }
5437:305:    return -1;
5438:306:}
5439:307:
5440:308:/* this sets the 'current_view' member to the view the pointer is in */
5441:309:/* this must be called before an marker operation on a view so that
5442:310:   the operation will occur in the correct view */
5443:311:/* (otherwise you may get markers appearing in an image other than the
5444:312:   on you clicked on, for example) */
5445:313:void set_current_from_pointer (Desktop * d, CEvent * e)
5446:314:{
5447:315:    int i;
5448:316:    i = get_pointer_view (d, e);
5449:317:    if (i >= 0)
5450:318:        set_current_view (d, i);
5451:319:}
5452:320:
5453:321:/* real position in zoom image */
5454:322:Vec get_zoom_pointer_pos (Picture * p, CEvent * ev)
5455:323:{
5456:324:    Vec x;
```

```
5457:325:    x.x = (double) p->xzoom + (double) (ev->x - 1) / p->mag;
5458:326:    x.y = (double) p->yzoom + (double) (ev->y - 1) / p->mag;;
5459:327:    x.z = 0;
5460:328:    return x;
5461:329:}
5462:330:
5463:331:/* real position in main image */
5464:332:Vec get_main_pointer_pos (Picture * p, CEvent * ev)
5465:333:{
5466:334:    Vec x;
5467:335:    x.x = (ev->x - 1) * p->real_width / p->width;
5468:336:    x.y = (ev->y - 1) * p->real_width / p->width;
5469:337:    x.z = 0;
5470:338:    return x;
5471:339:}
5472:340:
5473:341:Vec zoom_event_to_image_coord (Picture * p, CEvent * e)
5474:342:{
5475:343:    Vec v;
5476:344:    v.x = (double) p->xzoom + (double) (e->x - 1) / p->mag;
5477:345:    v.y = (double) p->yzoom + (double) (e->y - 1) / p->mag;
5478:346:    pictoimage (p, v.x, v.y);
5479:347:    v.z = 0;
5480:348:    return v;
5481:349:}
5482:1:#ifndef _MARKER_H
5483:2:#define _MARKER_H
5484:3:
5485:4:#include "quickmath.h"
5486:5:#include "display.h"
5487:6:
5488:7:#define CROSS_SIZE 10
5489:8:#define CROSS_COLOR Ccolor(6)
5490:9:
5491:10:/* checks if a view at that index has not been destroyed */
5492:11:#define exists(d,i) ((d)->view[(i)].filename)
5493:12:
5494:13:/* returns 0 if any are not calibrated */
5495:14:int images_calibrated (Desktop * d);
5496:15:
5497:16:/* reports an error message if there are not enough markers in enough views */
5498:17:int check_markers (Desktop * d, int min_views, int max_views, int min_markers, \
5499:18:                   int max_markers, const char *head);
5500:19:
5501:20:/* returns 1 on error */
5502:21:int set_current_view (Desktop * d, int i);
5503:22:
5504:23:void clear_markers (Desktop * d);
5505:24:int new_marker (Desktop * d, Vec x);
5506:25:int remove_last_marker (Desktop * d);
5507:26:void remove_vec (Vec * vec, int i, int n);
5508:27:void remove_a_marker (Desktop * d, int i);
5509:28:void remove_closest_marker (Desktop * d, Vec x);
5510:29:void move_closest_marker (Desktop * d, Vec x);
5511:30:void undrawmarker (View * v);
5512:31:void draw_line_to_pic (Picture * p, double x1, double y1, double x2, \
5513:32:                       double y2, unsigned long c);
5514:33:
5515:34:/* redraw all the markers */
5516:35:void draw_markers (Desktop * d);
5517:36:
5518:37:/* gets the view that the pointer is in. returns -1 if not found */
5519:38:int get_pointer_view (Desktop * d, CEvent * e);
5520:39:
5521:40:/* this sets the 'current' member to the view the pointer is in */
5522:41:void set_current_from_pointer (Desktop * d, CEvent * e);
5523:42:
5524:43:/* real position in zoom image */
5525:44:Vec get_zoom_pointer_pos (Picture * p, CEvent * ev);
5526:45:
5527:46:/* real position in main image */
5528:47:Vec get_main_pointer_pos (Picture * p, CEvent * ev);
5529:48:
5530:49:/* find the real pos of a mouse click on the zoom box */
5531:50:Vec zoom_event_to_image_coord (Picture * p, CEvent * e);
5532:51:
5533:52:/* void imagetopic (Picture *p, int x, int y); */
5534:53:#define imagetopic(p,x,y) { \
5535:54:    (x) += (p)->x0; \
5536:55:    (y) = -(y); \
```

```
5537:56:    (y) += (p)->y0; \
5538:57:}
5539:58:
5540:59:#define pictoimage(p,x,y) { \
5541:60:    (x) -= (p)->x0; \
5542:61:    (y) -= (p)->y0; \
5543:62:    (y) = -(y); \
5544:63:}
5545:64:
5546:65:int num_views_with_markers (Desktop * d);
5547:66:
5548:67:struct marker {
5549:68:    int n;                      /* number of markers */
5550:69:    Vec *v;                     /* markers */
5551:70:    Camera *cam;                /* Camera view of that 2D marker */
5552:71:};
5553:72:
5554:73:typedef struct marker Marker;
5555:74:
5556:75:Marker *get_all_markers (Desktop * d);
5557:76:
5558:77:#endif                          /* _MARKER_H */
5559:1:/*****************************************************************************/
5560:2:/* matrix.c - a matrix manipulation library                                  */
5561:3:/*****************************************************************************/
5562:4:
5563:5:#include <config.h>
5564:6:#include "global.h"
5565:7:#include <math.h>
5566:8:#include <stdlib.h>
5567:9:#include <stdio.h>
5568:10:#include <stdarg.h>
5569:11:#include "matrix.h"
5570:12:#include "string.h"
5571:13:#include "stringtools.h"
5572:14:
5573:15:#include "mad.h"
5574:16:
5575:17:
5576:18:/*
5577:19:   Rules:
5578:20:
5579:21:   (1) Always set matrix '->d' to NULL if you are going to
5580:22:   allow some of the routines to auto initialise the matrix
5581:23:   and you intend passing the matrix to the routine
5582:24:   without it ever having been initialised
5583:25:
5584:26:   eg.  C = Maadd(NULL, A, B);  /+is OK +/
5585:27:   C = Maadd (C, A, B); /+ is fine provided C is either NULL or has been
5586:28:   previously
5587:29:   Maallocmatrix'd or has C->d = NULL +/
5588:30:   Madd (NULL, A, B);      /+ NEVER, 'cos you won't be able to free the
5589:31:
5590:32:   allocated matrix if you don't store the
5591:33:   returned value. +/
5592:34:
5593:35:   (2)
5594:36: */
5595:37:
5596:38:static long total_elements = 0;
5597:39:
5598:40:long Magettotalelements ()
5599:41:{
5600:42:    return total_elements;
5601:43:}
5602:44:
5603:45:int Macheckmatrixbounds (int rows, int columns, int j, int i, \
5604:46:                         const char *file, int line)
5605:47:{
5606:48:    if (j >= rows || i >= columns || j < 0 || i < 0) {
5607:49:        printf (\
5608:50:                "Fatal error:\ntrying to access outside matrix bounds, %s:%d\n", file, line);
5609:51:        abort ();
5610:52:    }
5611:53:    return 0;
5612:54:}
5613:55:
5614:56:void Maerror (const char *e)
5615:57:{
5616:58:    fprintf (stderr, e);
```

```
5617:59:    exit (1);
5618:60:}
5619:61:
5620:62:/*
5621:63:   This sets the size datatype and data of a matrix that has not
5622:64:   had data allocated to it. Low level function.
5623:65: */
5624:66:void Maset (Matrix * A, int rows, int columns, int datatype, void **d)
5625:67:{
5626:68:    A->rows = rows;
5627:69:    A->columns = columns;
5628:70:    A->datatype = 'd';
5629:71:    A->d = d;
5630:72:}
5631:73:
5632:74:void *Mamalloc (size_t size)
5633:75:{
5634:76:    void *p;
5635:77:    if ((p = malloc (size)) == NULL)
5636:78:        Maerror ("Cannot allocate memory for matrix operation.\n");
5637:79:    return p;
5638:80:}
5639:81:
5640:82:/*
5641:83:   Sets all the elements of a matrix to zero.
5642:84: */
5643:85:
5644:86:
5645:87:/*
5646:88:   void *Macalloc (size_t nmemb, size_t size)
5647:89:   {
5648:90:   void *p;
5649:91:   if ((p = calloc (nmemb, size)) == NULL)
5650:92:   Maerror ("Cannot allocate memory for matrix operation.\n");
5651:93:   return p;
5652:94:   }
5653:95: */
5654:96:
5655:97:/*rotates in following order:
5656:98:   phi about x, theta about y, tsi about z
5657:99:   for an aeroplane:
5658:100:   +x = toward nose along fuselage
5659:101:   +y = along right wing
5660:102:   +z = downward
5661:103:   rotation is roll then pitch and then yaw.
5662:104:
5663:105:   for a camera instead of a plane.
5664:106:   +x = East
5665:107:   +y = North
5666:108:   +z = Upward
5667:109:   rotation is roll about optical axis, elevation and then pan.
5668:110:
5669:111:   Hence with no rotation, x points into the image, y horizontal to the left
5670:112:   and z is vertical down (for camera view or forward view
5671:113:   from cockpit window). This an inconvinient definition,
5672:114:   see below for 3dtocreen.
5673:115:
5674:116:   Usually a camera is not rotated, only panned and elevated up and down.
5675:117:   Hence the first rotation angle is close to zero.
5676:118: */
5677:119:
5678:120:
5679:121:
5680:122:Matrix *Magetrotationclassic (Matrix * m, double phi, double theta, double tsi)
5681:123:{
5682:124:    m = Mareinit (m, 3, 3, 'd');
5683:125:
5684:126:    Mard ((*m), 0, 0) = cos (theta) * cos (tsi);
5685:127:    Mard ((*m), 0, 1) = cos (theta) * sin (tsi);
5686:128:    Mard ((*m), 0, 2) = -sin (theta);
5687:129:
5688:130:    Mard ((*m), 1, 0) = sin (phi) * sin (theta) * cos (tsi) - cos (phi) * sin (tsi);
5689:131:    Mard ((*m), 1, 1) = sin (phi) * sin (theta) * sin (tsi) + cos (phi) * cos (tsi);
5690:132:    Mard ((*m), 1, 2) = sin (phi) * cos (theta);
5691:133:
5692:134:    Mard ((*m), 2, 0) = cos (phi) * sin (theta) * cos (tsi) + sin (phi) * sin (tsi);
5693:135:    Mard ((*m), 2, 1) = cos (phi) * sin (theta) * sin (tsi) - sin (phi) * cos (tsi);
5694:136:    Mard ((*m), 2, 2) = cos (phi) * cos (theta);
5695:137:
5696:138:    return m;
```

