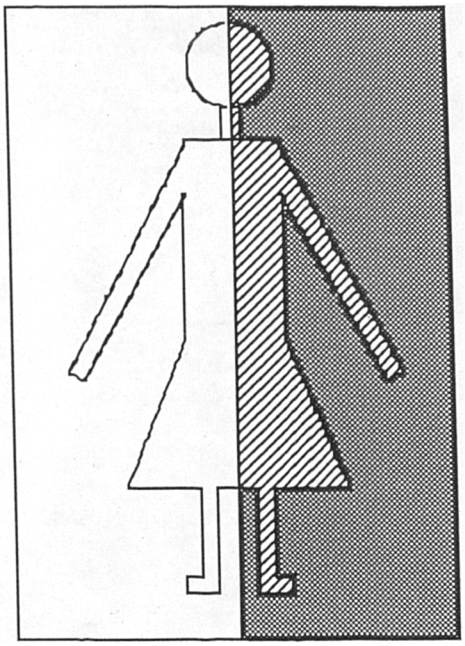
AUTOMATED SHAPE ANTHROPOMETRY

By Gordon M West



A Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of the Loughborough University of Technology

 November 1993

ACKNOWLEDGEMENTS

It is here with grateful thanks that I wish to acknowledge the efforts of Professor R. Goldsmith and Professor P.R.M.Jones who so readily agreed to be my Research Dir ctor and Supervisor respectively. To David Harris who built the electronics and Jeffrey Read who built the mechanics, of the scanner , to Louise Deamer who helped in every way to operate the scanner and build up the data base and also with the later editing of the data. To Dr Katherine Brooke-Wavell who took over from Louise later in the project and to Gwenda Scott who helped with the layout of this thesis.

DECLARATION

This is to certify that the subject of this thesis is all my own work except where stated. The initial design of the electronic circuitry of a single camera system was mine but this was later expanded to incorporate fourteen cameras by David Harris who designed and built of the camera interface. I also provided all the computer programs for the interface computer and the Silicon Graphics computer. Jeffrey Read provided the mechanical design and construction of the whole scanner system. During the data collection and analysis phase I was assisted by Louise Deamer and later by Dr Katherine Brooke-Wavell.

Gordon M. West

ABSTRACT

In medicine, ergonomics, the clothing Industry and many other areas such as the design of 'g' suits for military aeroplane pilots and protective clothing for chemical warfare, there is a requirement for the accurate 3-D measurement of the size and shape of the human form.

To meet this need a novel whole body scanner has been designed which is capable of measuring both the size and shape of people in a non invasive socially acceptable manner. The scanner uses structured light and an array of television cameras to view free standing subjects while they are being rotated on an electrically driven turntable. The accuracy and repeatability of the scanner is as good as trained anthropometrists using traditional manual methods.

A computer program has been written which uses a cubic spline interpolation method to edit and interrogate the data from the scanner and arrange it in a *shape matrix* form. This is a new way of arranging the data which allows for the 3-D average of several bodies to be obtained and also for the comparison of one body with another. A technique which is essential if 3-D survey work is to be undertaken.

Using master files which contain information from a data base of previously scanned people and eight circumferential measurements it is possible to re-create body forms of any size but which correspond to the average shape for that size. The re-creation of body shapes from eight circumferences is accurate enough for the manufacture of all but the most close fitting garments but may be more useful in the future as a replacement for somatotyping (physique classification). It is perfectly possible to manipulate the eight circumferences to create body torsos of almost any shape. Conversely a torso of almost any shape could be defined by eight two digit numbers.

Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION AND REVIEW OF PREVIOUS WORK

Introduction

The measurement of the size and shape of the human body is an important source of information for application in garment manufacture McCartney,Hinds.(1992), medicine, ergonomics and many other areas. Some of the problems to be solved are the determination of the size and shape of the population to provide a norm for the study of abnormality or the effects of ageing and disease Smith,Jones, and West.(1990) or changes due to body growth and development, seasonal biological rhythms or nutritional or pathological conditions. There is a need to determine size and shape of people for the design of protective clothing for use in hostile environments such as special 'g' suits for military aeroplane pilots, for protective clothing for chemical warfare and protective clothing for many hazardous industrial environments. Realistic body shapes are required for architectural models, for use in ergonomics in the evaluation of escape envelopes and for interference checks. Now that computers, with ever increasing power, are commonplace the traditional ways of measurement, using tapemeasures, stadiometers, callipers etc., are inadequate for modern requirements. What is now required is an automatic digitiser for the human body so that total shape information is available for full 3-D manipulation of the data in a computer. A novel method of digitising the human body known as The Loughborough Anthropometric Shadow Scanner (LASS),West and Jones.(1985) is further explained in Jones, West, Harris and Read (1989). With this scanner it is possible to generate a very large quantity of body shape data. These data need modification and manipulation before they can be used. For instance, when a person is measured on LASS their arms are included in the resultant data and they interfere with typical circumferential measurements usually taken under the arms. A suitable 3-D editing program is described which allows the arms to be removed. Included in

the same edit computer program are means of taking measurements from the digitised body forms. The measurements can be through the subject, as in engineering type dimensions, surface distance or surface distance as measured by a tape measure. These measurements can be taken horizontally or at any oblique angle. Surface area and volume can also be calculated. The output of the editor program is a standard file format such that different body shapes can be directly compared and also averaged, which is very important in size survey work. The method devised by me is called " Shape Matrix Analysis" which is a method of comparing the physical position of a point or feature on one person with a similar point or feature on another person in a way which is *independent of stature*. If the positions of single points correspond between two people, then the correspondence of groups of points can be taken as a measure of shape. Differences between the points of certain well defined groups can be calculated and mean and standard deviations computed to give relative factors to differentiate one shape from another. The ability to compare and average shapes is of major importance. Having measured many body forms it is possible to use these data to recreate body shapes from a few simple circumferential measurements. The recreated shapes will represent the average shape of the population represented in a master file, but scaled to match an individual.

Review of previous work

Human morphology has been studied from the times of the ancient Greeks in an attempt to classify the way people are prone to certain ailments. Hippocrates (c 400 BC) thought that there were two physical types which he called the *phthisic habitus* and the *apoplectic habitus*. The former had a long thin body subject to tuberculosis while the latter had a short thick body subject to apoplexy. Many more attempts at classification of the human form have been made until the present day and are referred to in Sheldon, Stevens and Tucker (1940). The main work of Sheldon has been to further refine human classification into somatotypes. He maintains that the human form is intermediate between three extremes, the *Endomorph*, the *Mesomorph* and the *Ectomorph*. The method of classification involved a series of detailed measurements taken from photographs but there was a significant

element of anthroscopy involved, in other words a judgement of the appearance of the photographs modified by the measurements.

This work has been useful in many fields since it is far more accurate to have a known somatotype than a simple average. More recently, size surveys have been carried out, mainly for the clothing trade but with obvious spin-off in the medical field. These have been direct measurements of the body using tape-measures and callipers etc. With modern tight fitting garments the number of desirable measurements exceeds forty and is limited by the costs of taking so many measurements and the time that the available subjects are prepared to give. Even if such measurements are taken the data are not sufficient to provide answers to questions such as " What is the shape of the leg hole in a swim suit?". Also the data are not sufficient to provide accurate three dimensional computer simulation models.

There have been many attempts to measure various parts of the human body automatically, but none have successfully measured the whole body. Measurement of component parts has always been an essential part of any manufacturing process. Because most parts are rigid structures there was little requirement for non-contacting methods. More recently non-contacting methods have been introduced because of the need to take large numbers of measurements at speed. The medical profession need to take measurements for a variety of reasons but especially to determine the 3-D shape of a face before and after re constructive surgery or to measure the shape of the back for diagnosis and correction of scoliosis.

When taking a large number of measurements on a human body obviously a non-contacting method is desirable. Non contact measurement at a distance was achieved by surveyors before the turn of the century to determine the shape of the surface of the earth. The system of triangulation which they used is the basis of the modern methods except that the theodolite is replaced with some kind of light sensing array, usually a television camera. Also borrowed from land surveyors is stereophotogrammetry used for terrain mapping by taking stereo pairs of aerial photographs.

This method was recently used by Burke,Banks,Beard,Tee and Hughes (1983) to study changes in soft tissue morphology induced by facial surgery.

A system of triangulation was used by Turner-Smith(1982) to map the contours of the back for assessment and treatment of scoliosis and other disorders of the spine. A light projector was used which projected a horizontal strip of light onto the patients back which was viewed by a television camera. Because the television camera field of view was at an angle to the optical axis of the projector the line of light was displaced by an amount proportional to the position of the back profile at any particular point. By knowing the geometry of the system and by triangulation the surface co-ordinates were calculated by computer. To take measurements over the whole back rather than over one line the projector and camera were rigidly mounted together and the whole pivoted about an axis to cause the projected line of light to scan the whole body. A rotary transducer was used to measure the angle of rotation.

Another system for scoliosis measurement was made by Ishida,Mori,Kishimoto,Nakazima and Tsubakimoto (1987). They used ten light emitting diode (LED) projectors which each projected a spot of light onto the patients skin. The vertical line of light spots were viewed by a line sensor camera and their positions determined by triangulation. To measure all round the body the whole assembly was rotated about a vertical axis coaxial with the axis of the patient.

Magnant (1985) produced a system which used a horizontal sheet of light to completely surrounded the body being measured. To measure the whole body a framework carrying the projectors and cameras was moved to scan the body from head to toe.