```
5697:139:}
5698:140:
5699:141:
5700:142:Matrix *Magetrotation (Matrix * m, double phi, double theta, double tsi)
5701:143:{
5702:144:    m = Mareinit (m, 3, 3, 'd');
5703:145:
5704:146:    Mard ((*m), 1, 1) = cos (theta) * cos (tsi);
5705:147:    Mard ((*m), 1, 0) = cos (theta) * sin (tsi);
5706:148:    Mard ((*m), 1, 2) = -sin (theta);
5707:149:
5708:150:    Mard ((*m), 0, 1) = sin (phi) * sin (theta) * cos (tsi) - cos (phi) * sin (tsi);
5709:151:    Mard ((*m), 0, 0) = sin (phi) * sin (theta) * sin (tsi) + cos (phi) * cos (tsi);
5710:152:    Mard ((*m), 0, 2) = sin (phi) * cos (theta);
5711:153:
5712:154:    Mard ((*m), 2, 1) = cos (phi) * sin (theta) * cos (tsi) + sin (phi) * sin (tsi);
5713:155:    Mard ((*m), 2, 0) = cos (phi) * sin (theta) * sin (tsi) - sin (phi) * cos (tsi);
5714:156:    Mard ((*m), 2, 2) = cos (phi) * cos (theta);
5715:157:
5716:158:    return m;
5717:159:}
5718:160:
5719:161:
5720:162:void getrotation (Vec * m0, Vec * m1, Vec * m2, double phi, double theta, double tsi)
5721:163:{
5722:164:    m1->y = cos (theta) * cos (tsi);
5723:165:    m1->x = cos (theta) * sin (tsi);
5724:166:    m1->z = -sin (theta);
5725:167:
5726:168:    m0->y = sin (phi) * sin (theta) * cos (tsi) - cos (phi) * sin (tsi);
5727:169:    m0->x = sin (phi) * sin (theta) * sin (tsi) + cos (phi) * cos (tsi);
5728:170:    m0->z = sin (phi) * cos (theta);
5729:171:
5730:172:    m2->y = cos (phi) * sin (theta) * cos (tsi) + sin (phi) * sin (tsi);
5731:173:    m2->x = cos (phi) * sin (theta) * sin (tsi) - sin (phi) * cos (tsi);
5732:174:    m2->z = cos (phi) * cos (theta);
5733:175:}
5734:176:
5735:177:
5736:178:/*calculates C =  AB
5737:179: */
5738:180:Matrix *Mamultiply (Matrix * C, Matrix * A, Matrix * B)
5739:181:{
5740:182:    int i, j, k;
5741:183:    int n = B->columns, m = A->rows, p;
5742:184:    double sum;
5743:185:
5744:186:    if ((p = A->columns) != B->rows) {
5745:187:        printf ("rows = %d, columns = %d and rows = %d, columns = %d.\n",
5746:188:                A->rows, A->columns, B->rows, B->columns);
5747:189:        Maerror ("trying to multiply matrices of incompatable size.\n");
5748:190:    }
5749:191:    C = Mareinit (C, m, n, 'd');
5750:192:
5751:193:    for (i = 0; i < n; i++)
5752:194:        for (j = 0; j < m; j++) {
5753:195:            sum = 0;
5754:196:            for (k = 0; k < p; k++)
5755:197:                sum += Mard ((*A), j, k) * Mard ((*B), k, i);
5756:198:            Mard ((*C), j, i) = sum;
5757:199:        }
5758:200:    return C;
5759:201:}
5760:202:
5761:203:/*calculates C = A+B
5762:204: */
5763:205:Matrix *Maadd (Matrix * C, Matrix * A, Matrix * B)
5764:206:{
5765:207:    int i, j;
5766:208:    int n = A->columns, m = A->rows;
5767:209:
5768:210:    if (n != B->columns || m != B->rows) {
5769:211:        printf ("rows = %d, columns = %d and rows = %d, columns = %d.\n",
5770:212:                A->rows, A->columns, B->rows, B->columns);
5771:213:        Maerror ("trying to add matrices of incompatable size.\n");
5772:214:    }
5773:215:    C = Mareinit (C, m, n, 'd');
5774:216:
5775:217:    for (i = 0; i < n; i++)
5776:218:        for (j = 0; j < m; j++)
```

```
5777:219:            Mard ((*C), j, i) = Mard ((*A), j, i) + Mard ((*B), j, i);
5778:220:
5779:221:    return C;
5780:222:}
5781:223:
5782:224:/*calculates C = A-B
5783:225: */
5784:226:Matrix *Masubtract (Matrix * C, Matrix * A, Matrix * B)
5785:227:{
5786:228:    int i, j;
5787:229:    int n = A->columns, m = A->rows;
5788:230:
5789:231:    if (n != B->columns || m != B->rows) {
5790:232:        printf ("rows = %d, columns = %d and rows = %d, columns = %d.\n",
5791:233:                A->rows, A->columns, B->rows, B->columns);
5792:234:        Maerror ("trying to subtract matrices of incompatable size.\n");
5793:235:    }
5794:236:    C = Mareinit (C, m, n, 'd');
5795:237:
5796:238:    for (i = 0; i < n; i++)
5797:239:        for (j = 0; j < m; j++)
5798:240:            Mard ((*C), j, i) = Mard ((*A), j, i) - Mard ((*B), j, i);
5799:241:    return C;
5800:242:}
5801:243:
5802:244:
5803:245:/*
5804:246:   For a camera view, with no rotation, this is routine effectively
5805:247:
5806:248:   phi and theta. With no rotation y is into the screen
5807:249:   x is to the right ans z upward.
5808:250:   Increasing theta rotates the view. Increasing phi tilts the view upward
5809:251:   (scrolls downward), and increasing tsi panns to the left (scrolls right).
5810:252:
5811:253:   This calculates the screen postion of a 3D position vector in space.
5812:254:   m is rotation matrix, v is 3D position vector, V is camera position vector,
5813:255:   f is focal length.
5814:256:
5815:257:   returns
5816:258:   x = -f(m_row1(v + V))/(m_row2(v + V))
5817:259:   y = -f(m_row3(v + V))/(m_row2(v + V))
5818:260:   return z = m_row2(v + V);
5819:261:
5820:262:   z(into the screen) and is y before rotation,
5821:263:   x(screen position) is x before rotation, and
5822:264:   y(screen position) is z before rotation.
5823:265:
5824:266:   "As if" the screen was the front windscreen of the aeroplane,
5825:267:   or the camera was pointing north.
5826:268:
5827:269:   Don't forget that y might have to be changed to -y for
5828:270:   a raster display with the origin at top-left.
5829:271: */
5830:272:
5831:273:double Ma3dtoscreen (Matrix * m, Matrix * v, Matrix * V, double f, \
5832:274:                     double *x, double *y)
5833:275:{
5834:276:    Matrix *X = NULL;
5835:277:    Matrix *C = NULL;
5836:278:    Matrix *D = NULL;
5837:279:    double *q;
5838:280:
5839:281:    C = Masubtract (C, v, V);
5840:282:
5841:283:    if (C->rows == 1) {
5842:284:        D = Matranspose (NULL, C);
5843:285:        Mafreematrix (C);
5844:286:        C = D;
5845:287:    }
5846:288:    X = Mamultiply (X, m, C);
5847:289:
5848:290:    q = &(Mard ((*X), 0, 0));
5849:291:
5850:292:    if (q[1] <= 0)
5851:293:        q[1] = -1e-10;
5852:294:
5853:295:    *x = -f * q[0] / q[1];
5854:296:    *y = -f * q[2] / q[1];
5855:297:
5856:298:    Mafreematrix (C);
```

```
5857:299:    Mafreematrix (X);
5858:300:
5859:301:    return q[1];
5860:302:}
5861:303:
5862:304:Vec Mamatrixtovec (Matrix * V)
5863:305:{
5864:306:    double *p = &Mard (*V, 0, 0);
5865:307:    Vec v;
5866:308:    v.x = *p;
5867:309:    v.y = *(p + 1);
5868:310:    v.z = *(p + 2);
5869:311:    return v;
5870:312:}
5871:313:
5872:314:/* returns a column vector */
5873:315:Matrix *Mavectomatrix (Vec v)
5874:316:{
5875:317:    return Madoublestomatrix (1, 3, v.x, v.y, v.z);
5876:318:}
5877:319:
5878:320:/*
5879:321:   inline Vec times3x3(Matrix *m, Vec c)
5880:322:   {
5881:323:   Vec v;
5882:324:   v.x = Mard(*m, 0, 0) * c.x + Mard(*m, 0, 1) * c.y + Mard(*m, 0, 2) * c.z;
5883:325:   v.y = Mard(*m, 1, 0) * c.x + Mard(*m, 1, 1) * c.y + Mard(*m, 1, 2) * c.z;
5884:326:   v.z = Mard(*m, 2, 0) * c.x + Mard(*m, 2, 1) * c.y + Mard(*m, 2, 2) * c.z;
5885:327:   return v;
5886:328:   }
5887:329: */
5888:330:
5889:331:
5890:332:
5891:333:
5892:334:/*returns the sum of all the element of A squared */
5893:335:double Manormal (Matrix * A)
5894:336:{
5895:337:    int i, j;
5896:338:    double sum = 0;
5897:339:
5898:340:    for (i = 0; i < A->columns; i++)
5899:341:        for (j = 0; j < A->rows; j++)
5900:342:            sum += pow (Mard ((*A), j, i), 2);
5901:343:    return sum;
5902:344:}
5903:345:
5904:346:
5905:347:
5906:348:/*performs guass jordan elimination (with partial pivoting)
5907:349:   to diagonalise an augmented square matrix.
5908:350:   matrix must be of size n+1 columns by n rows.
5909:351:   The results are hence left in the last column of
5910:352:   C, and the rest of C is an identity matrix. */
5911:353:
5912:354:Matrix *Madiag (Matrix * C)
5913:355:{
5914:356:    int i, j, k, matrixsize, max;
5915:357:    double temp;
5916:358:
5917:359:    if (C->rows + 1 != C->columns)
5918:360:        Maerror ("Matrix not square augmented in call to Madiag.\n");
5919:361:
5920:362:    matrixsize = C->rows;
5921:363:
5922:364:/*Guass-Jordan elimination with partial pivotting: */
5923:365:
5924:366:    for (i = 0; i < matrixsize; i++) {
5925:367:        max = i;
5926:368:/*Pivot: */
5927:369:        for (j = i + 1; j < matrixsize; j++) {
5928:370:            if (fabs (Mard ((*C), j, i)) >= fabs (Mard ((*C), max, i)))
5929:371:                max = j;
5930:372:        }
5931:373:        for (k = i; k < matrixsize + 1; k++) {
5932:374:            temp = Mard ((*C), i, k);
5933:375:            Mard ((*C), i, k) = Mard ((*C), max, k);
5934:376:            Mard ((*C), max, k) = temp;
5935:377:        }
5936:378:/*Resolve into uppertriangular form: */
```