A further method by Frobin and Hierholzer (1983) removed the need for mechanical scanning. Rather than a single line of projected light viewed by a television camera this system used multiple parallel lines of light which completely covered the measuring area. Because the line spacing was known it was possible to compute by triangulation the co-ordinates of points on each line. This method will only measure the surface of one side of the body at a time. A commercial version of this method was produced by Axis Software Systems Ltd., Northampton.

Gourlay, Kaye, Denison, Peacock and Morgan (1984) used a similar system for lung function studies using vertical light stripes. They took photographs of the illuminated body with a still camera as an intermediate step to inputting the data to a computer. Generally operating in the same fashion Lewis and Sopwith (1986) produced a

system whereby the body surface was illuminated with an array of dots rather than lines. A computer was able to relate the dots seen from each of two cameras and determine their position by triangulation.
Arridge, Moss, Linney and James (1985) used two vertical slices of laser light and a single television camera to measure the shapes of faces for orthodontic and maxillo-facial surgery. The patient was seated on a chair which was mounted on a rotary table.
In addition to the medical applications the clothing trade was the target for several inventions designed to measure a person's silhouette. Vietorisz (1964) used a single light source and an array of photo detectors. Ito (1979) used an array of lights and an array of photo detectors which were rotated round the body being measured. Takada and Esaki (1981) had in principle a similar system but with a different arrangement of lights and photo detectors. Silhouette methods are incapable of measuring re-entrant parts of the body but they can provide sufficient size information to be able to select an appropriate standard size and shape from a library of patterns.
For both medical and clothing applications the Moiré fringe method has been used. Meadows, Johnson and Allen (1970) describe a method whereby the object is both viewed and illuminated through a grating. The horizontal profile of the body being measured distorts the grating shadow which when viewed through the grating again produces a contour map of the surface. This relatively simple system has many drawbacks when used for measuring the human form. An ambiguity exists such that it is difficult to decide between "hills and valleys". Because the method relies on counting fringe patterns there are problems due to random fluctuations of the signal produced by imperfections in the grating and also difficulty with surface reflectance variations of the body being measured. The system only measures one side, not all the way round and leaves a problem of matching a front and back view.

Halioua, Krishnamurphy, Liu and Chiang. (1984) have made improvements to the method whereby two small independent gratings are used, one for the light source and one for the camera. It is then possible to mechanically alter the position of one grating with a servo system to maintain stationary fringes and so deduce the body contour height from a transducer on the grating drive. This method gives the

height of a single point and therefore needs an optical scanning system in order to cover the whole body.

The main disadvantage of any Moiré system is that the contour intervals are small and so the contours become impossibly close together where there are rapid changes of height with distance on certain parts of the body.

Reid, Rixon, Marshall and Stewart (1986) used a Moiré system to measure turbine blades. The contour patterns were analysed in terms of the phase rather than the intensity of the fringes.

Other systems used a laser range finder technique. Clerget, Germain and Kryze (1977) illuminate the object being measured with a scanning laser beam. The reflected light is viewed from two ends of a base line and the parallax due to the differing points of view is measured. From this the range is computed. A similar system described by Addleman and Addleman (1985) is marketed by Cyberware.

NKK Corporation of Japan under the trade name of "Voxelan" are marketing a system by Uesugi (1991) which uses two laser slit light sources viewed by a television camera. The projection angle of each slit light source is encoded and used to triangulate the range of all points being scanned.

Rioux (1984) also made a laser range finder where both the light source and the field of view of the detector were scanned synchronously.

The range finder method requires that the subject is stood on a turntable if an all round view is required. For large objects it may be necessary to vertically translate the range finder head while measuring the displacement with a separate transducer.

Many methods *only measure one side of the body at a time. Because* the body does not maintain a particular posture for any length of time it is not a satisfactory solution to take front and back scans other than simultaneously.

None of the methods so far described have solved the problem of body sway. A person who is standing in a normal position sways a few centimetres in all directions, West (1987). while a person who is mechanically restrained from swaying does not assume a natural posture. A successful measuring device must be capable of making measurements and compensating for the inevitable body sway.

CHAPTER 2

THE MEASURING PROBLEM AND THE LASS SCANNER

The Measuring Problem

The ideal measuring machine would instantaneously take measurements of the whole body while moving or jumping and with an accuracy of a small fraction of a millimetre and with low cost, probably a few hundred pounds. Unfortunately such a specification may not be possible and is certainly beyond the available resources to achieve.

A choice has to be made whether to use available scanning devices such as television cameras, or mechanical scanners using oscillating mirrors or rotating prisms. The former providing scanning of the received signal while the latter providing scanning of the illuminating beam of light. The mechanical scanners were not as cheap as a TV camera and could offer no significantly better specification or long term stability. The use of lasers was not considered an advantage because the possible danger to eyesight, either real or imaginary, may prevent people from volunteering to be measured. Another factor is that a single low power laser does not have sufficient brightness adequately to illuminate more than a small part of the body.

The total shape measurement of the living human body to an accuracy in 3 dimensional space of say ± 2 mm. on any co-ordinate, poses several problems. Since the height of the taller persons extends to 2 metres there is the obvious vertical accuracy requirement of 1 part in 1000. The radial accuracy requirement is less stringent, since the same tolerance of ± 2 mm. on a maximum radius of 500 mm. is only 1 part in 250. A further more serious problem is that living people do not normally stand still, they change shape as they breath and they sway when standing normally in order to maintain their balance. The sway problem can be reduced by providing suitable support, but if this support is not to obscure the measurement, or cause the subject to

stand in an unnatural posture, then it needs to be reduced to the absolute minimum. In addition to the above problems there is need to consider both the speed of measurement and the method. It is not acceptable to use methods which make contact with the person by means of a mechanical probe since the total body surface needs to be measured. Time is also important, as few people would tolerate expending more than a few minutes for measurement. Another problem is that the human body presents a rather lumpy appearance and various parts of it such as the arms or breasts can obscure the view of the measuring apparatus of other parts of the body. Having captured the data there is the problem of presenting it in a useful form. Initially the data must be displayed in order to .ndicate that the data capture scan was successful before the subject departs. Because of the large amount of data collected from one subject, some kind of computer processing is required to handle and store the data, possibly to edit it and to perform data reduction for storage and to extract measurem· nts of every kind.

The Loughborough Anthropometric Shadow Scanner (LASS)

The method proposed as a solution to the above problems is to view a person using television cameras and derive dimensional information from the picture obtained. Because information is required in the round (3-D) the person is rotated on a turn-table so that every possible view can be examined. An initial study using a single camera is described in West (1987

Essentially the system comprises a television camera, turn-table and a computer. The subject being measured stands on the turn-table in the field of view of the television camera and data from the camera are stored in the computer as the subject is rotated. Unfortunately a television camera, in common with ordinary film cameras, represents the scene being viewed in two dimensions only. An ordinary photograph is a projection of the scene on to a flat plane which is parallel to the camera and to the plane of the film in the camera. Because the viewing plane is parallel to the film plane no obvious distortion is noticed when the film is processed. Everyone has observed the effect when taking photographs of tall buildings, of distortion

produced when the camera is inclined upwards, in order to get the top of the building into the field of view.

Two shortcomings are then apparent: 1) That the camera is only capable of recording 2 dimensional information. and 2) When taking an inclined view of a scene there is considerable distortion over and above the relatively small distortion due to the camera lens.

Since LASS is required to measure in three dimensions simultaneously, some further information is required in addition to that obtained by the camera. The extra information is provided by projecting a line of light towards the subject being measured. If the line of light (in particular the contrasting edge between light and shadow) is projected so that it passes exactly through the centre of rotation of the turn-table on which the subject is standing, then the line of light exactly defines the subjects radius. What the camera sees is a projection of the line defining the radius. The most obvious arrangement is to project the light at right angles to the line from the centre of the camera lens, then the camera records a scaled down, but otherwise undistorted view of the radius. Unfortunately this is not a practical solution, since the light falling on any re-entrant parts of the body, would be obscured by some other part of the body. In practice the light projector needs to project light at an angle of approximately 30 degrees to the line from the camera lens to the subject. This rather extreme angle will of course produce considerable distortion when seen by the camera. A description of the feasibility study for such a system is given by West (1987) which deals with a single camera system.

In order to measure a full size human being the system is expanded to include fourteen television cameras in two banks of seven and sixteen light slit projectors in four banks of four (Figure 1).

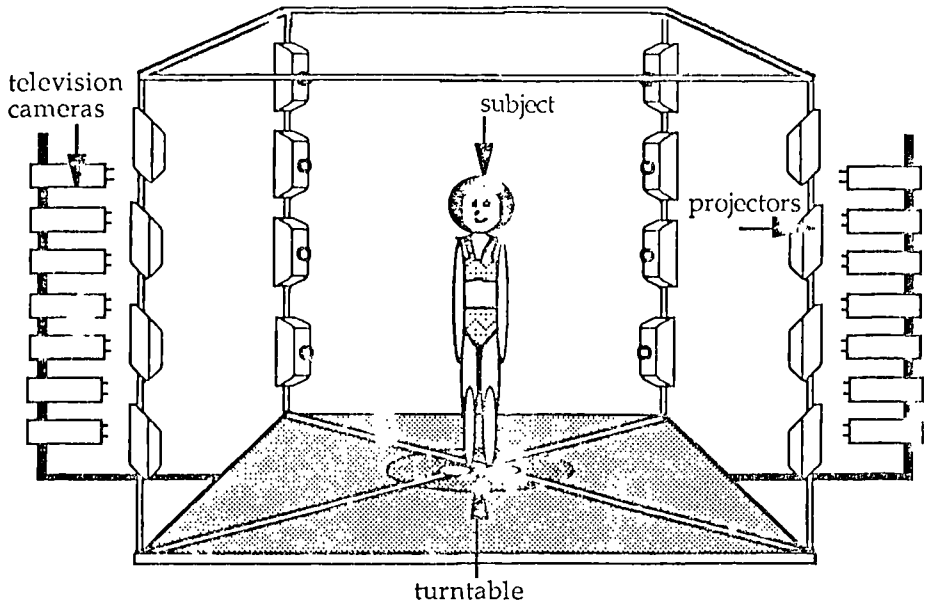


Figure. 1. Schematic representation of the multiple camera system

Technical Description

The simplest arrangement which illustrates the essentials of the system is shown in Figure 2. The person being measured stands on a turn-table within the field of view of a television camera. A projector projects a vertical slit of light towards the person on the turn-table. Because the vertical slit of light is arranged to pass exactly through the centre of rotation of the turn-table, where the light falls on the body defines the radius of the body at the particular angle of the turn-table.

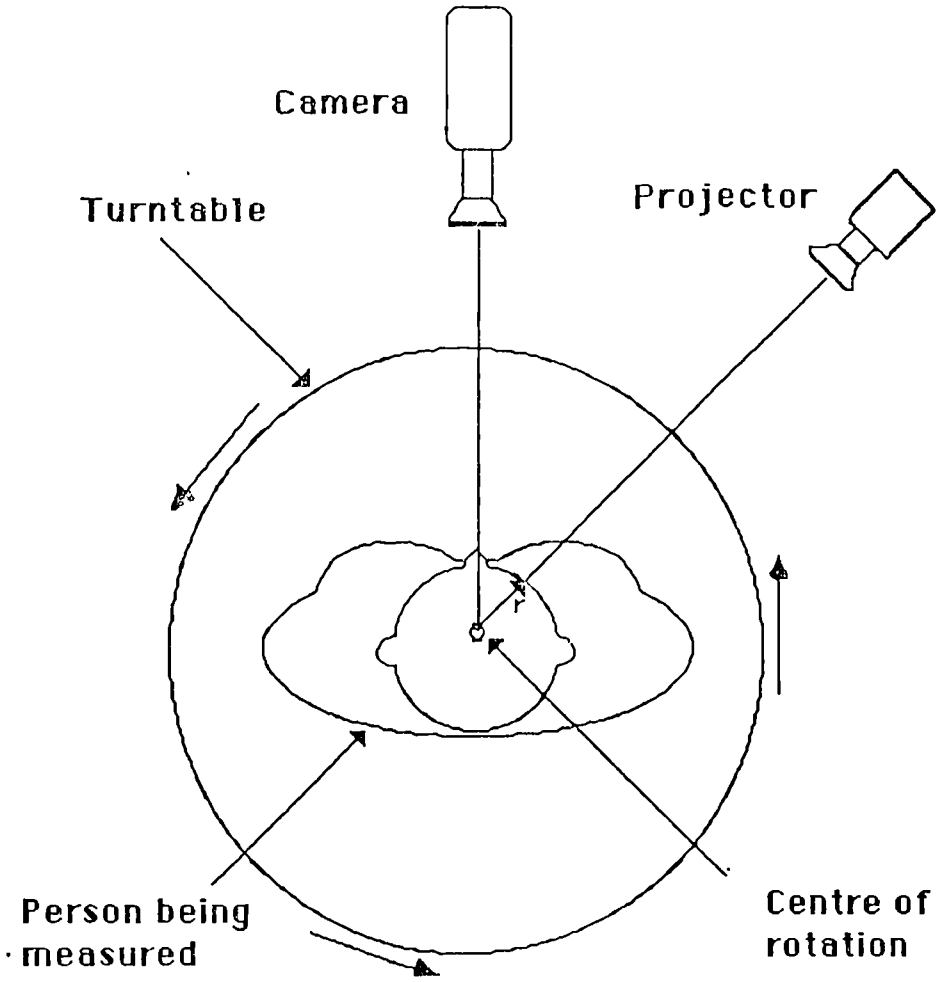


Figure 2. Simple Measuring System

The line of light projected onto the body is detected by the television camera and then, with suitable electronics, is digitised for each line of the television camera scan, and so 3-D measurements can be made in cylindrical co-ordinates, with the position along each linescan of the camera giving the radius, the position of the turn-table giving *theta*, and the particular line of the television scan giving height. The angle of the projector is set at approximately 30 degrees to reduce masking by the shoulders and other protruding parts of the body.

The camera now sees a projection of the radius which can be resolved as :-

radius = camera view / sin θ

In practice, a single television camera uses 312 scan lines each picture frame, and two frames are interlaced to give the effect of 625 lines as seen by a viewer.

In this equipment the 312 scan lines of the camera represented a vertical distance on the body of 1mm per line, giving a total vertical field of view of 312mm for each camera. Since it is intended to measure people up to a height of 2.1 metres it is necessary to use seven television cameras in order to cover the vertical distance.

It is undesirable to provide too many body restraints in order to prevent body sway during the measuring period, since these tend to cause an unnatural standing posture and are also likely to obscure some part of the surface being measured. If some body sway is to be tolerated then further projected lines of light are necessary in pairs opposing each other and more or less at right angles to each other, in order to resolve the sway component of the measurement into two orthogonal axes. Each television camera now gathers data from two projected lines of light while a further seven cameras view the other two projected lines of light. Having more than one projected line of light also improves the problem of the projecting or the re-entrant parts of the body obscuring the view of the line of light from the camera.

Each line of light is projected from four projectors, making sixteen projectors in all. This arrangement makes the design of the projectors easier but more important provides an extended light source allowing horizontal surfaces, such as shoulders, to be properly illuminated. The total system is shown in Figure 1.

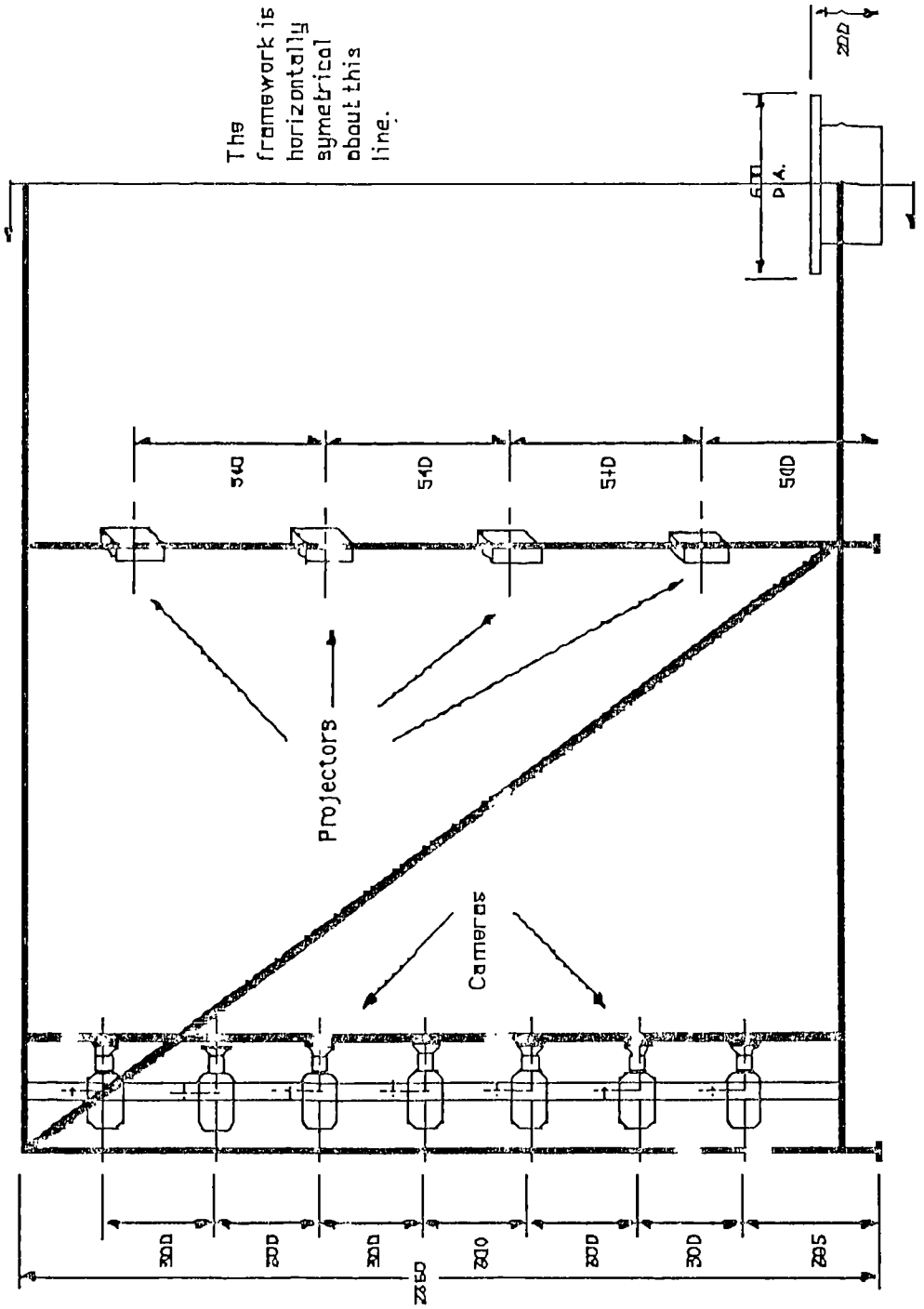
Projector design

A standard 35 mm slide projector was tried for the projection of the slit of light but it was found to be unsuitable because of its limited depth of focus. Since the depth of focus depends only on the diameter of the projection lens while the light output depends on the focal ratio (f Number.) it was decided that a projector could be made using a 16 mm f 1.4 lens. Several optical contractors were approached but all raised difficulties which were proved unfounded when we made the projectors ourselves. The calculations for the projector design are given