```
5937:379:        for (j = i + 1; j < matrixsize; j++)
5938:380:            for (k = matrixsize; k >= i; k--) {
5939:381:#ifdef CHECK_MATRIX_SINGULAR
5940:382:                if (Mard ((*C), i, i) == 0)
5941:383:                    return NULL;
5942:384:/*                  Maerror ("Matrix singular in call to Madiag\n"); */
5943:385:#endif
5944:386:                Mard ((*C), j, k) -= Mard ((*C), i, k) * Mard ((*C), j, i) / \
5945:387:                    Mard ((*C), i, i);
5946:388:            }
5947:389:    }
5948:390:
5949:391:/*Perform back substitution: */
5950:392:    for (i = matrixsize - 1; i > 0; i--)
5951:393:        for (j = i - 1; j >= 0; j--) {
5952:394:#ifdef CHECK_MATRIX_SINGULAR
5953:395:            if (Mard ((*C), i, i) == 0)
5954:396:                return NULL;
5955:397:/*              Maerror ("Matrix singular in call to Madiag\n"); */
5956:398:#endif
5957:399:            Mard ((*C), j, matrixsize) -= Mard ((*C), j, i) / \
5958:400:                Mard ((*C), i, i) * Mard ((*C), i, matrixsize);
5959:401:            Mard ((*C), j, i) = 0;
5960:402:        }
5961:403:
5962:404:/*Normalise diagonal to 1: */
5963:405:    for (i = 0; i < matrixsize; i++) {
5964:406:#ifdef CHECK_MATRIX_SINGULAR
5965:407:        if (Mard ((*C), i, i) == 0)
5966:408:            return NULL;
5967:409:/*          Maerror ("Matrix singular in call to Madiag\n"); */
5968:410:#endif
5969:411:        Mard ((*C), i, matrixsize) /= Mard ((*C), i, i);
5970:412:        Mard ((*C), i, i) = 1;
5971:413:    }
5972:414:
5973:415:    return C;
5974:416:}
5975:417:
5976:418:/*
5977:419:   Re-initialises a matrix if differs in size or type from the given
5978:420:   values. If the given matrix is a NULL pointer than reinit
5979:421:   allocates a matrix of the required size and returns a pointer
5980:422:   to it
5981:423: */
5982:424:Matrix *Mareinit (Matrix * A, int rows, int columns, int datatype)
5983:425:{
5984:426:    if (!A)
5985:427:        return Maallocmatrix (rows, columns, 'd');
5986:428:    if (A->rows != rows || A->columns != columns ||
5987:429:        A->datatype != datatype) {
5988:430:        Madestroymatrix (A);
5989:431:        Mainitmatrix (A, rows, columns, 'd');
5990:432:    }
5991:433:    return A;
5992:434:}
5993:435:
5994:436:
5995:437:/*calculates C = A'T */
5996:438:Matrix *Matranspose (Matrix * C, Matrix * A)
5997:439:{
5998:440:    int i, j;
5999:441:    int n = A->rows, m = A->columns;
6000:442:
6001:443:    C = Mareinit (C, m, n, 'd');
6002:444:
6003:445:    for (i = 0; i < n; i++)
6004:446:        for (j = 0; j < m; j++)
6005:447:            Mard ((*C), j, i) = Mard ((*A), i, j);
6006:448:
6007:449:    return C;
6008:450:}
6009:451:
6010:452:
6011:453:/* calculates C = [A:Y] */
6012:454:Matrix *Maaugment (Matrix * C, Matrix * A, Matrix * Y)
6013:455:{
6014:456:    int i, j;
6015:457:    int n, m, o;
6016:458:
```

```
6017:459:    C = Mareinit (C, m = A->rows, n = (o = A->columns) + Y->columns, 'd');
6018:460:
6019:461:    for (i = 0; i < o; i++)
6020:462:        for (j = 0; j < m; j++)
6021:463:            Mard ((*C), j, i) = Mard ((*A), j, i);
6022:464:    for (i = o; i < n; i++)
6023:465:        for (j = 0; j < m; j++)
6024:466:            Mard ((*C), j, i) = Mard ((*Y), j, i - o);
6025:467:
6026:468:    return C;
6027:469:}
6028:470:
6029:471:/*
6030:472:   Extracts a matrix C numrows by numcolumns from a larger matrix A
6031:473:   starting at the position row, column of A
6032:474: */
6033:475:Matrix *Masubmatrix (Matrix * C, Matrix * A, int row, int column, \
6034:476:                     int numrows, int numcolumns)
6035:477:{
6036:478:    int i, j;
6037:479:    int n, m;
6038:480:
6039:481:    C = Mareinit (C, m = numrows, n = numcolumns, 'd');
6040:482:
6041:483:    for (i = 0; i < n; i++)
6042:484:        for (j = 0; j < m; j++)
6043:485:            Mard ((*C), j, i) = Mard ((*A), j + row, i + column);
6044:486:
6045:487:    return C;
6046:488:}
6047:489:
6048:490:
6049:491:
6050:492:/*
6051:493:   The following finds the least squares solution to X where
6052:494:
6053:495:   AX = Y
6054:496:
6055:497:   where A has more rows than columns. I.e. it solves
6056:498:   an overconstrained linear system by least squares.
6057:499:
6058:500:   if A is square Madiag is called. This consumes more memory than a direct call to
6059:501:   Madiag as an augmented matrix has to be created first.
6060:502:
6061:503:   This does not change A or Y.
6062:504:
6063:505:   returns NULL if A is singular
6064:506: */
6065:507:
6066:508:Matrix *Maminimize (Matrix * X, Matrix * A, Matrix * Y)
6067:509:{
6068:510:    int m = A->rows, n = A->columns;
6069:511:    Matrix *AT = NULL, *C = NULL, *ATY = NULL;
6070:512:
6071:513:    if (m < n)
6072:514:/*could do a minimum entropy thing here */
6073:515:        Maerror (\
6074:516:                 "Number of rows less than or equal to number of columns in call to Maminimise\n");
6075:517:
6076:518:    if (m == n) {
6077:519:/*      AT = Maallocmatrix(n, n + 1, 'd'); */
6078:520:        AT = Maaugment (NULL, A, Y);
6079:521:    } else {
6080:522:/*      AT = Maallocmatrix(n, m, 'd'); */
6081:523:/*      C = Maallocmatrix(n, n, 'd'); */
6082:524:/*      ATY = Maallocmatrix(n, 1, 'd'); */
6083:525:
6084:526:        C = Mamultiply (NULL, AT = Matranspose (NULL, A), A);
6085:527:        ATY = Mamultiply (NULL, AT, Y);
6086:528:
6087:529:/*now we'll use AT to hold the augmented matrix. */
6088:530:        Maaugment (AT, C, ATY);
6089:531:        Mafreematrix (ATY);
6090:532:        Mafreematrix (C);
6091:533:/*This is almost to simple. */
6092:534:    }
6093:535:
6094:536:/*AT now holds a square augmented matrix */
6095:537:    if (Madiag (AT) == NULL) {
6096:538:        return NULL;
```

```
6097:539:    }
6098:540:    X = Masubmatrix (X, AT, 0, n, n, 1);
6099:541:    Mafreematrix (AT);
6100:542:    return X;
6101:543:}
6102:544:
6103:545:
6104:546:void Maprint (Matrix * Y)
6105:547:{
6106:548:    int i, j;
6107:549:    for (j = 0; j < Y->rows; j++) {
6108:550:        for (i = 0; i < Y->columns; i++)
6109:551:            printf ("% 9.8g ", Mard ((*Y), j, i));
6110:552:        printf ("\n");
6111:553:    }
6112:554:    printf ("\n");
6113:555:}
6114:556:
6115:557:
6116:558:void Maprintmatrix (Matrix * Y)
6117:559:{
6118:560:    int i, j;
6119:561:    for (j = 0; j < Y->rows; j++) {
6120:562:        for (i = 0; i < Y->columns; i++)
6121:563:            printf ("% 9.8g ", Mard ((*Y), j, i));
6122:564:        printf ("\n");
6123:565:    }
6124:566:    printf ("\n");
6125:567:}
6126:568:
6127:569:
6128:570:/*find sqrt of the sum squared diference of A and B
6129:571:   using the smaller of A and B as the extents to which it
6130:572:   compares */
6131:573:double Madistance (Matrix * A, Matrix * B)
6132:574:{
6133:575:    int i, j, n, m;
6134:576:    double d = 0;
6135:577:
6136:578:    n = min (A->rows, B->rows);
6137:579:    m = min (A->columns, B->columns);
6138:580:
6139:581:    for (j = 0; j < n; j++)
6140:582:        for (i = 0; i < m; i++)
6141:583:            d += pow (Mard ((*A), j, i) - Mard ((*B), j, i), 2);
6142:584:
6143:585:    return sqrt (d);
6144:586:}
6145:587:
6146:588:Matrix *Macross (Matrix * a, Matrix * b)
6147:589:{
6148:590:    Matrix *r = Maallocmatrix (1, 3, 'd');
6149:591:    Mard (*r, 0, 0) = Mard (*r, 0, 1) * Mard (*r, 0, 2) - \
6150:592:        Mard (*r, 0, 2) * Mard (*r, 0, 1);
6151:593:    Mard (*r, 0, 1) = Mard (*r, 0, 2) * Mard (*r, 0, 0) - \
6152:594:        Mard (*r, 0, 0) * Mard (*r, 0, 2);
6153:595:    Mard (*r, 0, 2) = Mard (*r, 0, 0) * Mard (*r, 0, 1) - \
6154:596:        Mard (*r, 0, 1) * Mard (*r, 0, 0);
6155:597:    return r;
6156:598:}
6157:599:
6158:600:/*returns the sum of the product of every corresponding element */
6159:601:double Madot (Matrix * A, Matrix * B)
6160:602:{
6161:603:    int i, j, n, m;
6162:604:    double d = 0;
6163:605:
6164:606:    n = min (A->rows, B->rows);
6165:607:    m = min (A->columns, B->columns);
6166:608:
6167:609:    for (j = 0; j < n; j++)
6168:610:        for (i = 0; i < m; i++)
6169:611:            d += Mard ((*A), j, i) * Mard ((*B), j, i);
6170:612:
6171:613:    return d;
6172:614:}
6173:615:
6174:616:
6175:617:Matrix *Mascale (Matrix * A, double s)
6176:618:{
```