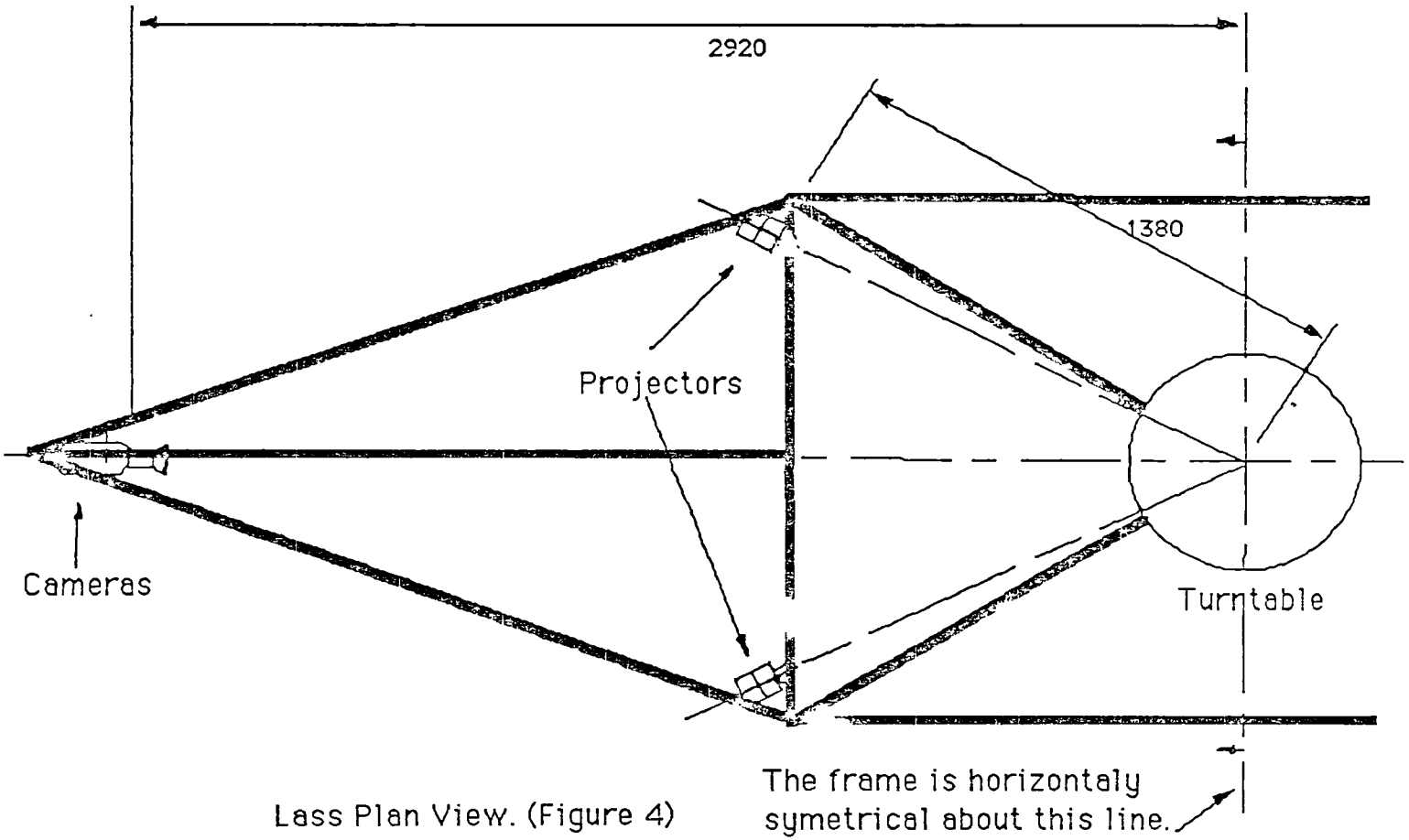
in Appendix 3. An additional requirement for the projector is that the light output should be controlled by an electrically operated shutter mechanism. This is necessary because with four banks of projectors the light from one pair of the banks shine directly into the opposite cameras. Using shutter control the cameras and projectors were synchronised to operate alternately. An effect of the choice of lens is that the projection system is rather like an inverted microscope, which projects a very much magnified image of the mechanical slit within the projector. This means that the slit had to be accurately made and free from dust. Because of its small size it was easy to make a shutter mechanism by attaching a simple flag to a standard electrical relay. The flag obscured the slit when the relay was not operated and cleared the slit when operated.

Mechanical Construction.

A framework was constructed using the Phillips Variable Building System (Maswell Engineering Ltd.,) in order to support the fourteen television cameras and sixteen light stripe projectors in the correct positions relative to the turntable. An important feature of the framework as illustrated in the elevation (Figure 3) and the plan drawings (Figure 4), is that each end is constructed using three tetrahedrons. the two ends are then joined using a simple rectangular construction. The tetrahedrons were the only configuration found which provided sufficient stiffness to the framework to maintain adequate camera and projector alignment. The whole construction is horizontally symmetrical about the centre of rotation of the turntable.



Lass Elevation. (Figure 3)



Lass Plan View. (Figure 4)

Camera Interface.

The video signals from the camera, if they are to be used for taking measurements, have to be converted to digital information. A television camera uses a raster scanning method in order to serially interrogate the picture present on the photo-sensitive layer within the camera. The cameras used for the LASS equipment are scanned with 625 lines / 50 fields / 25 frames per second. Each line must then take 1/(625*25) seconds to traverse across the screen, that is 64 micro-seconds.

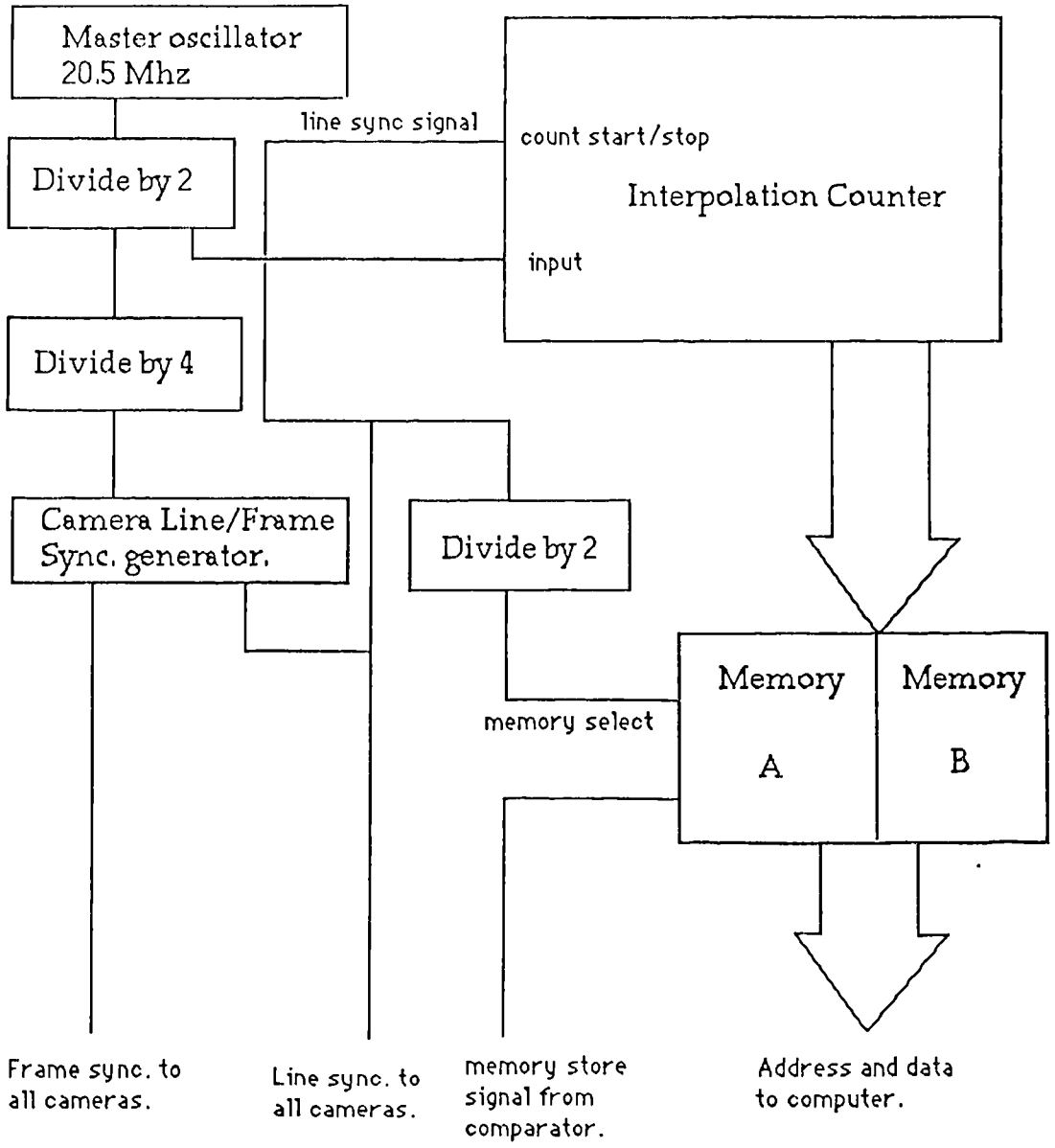


Figure. 5. Interface Block diagram

Referring to Figure 5. the timing is derived from a 20.5 MHz crystal oscillator which is electronically divided to provide standard line and frame synchronisation signals for all the cameras and an additional signal at 10.25 MHz which is used to drive the line interpolation counter. If we now consider a single line from a camera viewing a scene illuminated by a single line of light then we get the wave forms shown in Figure 6.