```
6177:619:    Maallelements (*A, double, *=s);
6178:620:    return A;
6179:621:}
6180:622:
6181:623:/*
6182:624:   Matrix *Mascalemany (Matrix * A, double *s,...)
6183:625:   {
6184:626:   double S, *d;
6185:627:   va_list ap;
6186:628:
6187:629:   if (!s)
6188:630:   return 0;
6189:631:
6190:632:   S = *s;
6191:633:
6192:634:   va_start (ap, s);
6193:635:
6194:636:   while ((d = va_arg (ap, double *))!=NULL)
6195:637:   S *= *d;
6196:638:
6197:639:   Maallelements(*A, double, *=S);
6198:640:
6199:641:   va_end (ap);
6200:642:
6201:643:   return A;
6202:644:   }
6203:645: */
6204:646:
6205:647:Matrix *Maaddscalar (Matrix * A, double s)
6206:648:{
6207:649:    Maallelements (*A, double, +=s);
6208:650:    return A;
6209:651:}
6210:652:
6211:653:/* args terminated by zero */
6212:654:Matrix *Maaddmanyscalar (Matrix * A, double *s,...)
6213:655:{
6214:656:    double S, *d;
6215:657:    va_list ap;
6216:658:
6217:659:    if (!s)
6218:660:        return 0;
6219:661:
6220:662:    S = *s;
6221:663:
6222:664:    va_start (ap, s);
6223:665:
6224:666:    while ((d = va_arg (ap, double *)) != NULL)
6225:667:         S += *d;
6226:668:
6227:669:    Maallelements (*A, double, +=S);
6228:670:
6229:671:/*    for (j = 0; j < A->rows; j++)
6230:672:   for (i = 0; i < A->columns; i++)
6231:673:   Mard ((*A), j, i) += S;
6232:674: */
6233:675:
6234:676:    va_end (ap);
6235:677:
6236:678:    return A;
6237:679:}
6238:680:
6239:681:
6240:682:/*inserts smaller matrix A into larger matrix C at the given position */
6241:683:Matrix *Mainsert (Matrix * C, Matrix * A, int row, int column)
6242:684:{
6243:685:    int i, j;
6244:686:    for (j = 0; j < A->rows; j++)
6245:687:        for (i = 0; i < A->columns; i++)
6246:688:            Mard ((*C), j + row, i + column) = Mard ((*A), j, i);
6247:689:    return C;
6248:690:}
6249:691:
6250:692:void Masetupdata (Matrix * A, int rows, int columns, int datatype, int type_size)
6251:693:{
6252:694:    int i;
6253:695:
6254:696:    Maset (A, rows, columns, datatype, Mamalloc (rows * sizeof (void *)));
6255:697:
6256:698:    *(A->d) = Mamalloc (rows * columns * type_size);
```

```
6257:699:    total_elements += rows * columns;
6258:700:    if (rows > 1)
6259:701:        for (i = 1; i < rows; i++)
6260:702:            *(A->d + i) = (byte *) * (A->d) + i * columns * type_size;
6261:703:}
6262:704:
6263:705:/*
6264:706:   sets up the row pointers A.d[i]. Must be called only
6265:707:   after A.d has been alloc´ated an array of pointers (obviously).
6266:708:   p must point to the first byte of the data.
6267:709: */
6268:710:
6269:711:void Mainitrows (Matrix * A, int rows, int columns, int sizeoftype, void *p)
6270:712:{
6271:713:    int i = 0;
6272:714:    total_elements += rows * columns;
6273:715:    for (; i < rows; i++)
6274:716:        *(A->d + i) = (byte *) p + i * columns * sizeoftype;
6275:717:}
6276:718:
6277:719:int Masizeof (int datatype)
6278:720:{
6279:721:    switch (datatype) {
6280:722:    case ´d´:
6281:723:        return (sizeof (double));
6282:724:        break;
6283:725:    case ´f´:
6284:726:        return (sizeof (float));
6285:727:        break;
6286:728:    case ´l´:
6287:729:        return (sizeof (long));
6288:730:        break;
6289:731:    case ´c´:
6290:732:        return (sizeof (char));
6291:733:        break;
6292:734:    case ´s´:
6293:735:        return (sizeof (short));
6294:736:        break;
6295:737:    case ´i´:
6296:738:        return (sizeof (short));
6297:739:        break;
6298:740:    default:
6299:741:        Maerror ("This matrix type is invalid or not supported.");
6300:742:        return 0;
6301:743:    }
6302:744:}
6303:745:
6304:746:void Maclearmatrix (Matrix * A)
6305:747:{
6306:748:    memset (&(Mard ((*A), 0, 0)), 0, Masizeof (A->datatype) * A->columns * A->rows);
6307:749:}
6308:750:
6309:751:
6310:752:/*
6311:753:   Initializes an existing matrix structure with the passed values,
6312:754:   allocates memory for its data, and initialises that memory to zero
6313:755:   Use for initialising a matrix declared as a constant.
6314:756: */
6315:757:Matrix *Mainitmatrix (Matrix * A, int rows, int columns, int datatype)
6316:758:{
6317:759:    int s = Masizeof (datatype);
6318:760:    Maset (A, rows, columns, datatype, Mamalloc (rows * sizeof (void *)));
6319:761:    Mainitrows (A, rows, columns, s, Mamalloc (rows * columns * s));
6320:762:    Maclearmatrix (A);
6321:763:    return A;
6322:764:}
6323:765:
6324:766:
6325:767:/*
6326:768:   Allocates memory for a matrix structure, intialises it with the
6327:769:   passed values, allocates memory for matrix data, sets matrix data to zero.
6328:770:   Use for initialising a matrix declared as a pointer.
6329:771: */
6330:772:Matrix *Maallocmatrix (int rows, int columns, int datatype)
6331:773:{
6332:774:    Matrix *A = Mamalloc (sizeof (Matrix));
6333:775:    return Mainitmatrix (A, rows, columns, datatype);
6334:776:}
6335:777:
6336:778:/*
```

```
6337:779:   Here you can make a matrix from a list of data
6338:780:   eg for a 2x3:
6339:781:   R = Maallocdata (&x, &y, &z, NULL, &a, &b, &c, NULL, NULL);
6340:782:
6341:783:   each row ends with a NULL and the final rows ends with two NULL's.
6342:784:
6343:785: */
6344:786:
6345:787:/* do not terminate by zero */
6346:788:/* used like this:
6347:789:   Matrix *Madoublestomatrix (int rows, int columns, double first,...) */
6348:790:Matrix *Madoublestomatrix (int rows, double first,...)
6349:791:{
6350:792:    va_list ap;
6351:793:
6352:794:    Matrix *R = Maallocmatrix (rows, (int) first, 'd');
6353:795:
6354:796:    va_start (ap, first);
6355:797:    Maallelements (*R, double, = va_arg (ap, double));
6356:798:    va_end (ap);
6357:799:
6358:800:    return R;
6359:801:}
6360:802:
6361:803:
6362:804:/*
6363:805:   This COPIES an existing TWO DIMENSIONAL array p[][] into
6364:806:   a matrix. p must be larger than or equal to 'rows' and 'columns'.
6365:807:   If the data is a large contigous block, then rather use Maallocuser
6366:808:   or Mainituser, to avoid duplicating the data. p may be free'd after
6367:809:   calling these functions. The returned matrix must be Mafreenatrix'd.
6368:810: */
6369:811:
6370:812:Matrix *Mainitarray (Matrix * A, int rows, int columns, int datatype, void **p)
6371:813:{
6372:814:    int j = 0;
6373:815:    int s = Masizeof (datatype);
6374:816:    Maset (A, rows, columns, datatype, Mamalloc (rows * sizeof (void *)));
6375:817:    Mainitrows (A, rows, columns, s, Mamalloc (rows * columns * s));
6376:818:
6377:819:    for (; j < rows; j++)
6378:820:        memcpy (A->d[j], p[j], columns * s);
6379:821:
6380:822:    return A;
6381:823:}
6382:824:
6383:825:Matrix *Maallocarray (int rows, int columns, int d, void **p)
6384:826:{
6385:827:    Matrix *R = Mamalloc (sizeof (Matrix));
6386:828:    Mainitarray (R, rows, columns, d, p);
6387:829:    return R;
6388:830:}
6389:831:
6390:832:/*
6391:833:   This initialises a matrix to the values of a ONE DIMENSIONAL
6392:834:   array q[]. q must have sufficient space allocated to it
6393:835:   or be of sufficient size.
6394:836:   The matrix may have width or height greater than one, but
6395:837:   then the data will scan the matrix from left to right and then
6396:838:   from top to bottom. q may be free'd after the matrix
6397:839:   has been Madestroyuser'd or Mafreeuser'd. If q is a
6398:840:   constant, the matrix must be destoyed only with Madestroyuser()
6399:841:   or Mafreeuser(). In this case be weary of commands that may like to
6400:842:   change the size of the matrix by re-malloc'ating it.
6401:843:   If q is allocated then the resulting matrix will be no different
6402:844:   (at least in this version) to a matrix created by any of the
6403:845:   other routines.
6404:846:
6405:847:   This is useful for turning a structure into a matrix.
6406:848:   eg.
6407:849:
6408:850:   typedef struct {
6409:851:   double x,
6410:852:   double y,
6411:853:   double z,
6412:854:   double a,
6413:855:   double b,
6414:856:   double c,
6415:857:   } Vector
6416:858:
```

```
6417:859:    .
6418:860:    .
6419:861:    .
6420:862:
6421:863:    Vector v;
6422:864:
6423:865:    Matrix *V = Maallocuser(2,3,'d',&(v.x));
6424:866:
6425:867:    Mard((*V),1,2) = THE_VALUE_OF_B;
6426:868:    .
6427:869:    .
6428:870:    .
6429:871:
6430:872:    Mafreeuser(V);
6431:873:
6432:874: */
6433:875:
6434:876:Matrix *Mainituser (Matrix * R, int rows, int columns, int d, void *q)
6435:877:{
6436:878:    Maset (R, rows, columns, d, Mamalloc (rows * sizeof (void *)));
6437:879:    Mainitrows (R, rows, columns, Masizeof (d), q);
6438:880:    return R;
6439:881:}
6440:882:
6441:883:Matrix *Maallocuser (int rows, int columns, int d, void *q)
6442:884:{
6443:885:    Matrix *R = Mamalloc (sizeof (Matrix));
6444:886:    Mainituser (R, rows, columns, 'd', q);
6445:887:    return R;
6446:888:}
6447:889:
6448:890:
6449:891:/*frees the data in A. Initialise the matrix structure and
6450:892:   frees the data, but does not free the matrix structure itself. */
6451:893:/*use for freeing a matrix declared as a constant */
6452:894:void Madestroymatrix (Matrix * A)
6453:895:{
6454:896:    if (A) {
6455:897:        if (*(A->d))
6456:898:            free (*(A->d));
6457:899:        if (A->d)
6458:900:            free (A->d);
6459:901:
6460:902:        total_elements -= A->rows * A->columns;
6461:903:        Maset (A, 0, 0, 0, NULL);
6462:904:    } else
6463:905:        printf ("Warning: trying to destroy a matrix that is a NULL pointer.\n");
6464:906:}
6465:907:
6466:908:
6467:909:/* free's the entire matrix, structure and data */
6468:910:/* for freeing matrices that are declared as pointers */
6469:911:void Mafreematrix (Matrix * A)
6470:912:{
6471:913:    if (A) {
6472:914:        Madestroymatrix (A);
6473:915:        free (A);
6474:916:    }
6475:917:}
6476:918:
6477:919:void Mafreeuser (Matrix * A)
6478:920:{
6479:921:    *(A->d) = NULL;
6480:922:    Mafreematrix (A);
6481:923:}
6482:924:
6483:925:void Madestroyuser (Matrix * A)
6484:926:{
6485:927:    *(A->d) = NULL;
6486:928:    Madestroymatrix (A);
6487:929:}
6488:930:
6489:931:/*
6490:932:   example:
6491:933:
6492:934:
6493:935:   main()
6494:936:   {
6495:937:
6496:938:   Matrix *A;
```

```
6497:939:   Matrix B;
6498:940:
6499:941:   Mainitmatrix(B, 3, 4, 'd');
6500:942:   Maallocmatrix(A, 3, 4, 'd'); /+ type double +/
6501:943:
6502:944:   .
6503:945:   .
6504:946:   .
6505:947:
6506:948:
6507:949:   Madestroymatrix(B);
6508:950:   Mafreematrix(A);
6509:951:
6510:952:   }
6511:953:
6512:954: */
6513:955:
6514:956:
6515:957:
6516:958:
6517:959:/*
6518:960:   Converts a table of ascii doubles to a matrix.
6519:961:   The table can be delimited by almost any character that wouldn't
6520:962:   be in a number. Rows must be delimited by newlines. The last
6521:963:   line must not end with a newline, or an extra row of zeros will
6522:964:   be added.
6523:965:   The number of doubles on the first line dictate the number of
6524:966:   columns. The remainder of the matrix will be padded with zeros
6525:967:   where there are not enough doubles on a line. The number of
6526:968:   lines dictates the number of rows. If there are a greater
6527:969:   number of columns than suggested by the first line, then
6528:970:   these will be ignored.
6529:971:
6530:972:   Returns matrix pointer which must be Mafree'd --- see Mafree().
6531:973:   Returns NULL on error or empty file --- although it will attempt
6532:974:   to convert almost anything.
6533:975: */
6534:976:
6535:977:Matrix *Maasciitomatrix (const char *text)
6536:978:{
6537:979:    int rows = strcountlines (text, 0, strlen (text), 32000) + 1, columns = 0;
6538:980:    Matrix *X;
6539:981:
6540:982:    char *endptr = (char *) text, *nptr;
6541:983:
6542:984:    int i = 0, j = 0;
6543:985:
6544:986:    do {
6545:987:        columns++;
6546:988:        nptr = endptr + strcspn (endptr, "+-1234567890.\n");
6547:989:        if (!(*nptr) || *nptr == '\n')
6548:990:            break;
6549:991:        strtod (nptr, &endptr);
6550:992:    } while (nptr != endptr);
6551:993:
6552:994:    columns--;
6553:995:
6554:996:    if (!columns)
6555:997:        return NULL;
6556:998:
6557:999:    nptr = (char *) text;
6558:1000:
6559:1001:    X = Maallocmatrix (rows, columns, 'd');
6560:1002:    Maclearmatrix (X);
6561:1003:
6562:1004:    while (*nptr && i < rows) {
6563:1005:        j = 0;
6564:1006:        endptr = nptr;
6565:1007:        do {
6566:1008:            j++;
6567:1009:            nptr = endptr + strcspn (endptr, "+-1234567890.\n");
6568:1010:            if (!(*nptr) || *nptr == '\n')
6569:1011:                break;
6570:1012:            Mard ((*X), i, j - 1) = strtod (nptr, &endptr);
6571:1013:        } while (nptr != endptr && j <= columns);
6572:1014:        nptr += strmovelines (nptr, 0, 1, 32000);
6573:1015:        i++;
6574:1016:    }
6575:1017:
6576:1018:    return X;
```

```
6577:1019:}
6578:1:#ifndef MATRIX_H
6579:2:#define MATRIX_H
6580:3:
6581:4:#include "../config.h"
6582:5:#include <stdlib.h>
6583:6:#include <math.h>
6584:7:#include <quickmath.h>
6585:8:
6586:9:/*
6587:10:   Uncomment this line to let any matrix reference check whether it is within
6588:11:   bounds of the matrix. This will easily show up code that references
6589:12:   outside the dimensions of the matrix. It gives the responce:
6590:13:   Fatal error: trying to access outside matrix bounds, <file>:<line>
6591:14: */
6592:15:/* #define CHECK_MATRIX_BOUNDS */
6593:16:
6594:17:/*uncomment to let the matrix diagonalisation pick up zero divides */
6595:18:#define CHECK_MATRIX_SINGULAR 1
6596:19:
6597:20:/*all matrix operations start with Ma */
6598:21:
6599:22:
6600:23:typedef struct {
6601:24:    int rows;                   /* size */
6602:25:    int columns;                /* of matrix */
6603:26:    int datatype;               /* is one of 'd', 'f', 'l', 'c', 's', or 'i' */
6604:27:    /* see below for definitions */
6605:28:    void **d;
6606:29:} Matrix;
6607:30:
6608:31:int Macheckmatrixbounds (int rows, int columns, int j, int i, \
6609:32:                          const char *file, int line);
6610:33:
6611:34:
6612:35:#ifdef CHECK_MATRIX_BOUNDS
6613:36:
6614:37:/* Macheckmatrixbounds always returns zero or aborts if out of bounds */
6615:38:
6616:39:/* eg: 'Ma'trix 'r'eturn 'd'ouble */
6617:40:#define Mard(m,j,i) (*((double *) *((m).d + j) + i \
6618:41:        + Macheckmatrixbounds((m).rows, (m).columns, j, i, __FILE__, __LINE__)))
6619:42:#define Marf(m,j,i) (*((float *) *((m).d + j) + i \
6620:43:        + Macheckmatrixbounds((m).rows, (m).columns, j, i, __FILE__, __LINE__)))
6621:44:#define Marl(m,j,i) (*((long *) *((m).d + j) + i \
6622:45:        + Macheckmatrixbounds((m).rows, (m).columns, j, i, __FILE__, __LINE__)))
6623:46:#define Marc(m,j,i) (*((unsigned char *) *((m).d + j) + i \
6624:47:        + Macheckmatrixbounds((m).rows, (m).columns, j, i, __FILE__, __LINE__)))
6625:48:/* 's'hort unsigned */
6626:49:#define Mars(m,j,i) (*((unsigned short *) *((m).d + j) + i \
6627:50:        + Macheckmatrixbounds((m).rows, (m).columns, j, i, __FILE__, __LINE__)))
6628:51:/* short 'i'nteger: */
6629:52:#define Mari(m,j,i) (*((signed short *) *((m).d + j) + i \
6630:53:        + Macheckmatrixbounds((m).rows, (m).columns, j, i, __FILE__, __LINE__)))
6631:54:
6632:55:#else
6633:56:
6634:57:#define Mard(m,j,i) (*((double *) *((m).d + j) + i))
6635:58:#define Marf(m,j,i) (*((float *) *((m).d + j) + i))
6636:59:#define Marl(m,j,i) (*((long *) *((m).d + j) + i))
6637:60:#define Marc(m,j,i) (*((unsigned char *) *((m).d + j) + i))
6638:61:/* 's'hort */
6639:62:#define Mars(m,j,i) (*((unsigned short *) *((m).d + j) + i))
6640:63:/* 'i'nteger: */
6641:64:#define Mari(m,j,i) (*((signed short *) *((m).d + j) + i))
6642:65:
6643:66:#endif
6644:67:
6645:68:/*
6646:69:   do operation 'o' to all elements in a matrix.
6647:70:   This is as fast as its going to get;
6648:71: */
6649:72:#define Maallelements(m, t, o) \
6650:73:                { \
6651:74:                    t *p_to_element = (t *) (*(m).d); \
6652:75:                    long num_elements = (long) (m).rows * (m).columns; \
6653:76:                    do { \
6654:77:                        *(p_to_element++) o; \
6655:78:                    } while (--num_elements); \
6656:79:                }
```

```
6657:80:
6658:81:
6659:82:Matrix *Magetrotation (Matrix * m, double phi, double theta, double tsi);
6660:83:
6661:84:Matrix *Mamultiply (Matrix * C, Matrix * A, Matrix * B);
6662:85:
6663:86:Matrix *Maadd (Matrix * C, Matrix * A, Matrix * B);
6664:87:
6665:88:Matrix *Masubtract (Matrix * C, Matrix * A, Matrix * B);
6666:89:
6667:90:double Ma3dtoscreen (Matrix * m, Matrix * v, Matrix * V, double f, \
6668:91:                     double *x, double *y);
6669:92:
6670:93:double Manormal (Matrix * A);
6671:94:
6672:95:Matrix *Madiag (Matrix * C);
6673:96:
6674:97:Matrix *Matranspose (Matrix * C, Matrix * A);
6675:98:
6676:99:Matrix *Maaugment (Matrix * C, Matrix * A, Matrix * Y);
6677:100:
6678:101:Matrix *Masubmatrix (Matrix * C, Matrix * A, int row, int column, \
6679:102:                      int numrows, int numcolumns);
6680:103:
6681:104:Matrix *Maminimize (Matrix * X, Matrix * A, Matrix * Y);
6682:105:
6683:106:void Maprintmatrix (Matrix * Y);
6684:107:
6685:108:void Maprint (Matrix * Y);
6686:109:
6687:110:double Madistance (Matrix * A, Matrix * B);
6688:111:
6689:112:Matrix *Macross (Matrix * a, Matrix * b);
6690:113:
6691:114:double Madot (Matrix * A, Matrix * B);
6692:115:
6693:116:Matrix *Mascale (Matrix * A, double s);
6694:117:
6695:118:Matrix *Maaddscalar (Matrix * A, double s);
6696:119:
6697:120:Matrix *Mainsert (Matrix * C, Matrix * A, int row, int column);
6698:121:
6699:122:Matrix *Mainitmatrix (Matrix * A, int rows, int columns, int datatype);
6700:123:
6701:124:Matrix *Maallocmatrix (int rows, int columns, int datatype);
6702:125:
6703:126:void Madestroymatrix (Matrix * A);
6704:127:
6705:128:void Mafreematrix (Matrix * A);
6706:129:
6707:130:Matrix *Maasciitomatrix (const char *text);
6708:131:
6709:132:void *Mamalloc (size_t size);
6710:133:
6711:134:
6712:135:Matrix *Madoublestomatrix (int rows, double first,...);
6713:136:
6714:137:Matrix *Mainitarray (Matrix * R, int rows, int columns, int d, void **p);
6715:138:
6716:139:Matrix *Maallocarray (int rows, int columns, int d, void **p);
6717:140:
6718:141:Matrix *Mainituser (Matrix * R, int rows, int columns, int d, void *q);
6719:142:
6720:143:Matrix *Maallocuser (int rows, int columns, int d, void *q);
6721:144:
6722:145:void Mafreeuser (Matrix * A);
6723:146:
6724:147:void Madestroyuser (Matrix * A);
6725:148:
6726:149:void Maerror (const char *e);
6727:150:
6728:151:Matrix *Mareinit (Matrix * A, int rows, int columns, int data_type);
6729:152:
6730:153:long Magettotalelements ();
6731:154:
6732:155:Matrix *Maaddmanyscalar (Matrix * A, double *s,...);
6733:156:
6734:157:void Maclearmatrix (Matrix * A);
6735:158:
6736:159:Vec Mamatrixtovec (Matrix * V);
```

```
6737:160:
6738:161:Matrix *Mavectomatrix (Vec v);
6739:162:
6740:163:void getrotation (Vec * m0, Vec * m1, Vec * m2, double phi, double theta, double tsi);
6741:164:
6742:165:#endif
6743:1:
6744:2:/*****************************************************************************************/
6745:3:/* output.c - this outputs an object (line, circle, etc) to the editor                   */
6746:4:/*****************************************************************************************/
6747:5:
6748:6:#include "display.h"
6749:7:#include "imagefit.h"
6750:8:#include "app_glob.c"
6751:9:#include "hugeimage.h"
6752:10:#include "widget3d.h"
6753:11:#include "marker.h"
6754:12:#include "displaycam.h"
6755:13:#include "picsetup.h"
6756:14:#include "dialog.h"
6757:15:#include "stringtools.h"
6758:16:#include "callback.h"
6759:17:#include "matrix.h"
6760:18:#include "edit.h"
6761:19:
6762:20:/*