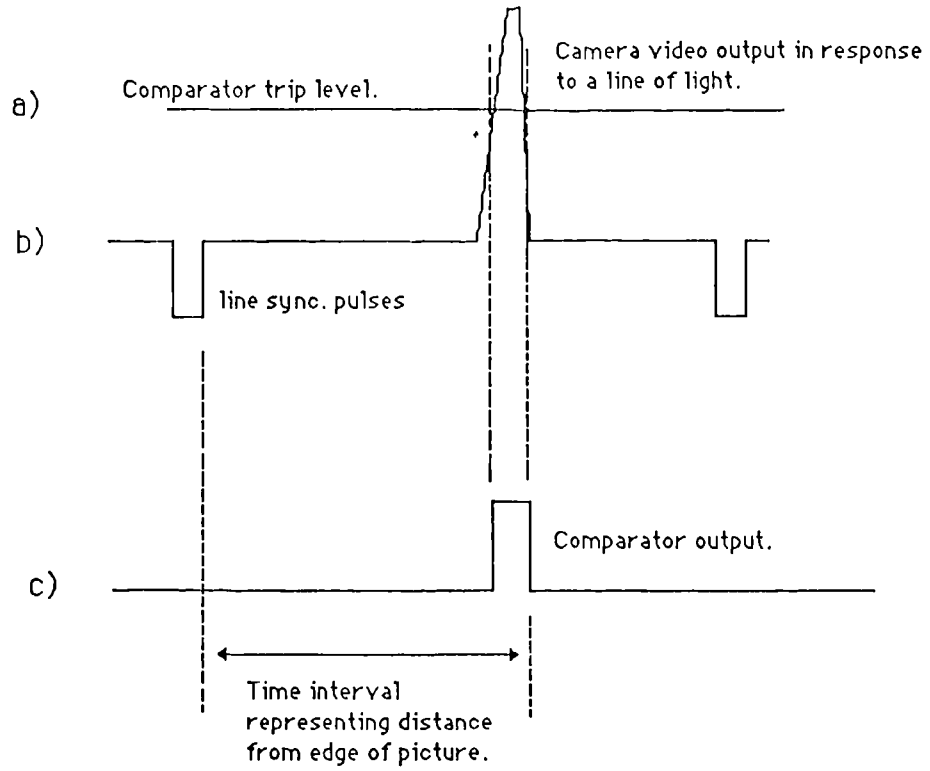


Figure 6. Wave forms from the Camera & Comparator

The wave form at b) in Figure 6 represents the voltage output of a single line of the camera scan plotted against time. This voltage is also applied to an electronic voltage comparator circuit which is designed to sense a particular voltage at its input and then produce an output which indicates whether the input of the comparator is above the threshold level or below it. By measuring the time interval from the trailing edge of the line synchronising pulse to the trailing edge of the comparator output pulse a measurement of the distance from the edge of the picture (as signalled by the line synchronising pulse) to the trailing edge of the line of light can be obtained. Time measurement of this interval is obtained by resetting to zero the line interpolation counter on the trailing edge of the line synchronising pulse, and storing the reading of the counter on the trailing edge of the comparator output pulse. The counter, in the meantime, counts the 10.25 MHz. signal derived from the master oscillator.

Each camera in the system has associated with it a separate comparator and a high speed memory circuit which can store up to 16 time intervals during one line scan interval (64 microseconds). The circuitry associated with the storage of the interpolating counter data, will store

a count whenever there is a change in the comparator output signal, so that both edges of the image from the strip of light are stored. It is also convenient, and more economical, to use one camera to record the data from two strips of light, which comprises data for four edges. Because of the finite rise time of the pulse from the camera, the exact position of the edge, as determined by the comparator, can change with the amount of reflected light available at the time. In order to minimise this effect, the computer selects only the data from the trailing edge of each light stripe, so that the errors are the same sign and magnitude for each measurement and can be compensated for as part of the body sway correction.

General Electronics

The electronics are housed in a number of major modules as follows :

1. **Eltec Computer** which is the heart of the system. The computer comprises standard computer, memory and interface cards and disk drive all fitted into a VME bus. The full specification is given in Appendix 1.
2. **Line Event Memory Unit** which has two memory cards for each camera as a buffer for the data until it can be read into the Eltec computer. This unit also houses the **Edge detection, Monitor and Synchroniser drive** modules.
3. **Motor Control Unit** houses the stepper motor drive for the turntable.
4. **Auxiliary Control Unit** provides an interface for the computer to switch on the projectors, operate the calibrator and drive the shutter control unit.
5. **Shutter Control Unit** which contains the field effect transistors (FET) drivers to drive the shutter relays.
6. **Projector Power Supply** is a switched mode supply to provide a regulated 12 volts to drive the projector lamps.

Circuit Description

All the television cameras are synchronised together from a master oscillator. Driven from the same master oscillator is a Line Event Counter which starts counting from zero at the beginning of each

television line scan. The video signal from each camera is monitored by a comparator circuit, one for each camera, which detects when the video signal exceeds a pre-set threshold level. This level will normally be exceeded only when the camera senses the image of the line of light projected onto the body being measured. When this happens the signal from the comparator causes the count of the Line Event Counter to be retained in a memory which is arranged to occur when the comparator detects either the leading edge of the light image or again on the trailing edge.

The memory is arranged to store up to sixteen separate counts so that multiple edges can be detected during one line scan. The contents of these memories have to be transferred to the main computer before they are likely to be overwritten during the next line scan. To allow time for the data to be transferred from all fourteen cameras a duplicate set of memories is provided for each camera. While one set of memories is being filled during a particular line scan the second memory, which stores the data from the previous line, is being read into the main computer.

This sequence of events is repeated for each line until all the data for a complete television frame, from all the cameras, are transferred into the main computer.

The circuit naturally divides into several functional blocks as shown in Figure LASS01(Appendix 4).These will be described separately.

Master oscillator and synchronisation

The circuit diagram is shown in Figure LASS04(Appendix 4). A crystal oscillator is the master time reference made from a 20.5 MHz quartz crystal and the two inverters in integrated circuit (IC) 1. The first stage of a four stage binary counter in IC 2 divides the 20.5 MHz signal by two to feed the Line Interpolation Counter with a 10.25 MHz signal. Further division in IC2 produces a 2.5625 MHz signal into IC4. which is a specialised integrated circuit designed to provide television line and field synchronising signals. The line synchronising signal, as well as being buffered by IC 7 also provides a reset signal to the Line Event Counter (IC's 8,9,10). This counter is then reset at the beginning of each television line and provides an accurate time reference throughout the

duration of the line, each count representing 48.78 nano seconds. To identify each line of the television scan, the line synchronising signal is counted by the counter comprising IC 11, IC 12 and IC 13 which are reset by the field synchronising signal. The D type flip flop IC 3/b divides the line synchronising signal by two to provide two control signals **xram** and **yram** which select the memories for alternate lines. Because standard television cameras operate with alternate fields having lines interlaced between the lines of the previous field to give the illusion of 625 lines per field instead of 312.5 actual lines, a field status signal is latched into IC 3/a so that, if necessary, the computer can identify the even or odd field.

Comparator

The comparator circuits for all the cameras are contained on one printed circuit card as shown on the circuit Figure LASS02 (Appendix 4). The video signal is D.C. restored for the comparator circuit contained in IC 1 by the series capacitor C1 and the diode D1. All the comparator circuits have a common reference level from the voltage follower circuit IC 17 which buffers the variable output from the potentiometer PST 1.

Line Event Memory Unit.