 */
6763:21:
6764:22:int edit_print_string (WEdit * e, const char *s);
6765:23:
6766:24:void oprint (const char *fmt,...)
6767:25:{
6768:26:    char *s;
6769:27:    va_list ap;
6770:28:    va_start (ap, fmt);
6771:29:    s = vsprintf_alloc (fmt, ap);
6772:30:    edit_print_string (Cwidget ("editor")->editor, s);
6773:31:    free (s);
6774:32:    va_end (ap);
6775:33:}
6776:34:
6777:35:
6778:36:
6779:37:void print_surface (Surface * s)
6780:38:{
6781:39:    int i;
6782:40:    oprint ("\nsurface %d %d ", s->w, s->h);
6783:41:    for (i = 0; i < s->w * s->h; i++)
6784:42:        oprint ("%f %f %f\n", (float) s->p[i].x, (float) s->p[i].y, \
6785:43:                (float) s->p[i].z);
6786:44:    oprint ("\n");
6787:45:}
6788:46:
6789:47:void print_circle (Circle * c)
6790:48:{
6791:49:    oprint ("\ncircle %f %f %f %f\n", c->p.x, c->p.y, c->p.z, c->r);
6792:50:}
6793:51:
6794:52:void print_cylinder (Cylinder * c)
6795:53:{
6796:54:    Vec join;
6797:55:    join = minus (c->l.p2, c->l.p1);
6798:56:    oprint ("\ncappedcylinder %f %f %f %f %f %f %f\n", \
6799:57:          join.x, join.y, join.z, c->l.p1.x, c->l.p1.y, c->l.p1.z, c->r);
6800:58:}
6801:59:
6802:60:#define LINE_CYL_RADIUS 100
6803:61:
6804:62:void print_line (LineSegment * l)
6805:63:{
6806:64:    Vec join;
6807:65:    join = minus (l->p2, l->p1);
6808:66:    oprint ("\ncappedcylinder %f %f %f %f %f %f %f\n", \
6809:67:     join.x, join.y, join.z, l->p1.x, l->p1.y, l->p1.z, LINE_CYL_RADIUS);
6810:68:}
6811:69:
6812:70:void print_all_object_data (Object * o)
6813:71:{
```

```
6814:72:    switch (o->type) {
6815:73:    case NO_TYPE:
6816:74:        oprint ("# NO_TYPE\n");
6817:75:        break;
6818:76:    case VECTOR:
6819:77:        oprint ("# p = (%f, %f, %f), direction = (%f, %f, %f)\n",
6820:78:                (float) o->vector.p.x, (float) o->vector.p.y, (float) o->vector.p.z,
6821:79:                (float) o->vector.u.x, (float) o->vector.u.y, (float) o->vector.u.z);
6822:80:        break;
6823:81:    case POINT:
6824:82:        oprint ("# p = (%f, %f, %f)\n",
6825:83:        (float) o->point.p.x, (float) o->point.p.y, (float) o->point.p.z);
6826:84:        break;
6827:85:    case LINE_SEGMENT:
6828:86:        oprint ("# p1 = (%f, %f, %f), ",
6829:87:        (float) o->line.p1.x, (float) o->line.p1.y, (float) o->line.p1.z);
6830:88:        oprint (" p2 = (%f, %f, %f) \n",
6831:89:        (float) o->line.p2.x, (float) o->line.p2.y, (float) o->line.p2.z);
6832:90:        oprint ("# l = %f ", (float) o->line.l);
6833:91:        oprint (" direction = (%f, %f, %f) \n",
6834:92:          (float) o->line.u.x, (float) o->line.u.y, (float) o->line.u.z);
6835:93:        break;
6836:94:    case CYLINDER:
6837:95:        oprint ("# p1 = (%f, %f, %f), ",
6838:96:                (float) o->cylinder.l.p1.x, (float) o->cylinder.l.p1.y, (float) o->cylinder.l.p1.z);
6839:97:        oprint (" p2 = (%f, %f, %f) \n",
6840:98:                (float) o->cylinder.l.p2.x, (float) o->cylinder.l.p2.y, (float) o->cylinder.l.p2.z);
6841:99:        oprint ("# l = %f ", (float) o->cylinder.l.l);
6842:100:        oprint (" direction = (%f, %f, %f) \n",
6843:101:                (float) o->cylinder.l.u.x, (float) o->cylinder.l.u.y, (float) o->cylinder.l.u.z);
6844:102:        oprint ("# radius = %f \n", (float) o->cylinder.r);
6845:103:        break;
6846:104:    case CIRCLE:
6847:105:        oprint ("# p = (%f, %f, %f), ",
6848:106:                (float) o->circle.u.x, (float) o->circle.u.y, (float) o->circle.u.z);
6849:107:        oprint ("direction = (%f, %f, %f) \n",
6850:108:                (float) o->circle.p.x, (float) o->circle.p.y, (float) o->circle.p.z);
6851:109:        oprint ("# radius = %f \n", (float) o->circle.r);
6852:110:        break;
6853:111:    case PLANE_SEGMENT:
6854:112:        break;
6855:113:    case ELLIPSE:
6856:114:        break;
6857:115:    }
6858:116:}
6859:117:
6860:118:void output_object (Object * o)
6861:119:{
6862:120:    print_all_object_data (o);
6863:121:    switch (o->type) {
6864:122:    case VECTOR:
6865:123:        break;
6866:124:    case POINT:
6867:125:        break;
6868:126:    case LINE_SEGMENT:
6869:127:        print_line (&(o->line));
6870:128:        break;
6871:129:    case CYLINDER:
6872:130:        print_cylinder (&(o->cylinder));
6873:131:        break;
6874:132:    case CIRCLE:
6875:133:        break;
6876:134:    case PLANE_SEGMENT:
6877:135:        break;
6878:136:    case SURFACE:
6879:137:        print_surface (&(o->surface));
6880:138:        break;
6881:139:    }
6882:140:}
6883:1:
6884:2:
6885:3:#ifndef  _OUTPUT_OBJECT_H
6886:4:#define  _OUTPUT_OBJECT_H
6887:5:
6888:6:void output_object (Object * o);
6889:7:
6890:8:
6891:9:#endif
6892:1:
6893:2:/*****************************************************************************************/
```

```
6894:3:/* picsetup.c - setup and destroy pictures and views                                    */
6895:4:/*****************************************************************************************/
6896:5:
6897:6:#include <config.h>
6898:7:#include "global.h"
6899:8:#include <stdlib.h>
6900:9:#include <stdio.h>
6901:10:#include <sys/types.h>
6902:11:
6903:12:#ifdef HAVE_UNISTD_H
6904:13:#include <unistd.h>
6905:14:#endif
6906:15:
6907:16:#include <my_string.h>
6908:17:#include <sys/stat.h>
6909:18:
6910:19:#ifdef HAVE_FCNTL_H
6911:20:#include <fcntl.h>
6912:21:#endif
6913:22:
6914:23:#include <stdlib.h>
6915:24:#include <stdarg.h>
6916:25:
6917:26:#ifdef HAVE_SYS_TIME_H
6918:27:#include <sys/time.h>
6919:28:#endif
6920:29:
6921:30:#include "regex.h"
6922:31:
6923:32:#include "display.h"
6924:33:#include "app_glob.c"
6925:34:#include "hugeimage.h"
6926:35:#include "widget3d.h"
6927:36:#include "main/marker.h"
6928:37:#include "main/displaycam.h"
6929:38:#include "picsetup.h"
6930:39:#include "dialog.h"
6931:40:#include "stringtools.h"
6932:41:#include "callback.h"
6933:42:
6934:43:
6935:44:/* returns 1 on error */
6936:45:int setup_picture (const char *ident, Picture * image, int x, int y, \
6937:46:                   int width, const char *file)
6938:47:{
6939:48:    Window win;
6940:49:    int h;
6941:50:    image->width = width;
6942:51:    image->height = 0;
6943:52:    image->mag = 4;
6944:53:
6945:54:    win = Cdrawwindow (catstrs (ident, ".main_win", 0), CMain, \
6946:55:                       x, y, image->width + 16, image->width, "");
6947:56:    image->main_win = Cwidget (catstrs (ident, ".main_win", 0));
6948:57:    image->main_image = Cdrawhugebwimage (catstrs (ident, ".main_image", 0), \
6949:58:                      win, 6, 6, &(image->width), &(image->height), file);
6950:59:
6951:60:    if (image->main_image == 0) {
6952:61:        Cundrawwidget (catstrs (ident, ".main_win", 0));
6953:62:        memset (image, 0, sizeof (struct picwithzoom));
6954:63:        return 1;
6955:64:    }
6956:65:    Csetwidgetsize (Cwidget (catstrs (ident, ".main_win", 0))->ident, \
6957:66:                    image->width + 16, h = image->height + 16 + 150);
6958:67:
6959:68:    image->Ttext = Cdrawtext (catstrs (ident, ".text", 0), win, 10, h - 30, file);
6960:69:    image->Binfo = Cdrawbutton (catstrs (ident, ".caminfo", 0), win, 10, h - 150, \
6961:70:                                100, 20, " Cam Info ");
6962:71:    image->Blast = Cdrawbutton (catstrs (ident, ".lastM", 0), win, 120, h - 150, \
6963:72:                                100, 20, " Remove Last ");
6964:73:    image->Ball = Cdrawbutton (catstrs (ident, ".allM", 0), win, 230, h - 150, \
6965:74:                               100, 20, " Remove All ");
6966:75:    image->Bcalibrate = Cdrawbutton (catstrs (ident, ".calibrate", 0), win, 10, \
6967:76:                                     h - 120, 100, 20, " Calibrate ");
6968:77:    image->Bkill = Cdrawbutton (catstrs (ident, ".kill", 0), win, \
6969:78:                                120, h - 120, 100, 20, " Kill View ");
6970:79:    image->Bleave = Cdrawbutton (catstrs (ident, ".leave", 0), win, 10, \
6971:80:                                 h - 90, 100, 20, " Leave Out: ");
6972:81:    image->Tleave = Cdrawtext (catstrs (ident, ".text", 0), win, 120, \
6973:82:                               h - 90, " 0 ");
```

```
6974:83:    image->Ssig = Cdrawbutton (catstrs (ident, ".sig", 0), win, 10, \
6975:84:                                h - 60, 100, 20, " Aspect Rat: ");
6976:85:
6977:86:    win = Cdrawwindow (catstrs (ident, ".zoom_win", 0), CMain, x, y, \
6978:87:                       ZOOMSIZE + 16, ZOOMSIZE + 16, ".zoom_win");
6979:88:    image->zoom_win = Cwidget (catstrs (ident, ".zoom_win", 0));
6980:89:    image->zoom_image = Cdrawzoombox (catstrs (ident, ".zoom_image", 0), \
6981:90:                catstrs (ident, ".main_image", 0), win, 6, 6, ZOOMSIZE, \
6982:91:                                      ZOOMSIZE, 0, 0, image->mag);
6983:92:
6984:93:    Caddcallback (image->main_image->ident, cb_mainimage);
6985:94:    Caddcallback (image->zoom_image->ident, cb_zoomimage);
6986:95:
6987:96:    image->main_rect = Cdrawpicture (catstrs (ident, ".main_rect", 0), \
6988:97:                                     image->main_image->winid, 2, 2, 2);
6989:98:    image->zoom_rect = Cdrawpicture (catstrs (ident, ".zoom_rect", 0), \
6990:99:                                     image->zoom_image->winid, 2, 2, 2);
6991:100:    image->main_markers = Cdrawpicture (catstrs (ident, ".main_markers", 0), \
6992:101:                                    image->main_image->winid, 2, 2, 1024);
6993:102:    image->zoom_markers = Cdrawpicture (catstrs (ident, ".zoom_markers", 0), \
6994:103:                                    image->zoom_image->winid, 2, 2, 1024);
6995:104:
6996:105:    Caddcallback (image->Binfo->ident, cb_draw_cam_data);
6997:106:    Caddcallback (image->Blast->ident, cb_removelastmarker);
6998:107:    Caddcallback (image->Ball->ident, cb_clearallmarkers);
6999:108:    Caddcallback (image->Bcalibrate->ident, cb_calibrate);
7000:109:    Caddcallback (image->Bleave->ident, cb_leave);
7001:110:    Caddcallback (image->Ssig->ident, cb_sigma);
7002:111:    Caddcallback (image->Bkill->ident, cb_killimage);
7003:112:
7004:113:    image->xzoom = image->yzoom = 0;
7005:114:    image->real_width = CHugeImageRealWidth (image->main_image->ident);
7006:115:    image->real_height = CHugeImageRealHeight (image->main_image->ident);
7007:116:    image->x0 = image->real_width / 2;
7008:117:    image->y0 = image->real_height / 2;
7009:118:
7010:119:    return 0;
7011:120:}
7012:121:
7013:122:
7014:123:
7015:124:void destroy_pic (Picture * p)
7016:125:{
7017:126:    if (p && p->main_image) {
7018:127:        Cundrawwidget (p->main_markers->ident);
7019:128:        Cundrawwidget (p->zoom_markers->ident);
7020:129:        Cundrawwidget (p->main_rect->ident);
7021:130:        Cundrawwidget (p->zoom_rect->ident);
7022:131:        Cundrawwidget (p->main_win->ident);
7023:132:        Cundrawwidget (p->zoom_win->ident);
7024:133:    }
7025:134:    clear (p, Picture);
7026:135:}
7027:136:
7028:137:
7029:138:void init_view (View * v)
7030:139:{
7031:140:    clear (v, View);
7032:141:}
7033:142:
7034:143:void destroy_view (View * v)
7035:144:{
7036:145:    if (!v->filename)
7037:146:        return;
7038:147:    destroy ((void *) &(v->filename));
7039:148:    destroy_pic (&(v->pic));
7040:149:    init_view (v);
7041:150:}
7042:151:
7043:152:void destroy_current_view (Desktop * d)
7044:153:{
7045:154:    if (!d->num_views)
7046:155:        return;
7047:156:    destroy_view (&(d->view[d->current_view]));
7048:157:    if (d->current_view == d->num_views - 1 && d->num_views > 0)
7049:158:        d->num_views--;
7050:159:}
7051:160:
7052:161:int setup_view (Desktop * d, char *filename, int x, int y, int i)
7053:162:{
```

```
7054:163:    static long count = 0;
7055:164:    count++;
7056:165:    clear (&d->view[i].pic, Picture);
7057:166:    return setup_picture (catstrs ("picture", itoa (count), 0), \
7058:167:                          &(d->view[i].pic), x, y, 380, filename);
7059:168:}
7060:169:
7061:170:/* return 1 on error */
7062:171:int new_view (Desktop * d, char *filename)
7063:172:{
7064:173:    int x = 50, y = 50;
7065:174:    int i = 0;
7066:175:
7067:176:    if (exists (d, d->current_view)) {
7068:177:        x = d->view[d->current_view].pic.main_win->x + 20;
7069:178:        y = d->view[d->current_view].pic.main_win->y + 20;
7070:179:    }
7071:180:    if (d->num_views)
7072:181:        for (i = 0; i < d->num_views; i++)
7073:182:            if (!exists (d, i))
7074:183:                break;
7075:184:    if (i == d->num_views)
7076:185:        d->num_views++;
7077:186:    set_current_view (d, i);
7078:187:
7079:188:    init_view (&(d->view[i]));
7080:189:    if (setup_view (d, filename, x, y, i))
7081:190:        return 1;
7082:191:
7083:192:    d->view[i].filename = strdup (filename);
7084:193:
7085:194:    return 0;
7086:195:}
7087:196:
7088:197:void load_view (Desktop * d)
7089:198:{
7090:199:    char *s;
7091:200:    s = Cgetfile (0, 0, 0, d->image_dir, "", " Load Image ");
7092:201:    if (s)
7093:202:        if (*s)
7094:203:            new_view (d, s);
7095:204:}
7096:1:#ifndef _PIC_SETUP_H
7097:2:#define _PIC_SETUP_H
7098:3:
7099:4:/* returns 1 on error */
7100:5:int setup_picture (const char *ident, Picture * image, \
7101:6:                  int x, int y, int width, const char *file);
7102:7:void destroy_pic (Picture * p);
7103:8:void destroy_view (View * v);
7104:9:void destroy_current_view (Desktop * d);
7105:10:int new_view (Desktop * d, char *filename);
7106:11:void load_view (Desktop * d);
7107:12:
7108:13:#endif                          /* _PIC_SETUP_H */
7109:1:/*****************************************************************************************/
7110:2:/* savewindow.c - saves the rendered 3D scene as a targa file                            */
7111:3:/*****************************************************************************************/
7112:4:
7113:5:#include "display.h"
7114:6:#include "imagewidget.h"
7115:7:#include "../global.h"
7116:8:#include "app_glob.c"
7117:9:#include "dialog.h"
7118:10:
7119:11:/*


 */
7120:12:
7121:13:/* returns 1 on error */
7122:14:int save_window (Window win, int x, int y, int width, int height, const char *file)
7123:15:{
7124:16:    XImage *image;
7125:17:    byte *p_byte;
7126:18:    byte *p, *q, *pix;
7127:19:    word *p_word;
7128:20:    quad_t *p_quad;
7129:21:
7130:22:    Colormap cmap;
```

```
7131:23:
7132:24:    int i, j, n;
7133:25:
7134:26:    image = XGetImage (CDisplay, win, x, y, width, height, \
7135:27:                       (unsigned long) -1, ZPixmap);
7136:28:    if (!image) {
7137:29:        Cerrordialog (0, 0, 0, " Save Window ", " Unable to create XImage ");
7138:30:        return 1;
7139:31:    }
7140:32:    p = image->data;
7141:33:    p_byte = p;
7142:34:    p_word = (word *) p;
7143:35:    p_quad = (quad_t *) p;
7144:36:
7145:37:    pix = Cmalloc (width * height * 3);
7146:38:    q = pix;
7147:39:
7148:40:    switch (image->bits_per_pixel) {
7149:41:    case 8:{
7150:42:            XColor *c;
7151:43:            n = DisplayCells (CDisplay, DefaultScreen (CDisplay));
7152:44:            c = Cmalloc (n * sizeof (XColor));
7153:45:            for (i = 0; i < n; i++)
7154:46:                c[i].pixel = i;
7155:47:            cmap = DefaultColormap (CDisplay, DefaultScreen (CDisplay));
7156:48:            XQueryColors (CDisplay, cmap, c, n);
7157:49:
7158:50:            for (i = 0; i < height; i++) {
7159:51:                p_byte = p;
7160:52:                for (j = 0; j < width; j++) {
7161:53:                    *q++ = c[*p_byte].blue >> 8;
7162:54:                    *q++ = c[*p_byte].green >> 8;
7163:55:                    *q++ = c[*p_byte].red >> 8;
7164:56:                    p_byte++;
7165:57:                }
7166:58:                p += image->bytes_per_line;
7167:59:            }
7168:60:            free (c);
7169:61:            break;
7170:62:        }
7171:63:    case 16:{
7172:64:            XColor c;
7173:65:            c.flags = DoRed | DoGreen | DoBlue;
7174:66:            cmap = DefaultColormap (CDisplay, DefaultScreen (CDisplay));
7175:67:            for (i = 0; i < height; i++) {
7176:68:                p_word = (word *) p;
7177:69:                for (j = 0; j < width; j++) {
7178:70:                    c.pixel = *p_word;
7179:71:                    XQueryColor (CDisplay, cmap, &c);
7180:72:                    *q++ = c.blue >> 8;
7181:73:                    *q++ = c.green >> 8;
7182:74:                    *q++ = c.red >> 8;
7183:75:                    p_word++;
7184:76:                }
7185:77:                p += image->bytes_per_line;
7186:78:            }
7187:79:            break;
7188:80:        }
7189:81:    case 24:
7190:82:        free (pix);
7191:83:        Cerrordialog (0, 0, 0, " Save Window ", \
7192:84:            " %s:%d 24bpp images not supported, use 8, 16 or 32bpp. \n" \
7193:85:                      " This may mean changing your display. ");
7194:86:        return 1;
7195:87:    case 32:{
7196:88:            XColor c;
7197:89:            c.flags = DoRed | DoGreen | DoBlue;
7198:90:            cmap = DefaultColormap (CDisplay, DefaultScreen (CDisplay));
7199:91:            for (i = 0; i < height; i++) {
7200:92:                p_quad = (quad_t *) p;
7201:93:                for (j = 0; j < width; j++) {
7202:94:                    c.pixel = *p_quad;
7203:95:                    XQueryColor (CDisplay, cmap, &c);
7204:96:                    *q++ = c.blue >> 8;
7205:97:                    *q++ = c.green >> 8;
7206:98:                    *q++ = c.red >> 8;
7207:99:                    p_quad++;
7208:100:                }
7209:101:                p += image->bytes_per_line;
7210:102:            }
```