This unit is situated on each of the fourteen camera interface cards. For the circuit diagram refer to LASS 16 (Appendix 4) and for the System Logic and Timing diagram refer to LASS 31 (Appendix 4). The Latch Data signal serves as a system clock and the rising edge of the wave form latches the comparator signal into the D type flip flop in 1/2 of IC 8 . Because the Latch Data signal is inverted by the exclusive or gate the second D type flip flop latches the data 1/2 cycle later. When the two flip flop signals are combined in the exclusive or gate IC 9 the result is a pulse one clock cycle long which coincides with either a rising edge or a falling edge of the comparator output. This pulse is combined with the **xram** or **yram** signal in the and gate IC 10 to differentiate between an odd or even line count. If the appropriate **xram** or **yram** signal is present the pulse is used to latch the data from the Line Interpolation count into the memory in IC1, IC 2 and IC3 at an address decided by the

address counter IC 7. which after a small delay due to the series of and gates in IC 7 is incremented to the next address. The memory now contains the Line Interpolation count which is a measure of the distance of a light stripe from a position corresponding to the edge of the television field of view. During the linescan when data are not being stored i.e., when the alternative memory is in use, data can be read from the memory into the main computer by asserting an address onto the pre-set input terminals of IC 7 while also asserting the pre-set enable signal. The data are then present on the data return terminals ready to be read by the main computer parallel interface. The address counter IC 7 is reset to zero by the pulse generated by IC 6 in response to an xram or yram signal ready for a new data capture cycle..

Sync Drive & Video Generator

Referring to Figure LASS 07 (Appendix 4). The composite synchronising signal from pin 6 of the Master Oscillator and Synchronisation circuit appears on pin 29 of the x16 Sync Drive & Digital Video Generator Board where it is inverted by IC1 and then its current drive capability is considerably increased by F1 and the sixteen output driver circuits Tr1-Tr16 to a level to drive the 75 ohm coaxial cable feeding each camera. An additional circuit comprising TR17 and three inverter gates from IC1 combines the video signal with the composite sync. signal and provides sufficient current to drive the coaxial cable to the video monitor.

Auxiliary Control Unit, Shutter Control Unit and Projector Power Supply

These circuits will not be described in detail as their operation is obvious from their circuit diagrams in Appendix 4.

CHAPTER 3

The Digitizer Computer Hardware, Software and Digitizer Calibration.

Digitizer Computer Programs

It is a feature of the scanner design that as many of the data capture techniques as possible should be carried out by the general purpose computer used to drive the scanner. This decision means that the camera data must be read into the computer memory, in real time, as fast as they are generated. In order to allow sufficient time for the data to be transferred to the computer the interface electronics was made so that the data accumulated duri ıg one line of the television scan, is stored in a local memory associated with the particular camera as described above. While this is taking place the data from the previous line, which is stored in a different local memory, can be read into the co.nputer. Each of the fourteen cameras has t' ese two temporary memories included in the camera interface electronics. The time taken for one television line to scan across the screen is 64 microseconds. During this time interval the data from all fourteen cameras must be read into the computer. If the computer is not ready to take it in when available, then the data will be lost. Unfortunately most computer operating systems work on an interrupt basis for completing essential housekeeping tasks such as updating the screen or refreshing the dynamic memory. If such an interrupt were to occur during a data capture period then that data would be lost.

To avoid this problem the OS9 operating system was used because, with it, it is possible to turn the operating system off during critical parts of the program.

Computer hardware.

The computer chosen was one made up from standard computer cards plugged into a VMEbus card frame (See Appendix 1 for complete specification.). This approach was chosen because it allowed for flexibility as the project developed and also it was possible to have

2 x 24 I/O ports on a parallel interface (APAL) card for connection to the camera interface cards.

Working in real time.

The computer is arranged so that the operating system is protected against interference from the user's program. This is achieved by special address lines to the memory which switch between "system state" and "user state". It is intended that normal user's programs shall run in "user state" and should they malfunction they will not be able to corrupt the memory of the "system state" which contains the operating system. With the OS9 operating system it is possible to write program modules in 68000 machine code language Leventhal,, Hawkins, Kane and Cramer (1982) which are part of the operating system and reside in the "system state" part of the computer memory. These modules can be accessed from a normal assembly language program using a "trap call" instruction. Because the modules are operating as "system state" programs they are permitted access to instructions which prevent the 68000 micro-processor from responding to interrupt signals, thus preventing the operating system from generating interrupts which could cause loss of data from the cameras. An assembly language version of the "system state" program for collecting data from the cameras as well as a calling program is given in Appendix 2.

Calibration

The calibration of the LASS scanner is a complex and vital part of the system design because of the multiplicity of cameras and projectors, which must be accurately aligned and their positions in space accurately determined. It is a function of the calibration method to relate the co-ordinate measuring space of each individual camera to the co-ordinate system of the real world (the co-ordinate system used for every day measurements where vertical is defined as the opposite direction to the pull of gravity). A novel method is used which provides for a large part of the determination of camera positions and projector angles to be automatically resolved under computer control.

The Calibration Method

The method to be described is intended to provide a correction for minor lens distortions, minor camera misalignment and also to relate the camera output to the machine measuring space in the real world In order to achieve this it is necessary to relate world measurements to corresponding measurements on the television screen.

All the real world measurements are related to a reference plane, which is a plane which lies along the centre of rotation of the turn-table and which is at right angles to the line of sight through the camera lens. Practically, this is in the form of a substantially supported panel (Figure 7) which carries an array of light emitting diodes (LED) which produce a pattern of light on the television screen when viewed by the television camera. This panel (the calibrator) has a circular base the same diameter as the turn-table and has three levelling screws spaced at 120 degrees intervals around the edge. So that the calibrator can easily be removed from or replaced on the turn-table without too much readjustment, the three screws fit on to three different pads on the turn-table. The first pad has a conical recess which gives precise location of the screw resting in it. The second pad has a V grove which locates the second screw at a precise angular position relative to the first screw but without adding other conflicting restraints. The third pad has a plain surface which provides a horizontal reference only. Using this fairly well known mechanical technique, the calibrator sits solidly on the turn-table and always returns to the same position if mechanically disturbed.

Alignment of the reference plane along the axis of rotation of the turn-table is measured using a simple resistive potentiometer type of transducer mounted on a bearing at the top and on the centre line of the calibrator. The other end of the transducer is connected to the general machine framework. In principal if the reference plane is correctly aligned along the turn-table axis, then when the turn-table is rotated, the transducer will continue to be unchanged. Any misalignment of the reference plane axis will show up as a sinusoidal error, which can be corrected by means of the three levelling screws on the calibrator base.

The array of LED's on the calibrator are arranged to suit the calibration rectangles on the television screen. The screen is arbitrarily divided into thirty rectangular areas arranged in a 5 x 6 array as shown in Figure 8.