```
7211:103:            break;
7212:104:        }
7213:105:    }
7214:106:    writetarga (pix, file, width, height, 0);
7215:107:    free (pix);
7216:108:    return 0;
7217:109:}
7218:110:
7219:111:
7220:112:int save_window_to_file (Window win, int x, int y, int width, int height)
7221:113:{
7222:114:    char *filename;
7223:115:    filename = Cgetfile (0, 0, 0, home_dir, "", " Save Window ");
7224:116:    if (filename)
7225:117:        if (*filename) {
7226:118:            save_window (win, x, y, width, height, filename);
7227:119:            free (filename);
7228:120:            return 0;
7229:121:        }
7230:122:    return 1;
7231:123:}
7232:1:/*****************************************************************************************/
7233:2:/* simplex.c - simplex optimisation algorithm                                            */
7234:3:/*****************************************************************************************/
7235:4:
7236:5:#include <config.h>
7237:6:#include "global.h"
7238:7:#include <math.h>
7239:8:#include <stdio.h>
7240:9:#include "matrix.h"
7241:10:
7242:11:/*
7243:12:   Finds an n'th dimensional simplex about the origin.
7244:13:   The n+1 vertices of the simplex are stored as rows
7245:14:   in the returned matrix. Each vertex is d units
7246:15:   euclidian distance from the origin.
7247:16: */
7248:17:Matrix *Mageneratesimplex (int n, double d)
7249:18:{
7250:19:    Matrix *A = Maallocmatrix (n + 1, n, 'd');
7251:20:    Matrix *v1 = NULL, *v2 = NULL;
7252:21:    double s, a, b;
7253:22:    int i, j;
7254:23:
7255:24:    d *= d;
7256:25:
7257:26:    Mard ((*A), 0, 0) = 1;
7258:27:    Mard ((*A), 1, 0) = -1;
7259:28:
7260:29:    for (i = 1; i < n; i++) {
7261:30:
7262:31:        v1 = Masubmatrix (v1, A, i - 1, 0, 1, i);
7263:32:        v2 = Masubmatrix (v2, A, i, 0, 1, i);
7264:33:
7265:34:        s = Madistance (v1, v2);
7266:35:        s *= s;
7267:36:
7268:37:        b = 0.5 * (s - 2 * d) / (sqrt (s - d));
7269:38:        a = sqrt (d / (d + b * b));
7270:39:
7271:40:        Mascale (A, a);
7272:41:
7273:42:        a *= b;
7274:43:
7275:44:        for (j = 0; j < i; j++) {
7276:45:            Mard ((*A), j, i) = -a;
7277:46:            Mard ((*A), i + 1, j) = 0;
7278:47:        }
7279:48:
7280:49:        Mard ((*A), i, i) = -a;
7281:50:        Mard ((*A), i + 1, i) = sqrt (d);
7282:51:
7283:52:    }
7284:53:
7285:54:    Mafreematrix (v2);
7286:55:    Mafreematrix (v1);
7287:56:
7288:57:    return A;
7289:58:}
7290:59:
```

```
7291:60:/*
7292:61:   Same as above, but r (row vector) is added to each vertice (i.e. row).
7293:62:   Matrix must be Mafreematrix'd
7294:63: */
7295:64:Matrix *Mageneratesimplexabout (Matrix * r, int n, double d)
7296:65:{
7297:66:    int i, j;
7298:67:    Matrix *A;
7299:68:
7300:69:    A = Mageneratesimplex (n, d);
7301:70:
7302:71:    for (i = 0; i < n + 1; i++)
7303:72:        for (j = 0; j < n; j++)
7304:73:            Mard ((*A), i, j) += Mard ((*r), 0, j);
7305:74:
7306:75:    return A;
7307:76:}
7308:77:
7309:78:#define NMAX 5000
7310:79:#define GET_PSUM \
7311:80:        for(j=0;j<ndim;j++) { \
7312:81:            for(sum=0.0,i=0;i<mpts;i++) sum += p[i][j]; \
7313:82:            psum[j]=sum; \
7314:83:        }
7315:84:
7316:85:int amoeba (double **p, double y[], int ndim, double ftol,
7317:86:            double (*funk) (double[]),
7318:87:            int (*callback) (double, double, double *, int), int *nfunk)
7319:88:{
7320:89:    double amotry (double **p, double y[], double psum[], int ndim,
7321:90:                   double (*funk) (double[]), int ihi, double fac);
7322:91:    int i, ihi, ilo, inhi, j, mpts = ndim + 1, ret_val = 0;
7323:92:    double rtol, sum, ysave, ytry, *psum;
7324:93:
7325:94:    psum = Mamalloc ((ndim + 2) * sizeof (double));
7326:95:    *nfunk = 0;
7327:96:    GET_PSUM;
7328:97:    for (;;) {
7329:98:        ilo = 0;
7330:99:        ihi = y[0] > y[1] ? (inhi = 1, 0) : (inhi = 0, 1);
7331:100:        for (i = 0; i < mpts; i++) {
7332:101:            if (y[i] <= y[ilo])
7333:102:                ilo = i;
7334:103:            if (y[i] > y[ihi]) {
7335:104:                inhi = ihi;
7336:105:                ihi = i;
7337:106:            } else if (y[i] > y[inhi] && i != ihi)
7338:107:                inhi = i;
7339:108:        }
7340:109:
7341:110:        rtol = 2.0 * fabs (y[ihi] - y[ilo]) / (fabs (y[ihi]) + fabs (y[ilo]));
7342:111:
7343:112:        if (*nfunk >= NMAX) {
7344:113:            fprintf (stderr, "Warning:stereo:%s:%d:" \
7345:114:                     " %d functional evalutations exceeded in optimisation - returning\n",
7346:115:                     __FILE__, __LINE__, (int) NMAX);
7347:116:            goto fin;
7348:117:        }
7349:118:        if (rtol < ftol || fabs (y[ilo]) < ftol * ftol)
7350:119:            goto fin;
7351:120:        if (callback) {
7352:121:            if ((*callback) (rtol, y[ilo], p[ilo], *nfunk)) {
7353:122:                ret_val = 1;
7354:123:              fin:
7355:124:                fswap (y[0], y[ilo]);
7356:125:                for (i = 0; i < ndim; i++)
7357:126:                    fswap (p[0][i], p[ilo][i]);
7358:127:                break;
7359:128:            }
7360:129:        }
7361:130:/*      if (*nfunk >= NMAX)
7362:131:   Maerror ("NMAX exceeded");
7363:132: */
7364:133:        *nfunk += 2;
7365:134:        ytry = amotry (p, y, psum, ndim, funk, ihi, -1.0);
7366:135:        if (ytry <= y[ilo])
7367:136:            ytry = amotry (p, y, psum, ndim, funk, ihi, 2.0);
7368:137:        else if (ytry >= y[inhi]) {
7369:138:            ysave = y[ihi];
7370:139:            ytry = amotry (p, y, psum, ndim, funk, ihi, 0.5);
```

```
7371:140:            if (ytry >= ysave) {
7372:141:                for (i = 0; i < mpts; i++) {
7373:142:                    if (i != ilo) {
7374:143:                        for (j = 0; j < ndim; j++)
7375:144:                            p[i][j] = psum[j] = 0.5 * (p[i][j] + p[ilo][j]);
7376:145:                        y[i] = (*funk) (psum);
7377:146:                    }
7378:147:                }
7379:148:                *nfunk += ndim;
7380:149:                GET_PSUM;
7381:150:            }
7382:151:        } else
7383:152:            --(*nfunk);
7384:153:    }
7385:154:    free (psum);
7386:155:    return ret_val;
7387:156:}
7388:157:
7389:158:double amotry (double **p, double y[], double psum[], int ndim,
7390:159:               double (*funk) (double[]), int ihi, double fac)
7391:160:{
7392:161:    int j;
7393:162:    double fac1, fac2, ytry, *ptry;
7394:163:
7395:164:    ptry = Mamalloc ((ndim + 2) * sizeof (double));
7396:165:    fac1 = (1.0 - fac) / ndim;
7397:166:    fac2 = fac1 - fac;
7398:167:    for (j = 0; j < ndim; j++)
7399:168:        ptry[j] = psum[j] * fac1 - p[ihi][j] * fac2;
7400:169:    ytry = (*funk) (ptry);
7401:170:    if (ytry < y[ihi]) {
7402:171:        y[ihi] = ytry;
7403:172:        for (j = 0; j < ndim; j++) {
7404:173:            psum[j] += ptry[j] - p[ihi][j];
7405:174:            p[ihi][j] = ptry[j];
7406:175:        }
7407:176:    }
7408:177:    free (ptry);
7409:178:    return ytry;
7410:179:}
7411:180:
7412:181:
7413:182:
7414:183:/*
7415:184:   Does Nelder and Mead (Downhill Simplex Method) minimisation of
7416:185:   the function funk.
7417:186:   returns minimising vector in result. ftol is the final tolerance
7418:187:   and itol is the initial tolerance (these are actually the size of the
7419:188:   final or initial simplexes. funk takes an ndim dimensional vector
7420:189:   of doubles and returns a function value.
7421:190:   Returns the number of functional iterations performed.
7422:191:   callback can be used by the user to to intermediate
7423:192:   processes during the optimisation --- for example monotoring
7424:193:   by graphical display. A NULL value my be given if you
7425:194:   have no use for it.
7426:195:   callback takes four values: the current tolerance, the current error,
7427:196:   the vertex associated with that error, and the number of times funk
7428:197:   has been called. It must return 0. A non-zero return value signals
7429:198:   for the itteration to stop and the current lowest point to be
7430:199:   returned.
7431:200: */
7432:201:
7433:202:int simplex_optimise (double *result, int ndim, double ftol, double itol,
7434:203:                      double (*funk) (double *), int (*callback) (double, double, double *, int))
7435:204:{
7436:205:    Matrix *r = Maallocmatrix (1, ndim, 'd');
7437:206:    Matrix *start_simplex;
7438:207:    int stop = 1;
7439:208:
7440:209:    double *y = Mamalloc ((ndim + 1) * sizeof (double));
7441:210:    int i, j, nfunk[3] =
7442:211:    {0, 0, 0};
7443:212:
7444:213:    memcpy (&(Mard ((*r), 0, 0)), result, ndim * sizeof (double));
7445:214:
7446:215:    for (j = 0; j < 3 && stop; j++) {  /* restart the algorithm three times to ensure
7447:216:                                          that there is a true minumim */
7448:217:
7449:218:        start_simplex = Mageneratesimplexabout (r, ndim, itol);
7450:219:
```

```
7451:220:        for (i = 0; i < ndim + 1; i++)
7452:221:            y[i] = funk (&(Mard ((*start_simplex), i, 0)));
7453:222:
7454:223:        if (amoeba ((double **) start_simplex->d, y, ndim,
7455:224:                    ftol, funk, callback, &(nfunk[j])))
7456:225:            stop = 0;
7457:226:
7458:227:/* amoeba always returns with the first simplex point having least error */
7459:228:        for (i = 0; i < ndim; i++)
7460:229:            Mard ((*r), 0, i) = Mard ((*start_simplex), 0, i);
7461:230:/* r now contains the point */
7462:231:
7463:232:        Mafreematrix (start_simplex);  /*Mageneratsimplexabout */
7464:233:
7465:234:        itol = ftol;
7466:235:    }
7467:236:
7468:237:    memcpy (result, &(Mard ((*r), 0, 0)), ndim * sizeof (double));
7469:238:
7470:239:    free (y);
7471:240:    Mafreematrix (r);
7472:241:
7473:242:    return nfunk[0] + nfunk[1] + nfunk[2];
7474:243:}
7475:1:#ifndef SIMPLEX_H
7476:2:#define SIMPLEX_H
7477:3:
7478:4:Matrix *Mageneratesimplex (int n, double d);
7479:5:
7480:6:Matrix *Mageneratesimplexabout (Matrix * r, int n, double d);
7481:7:
7482:8:int amoeba (double **p, double y[], int ndim, double ftol,
7483:9:            double (*funk) (double[]),
7484:10:            int (*callback) (double, double, double *, int), int *nfunk);
7485:11:
7486:12:int simplex_optimise (double *result, int ndim, double ftol, double itol,
7487:13:                      double (*funk) (double *),
7488:14:                      int (*callback) (double, double, double *, int));
7489:15:
7490:16:#endif                          /* SIMPLEX_H */
7491:17:
7492:18:/*
```