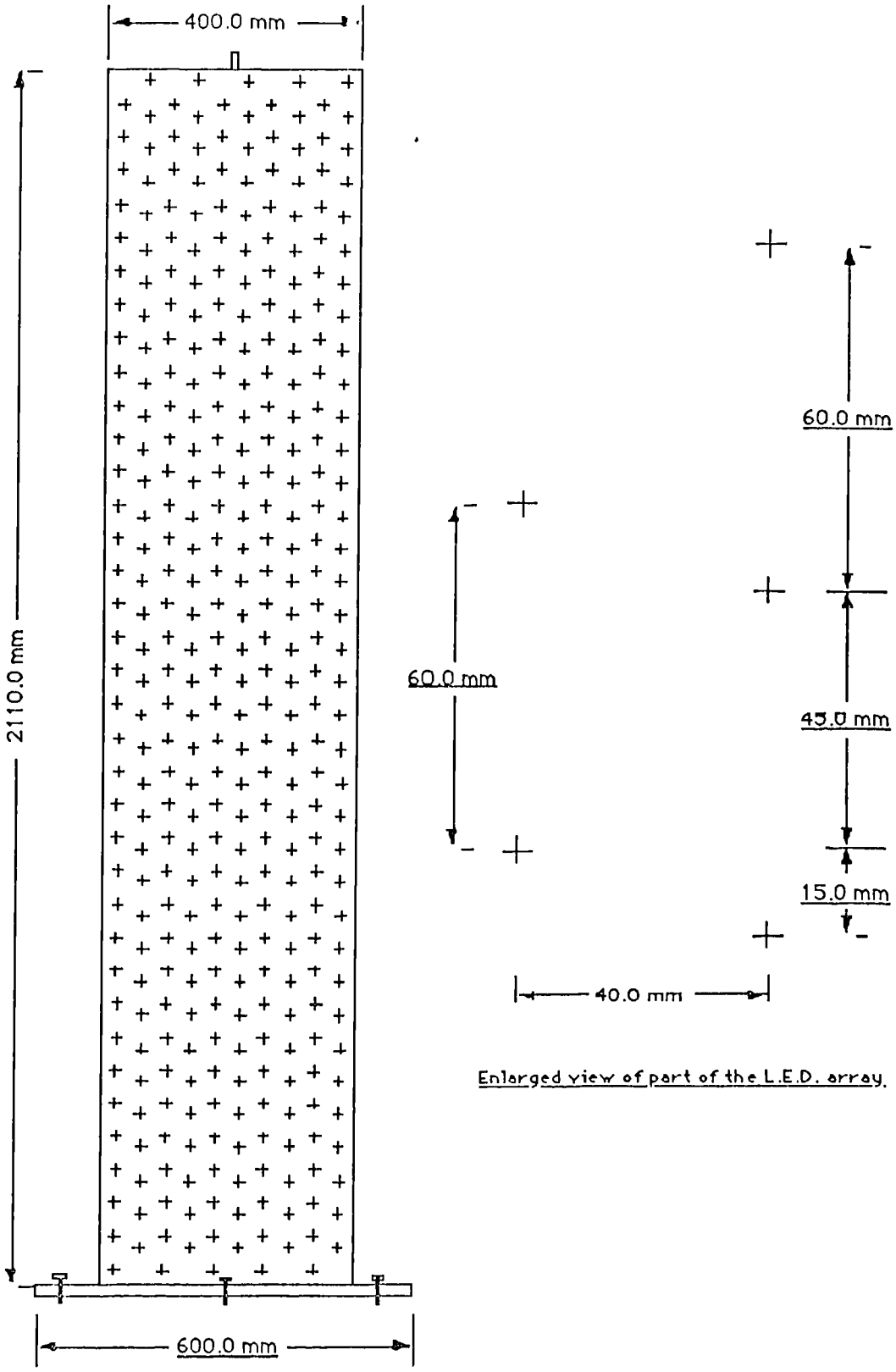


Figure. 7. Diagrammatic representation of the calibrator

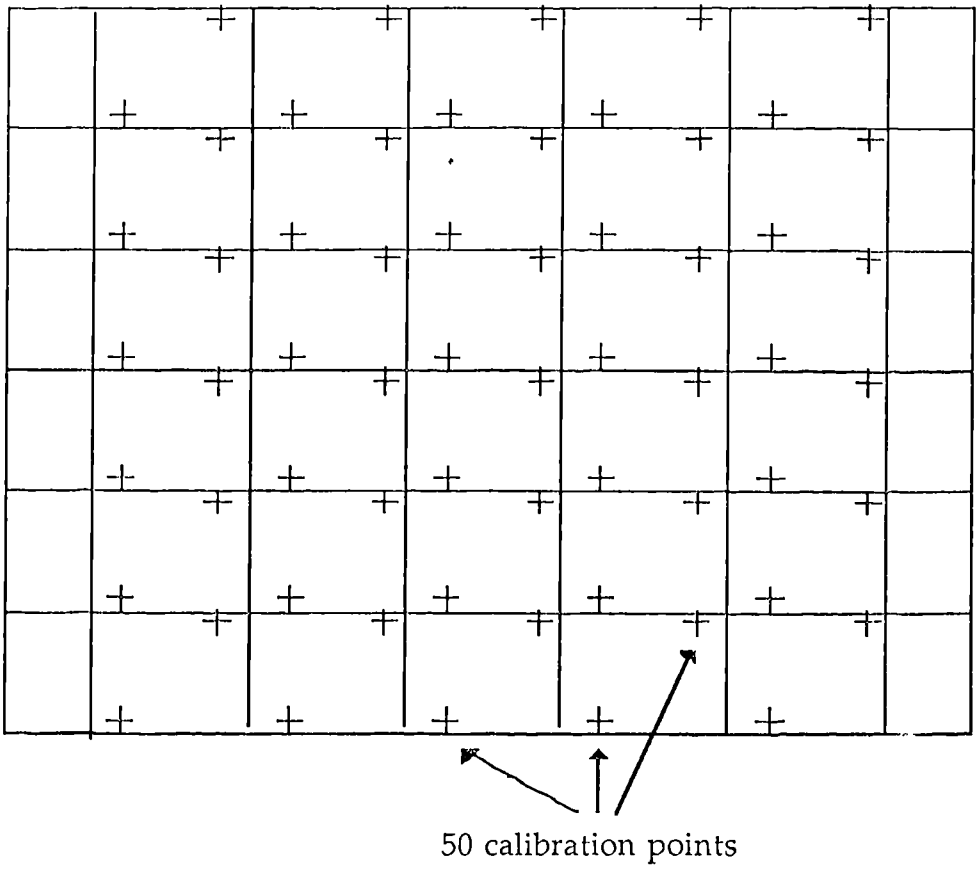


Figure. 8. Calibration Window

Both the right and left hand edges of the screen are ignored since in this instance they are not used and therefore do not need to be included in the calibration. The pattern of lights on the calibrator are arranged in columns 40.0 mm apart and staggered in a vertical direction by 45.0 mm, so that the camera, when it is in reasonable alignment, sees them as an array with two lights in each calibration rectangle. Referring to Figure 8 it can be seen that, for each calibration rectangle, four parameters can be determined, which are the x & y co-ordinates of the origin of each rectangle, and the x & y slopes within the rectangle. This information is stored within the computer to be used later to accurately determine the world co-ordinates P(x,y) from the local co-ordinates P(r,s) of any point P seen by the camera. The relationship of P(x,y) to P(r,s) is given as follows.

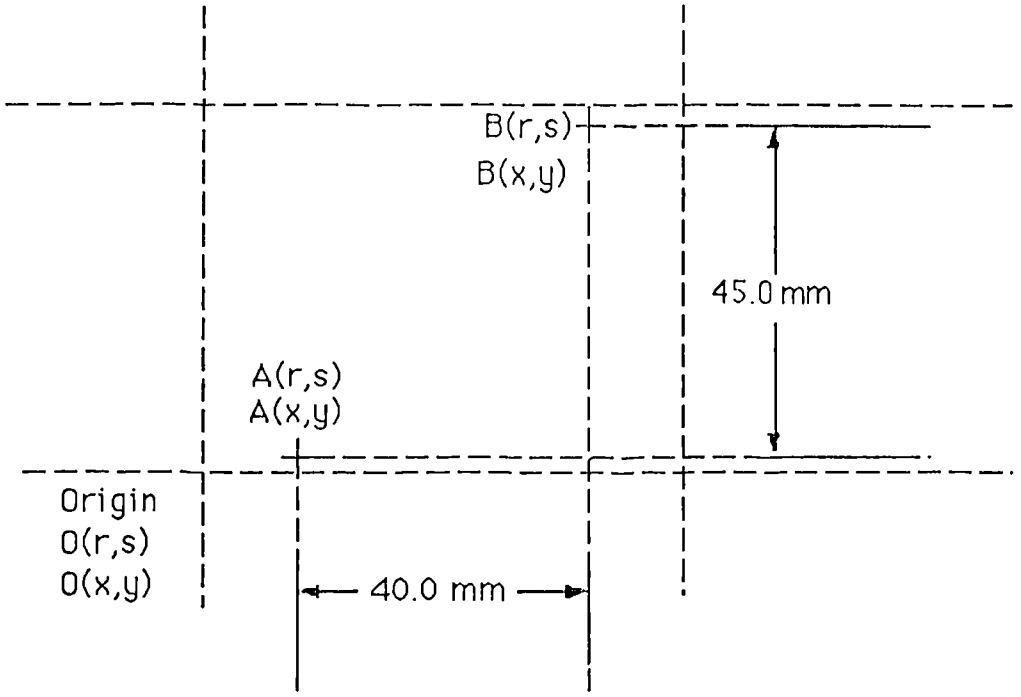


Figure. 9. Detail of calibration screen display

O(r,s) = origin of calibration square in local co-ordinates.
O(x,y) = origin of calibration square in world co-ordinates.
A(r,s) = position of reference point A in local co-ordinates.
A(x,y) = position of reference point A in world co-ordinates.
B(r,s) = position of reference point B in local co-ordinates.
B(x,y) = position of reference point B in world co-ordinates.

Let there be a factor F(x,y) to relate local and world co-ordinates such that F times local co-ordinates equals world co-ordinates.
Then

$$Fx\,(Br - Ar) = 40$$
$$Fy\,(Bs - As) = 45$$
$$Ox = Ax - Ar*Fx$$
$$Oy = Ay - As*Fy$$

Let Px = the x world co-ordinate of any point P in the calibration square.
& Py = the y world co-ordinate of any point P in the calibration square.
& Pr = the local horizontal co-ordinate of any point P in the calibration square.

& Ps = the local vertical co-ordinate of any point P in the calibration square.

Then

Px = (Pr - Or)*Fx + Ox .

& Py = (Ps - Os)*Fy + Oy

This method automatically adjusts the individual camera data to that required for one single array of data in world space and covering an area of 400.0 mm x 2100 mm.

The calibration process described above is controlled entirely by the computer which controls the lights, to switch them column by column until the calibration is complete.

Testing the accuracy of measurement

The accuracy of measurement was tested using a series of six wooden cylinders. The diameters of the cylinders ranged from 100 mm to 600 mm in steps of 100 mm. Each cylinder was 50 mm thick and arranged one above the other with the largest at the bottom and the smallest at the top. Tubular supports were made to support the six cylinders at thirteen heights within the 2.1 m height of the LASS measuring system. Taken as a whole the mean error in measuring a diameter between 100 and 600 mm at heights ranging up to 2.1 m was -0.6 mm with a SEM of 0.3 mm.

Vertical heights were checked by measuring the edge of the largest cylinder when positioned at different heights above the ground. The actual heights were measured using a 2 m steel ruler. Every measurement of vertical height gave a constant 3mm. error. It should be remembered that the vertical resolution of the system is 5mm., so any value within 5mm. can be expected.

Correction for body sway in the LASS system.

When a person is being measured it is necessary for them to stand in as natural a posture as possible if the measured result is not going to be distorted. The amount of sway measured on a free standing person West (1987) is approximately 15 mm measured at the shoulder. If no method is used to compensate for this amount then it will directly contribute to the errors in measuring a radius . On the LASS

equipment this problem has been solved in a very simple but novel fashion. For elimination of sway in the lower part of the body a telescopic pole is used to provide support between the subjects legs. This provides a simple reference point which virtually eliminates movement of the pelvis without interfering with a natural posture.

To remove the effects of movement of the upper part of the body the LASS system uses four sets of projectors which project, on to the body, four lines of light. In effect, at the time every measurement is taken, four measurements are taken simultaneously. The angles of the light beams are at 60 degrees, 120 degrees, 240 degrees and 300 degrees, chosen to optimise the ability to "see" re-entrant parts of the body rather than for sway compensation. The 240 degree angle is however exactly opposite the 60 degree angle and the 300 degree angle is exactly opposite the 120 degree angle for the good reason that any movement along one of the angles produces an equal and opposite movement along the reciprocal angle. If the object being measured was a perfect cylinder an exact compensation method could be calculated, but as a body is far from being a cylinder the best that can be said is that the measurement of the distance through the body is reasonably constant in spite of the amount of sway remaining. The way that the radius at any point is calculated is that the four measurements are resolved into polar co-ordinates; like angles are collected together so that over the period of one revolution of the turntable each measuring angle, (2.4 degrees on high resolution measurements), receives four measurements. These are then averaged to give the final radius measurement for that angle. In practice this method, although not a mathematically perfect method of correcting for sway, gives excellent results as shown by the overall machine performance given in Chapter 8 and is vital for the successful measurement of unsupported bodies.

The "snapy" program

The aim of the program, now called "newsnapy" for the latest version, is to provide all of the computer control for the LASS body scanner and sufficient extra facilities to check that the system is working correctly. Some extra subroutines are included to provide print out of cross sections and of data files. In use the program is controlled from a two

page menu which is displayed if y or z is typed after the prompt. If y is typed page 1 is displayed and page 2 for z. Where the menu says "print" the data is listed on the monitor screen and the printer is driven from the monitor. The function key F1 switches on the output to the printer and the function key F2 switches it off again. Details of compilation of this program are given in Appendix 5.

Menu used by the newsnapy program running on the Eltec computer.

page1

0 = Not used.
1 = List one frame.
2 = Read mean slice.
3 = Plot cartesian.
4 = Flat plot 1 frame & clear.
5 = Flat plot 1 frame without clear.
6 = Select table control.
7 = List directory of /h0/newdata.
8 = Print array.
9 = Print multiple slices.
a = Plot front view.
b = Plot slice & clear.
c = Plot slice without clear.
d = Print calibration.
e = Use wand.
f = Toggle correction flag.
g = Collect data & process later.
h = Print slice data to a file.
i = Wand print.
j = Save data file.
k = Load data file.
l = Not used.

page2

m = Set height.
n = Select camera.
o = Select no. of cameras in the system.
p = Control calibration lights.

q = Enter shadow angle.
r = Toggle arm flag.
s = Save data to a text file.
t = List data.
u = Toggle the projector.
v = Set zero.
w = Toggle average flag.
x = Plot on printer.

1. List one frame

Takes a snapshot and lists the numbers in the file as a result. The results are in four columns. The first column gives height at five or ten mm. intervals depending on the resolution, the second the sum of the radial distances as seen by the camera, the third the number of times the particular distance has been measured, and the last column gives column three divided by column two to give the average position. The multiple measures of distance for given point are due to overlapping fields of view of the cameras and to the same point being recorded by more than one TV line scan. Used for diagnostic purposes in the early stages of development.

2. Read mean slice

Gives the same result as 1) above but also calculates the mean radius. Used for diagnostic purposes in the early stages of development.

3. Plot cartesian

Plots on the graphics screen both a front and a side view of the data which can be scaled to any size. The height of the view shown on the screen can also be selected.

4. Flat plot 1 frame & clear

Plots on the screen the radial line as seen by the cameras. Used for diagnostic purposes in the early stages of development.

5. Flat plot 1 frame without clear

As 4) above but without clearing the screen from the previous display.

6. Select table control

Brings up a separate menu to allow for control of the turntable from the keyboard.

7. List directory of /h0/newdata

Lists on the monitor screen the contents of the "newdata" directory as an aid to selecting files to display.

8. Print array

Prints the contents of the data array in the format of 1) above.

9. Print multiple slices

Prompts for slice numbers and then both lists and plots the vertical slice data between the slices specified.

a. Plot front view

Plots on the graphics screen a view of the data as a series of dots, one for each measured point. The field of view is limited to 300 mm. but a mouse selected menu allows for the field of view to be moved up and down as well as around the object represented by the data. In addition, by operating the mouse key when the cursor is at a required height, the horizontal cross section is displayed.

b. Plot slice & clear

Clears the graphics screen and plots the horizontal cross section at the set height.

c. Plot slice without clear

As in b) above but without clearing the screen. Enables multiple plots to be made.

d. Print calibration

Lists on the monitor screen the calibration data for the selected camera.

e. Use wand

Enables the program which allows the light wand to input position data.

f. Toggle correction flag

The correction flag decides whether the correction derived from calibration shall be applied.

g. Collect data & process later

Initiates the start of a scan.

h. Print slice data to a file

The program prompts for a file name, the height of the top and bottom slices and the slice spacing. The file first line starts with the number of angular intervals followed by the number of slices. Each slice record has a first line giving the slice height in mm then subsequent lines give the x and y co-ordinates of the raw data points. Each point has a constant offset added to bring the plot to the centre of the screen.

i. Wand print

Prints the co-ordinates of the points pointed to by the light wand.

j. Save data file

Saves the entire raw data file.

k. Load data file

Loads the entire raw data file.

m. Set height

Allows input of the height of the display window or slice from the keyboard.

n. Select camera

Enables the particular camera to be selected via the keyboard for calibration print out or single frame print.

o. Select no. of cameras in the system

Chooses the particular camera list. If a small object is being scanned it is quicker to chose a smaller number of cameras.

p. Control calibration lights

When the calibrator column is in use the lights can be manually operated from the keyboard for test purposes.

q. Enter shadow angle

During the calibration process the angle of the projector lights is automatically determined. If it is required to change these numbers manually it can be done here.

r. Toggle arm flag

When cross-sections are plotted it is possible to invoke a simple algorithm to automatically remove the arms. This is controlled by this flag.

s. Save data to a text file

Prompts for file name, top and bottom heights and produces an ASCII file of radius, angle and height for the raw data.

t. List data

Lists the data as radius, angle and height between top and bottom limits.

u. Toggle the projector

Switches the projectors off if they are on or on if they are off.

v. Set zero

Allows adjustment of the x axis zero offset for each camera.

w. Toggle average flag

Obsolete item. Currently serves no purpose.

x. Plot on printer

Produces a pseudo graph plot on an Imagewriter printer using tabs and printer characters.

The following photographs show the computer screen displaying the raw data in the various display modes.

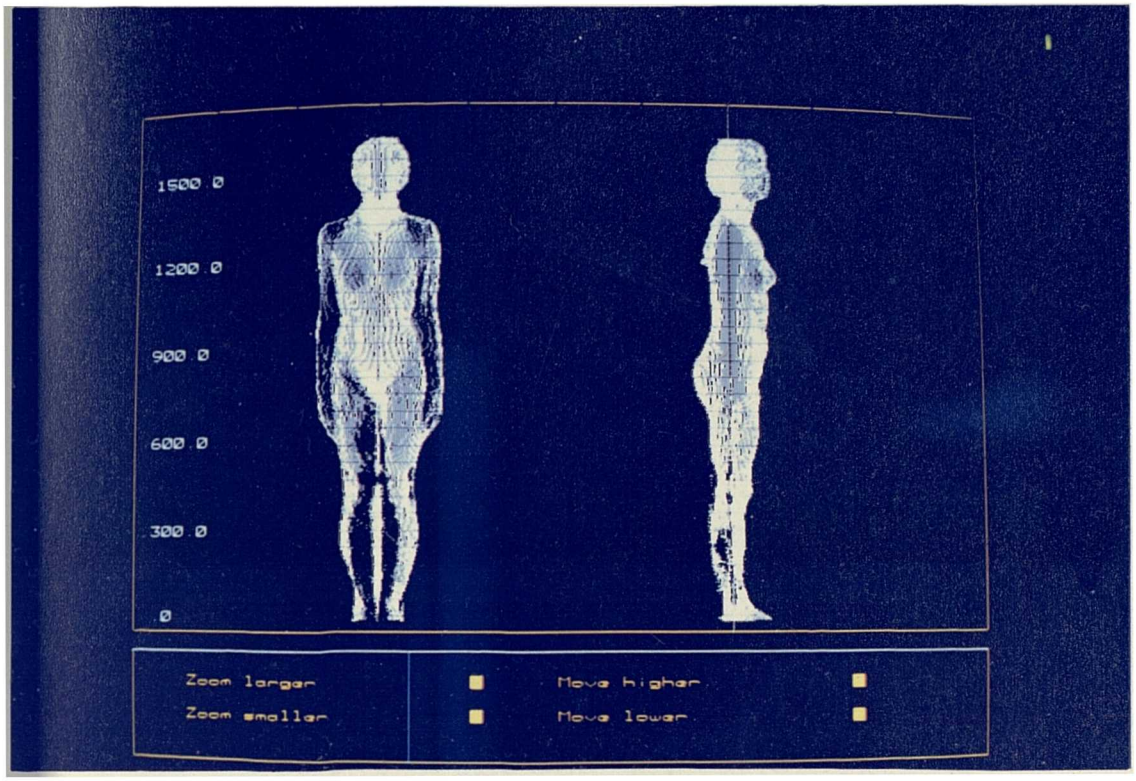


Figure. 10. Appearance of the screen showing a cartesian plot

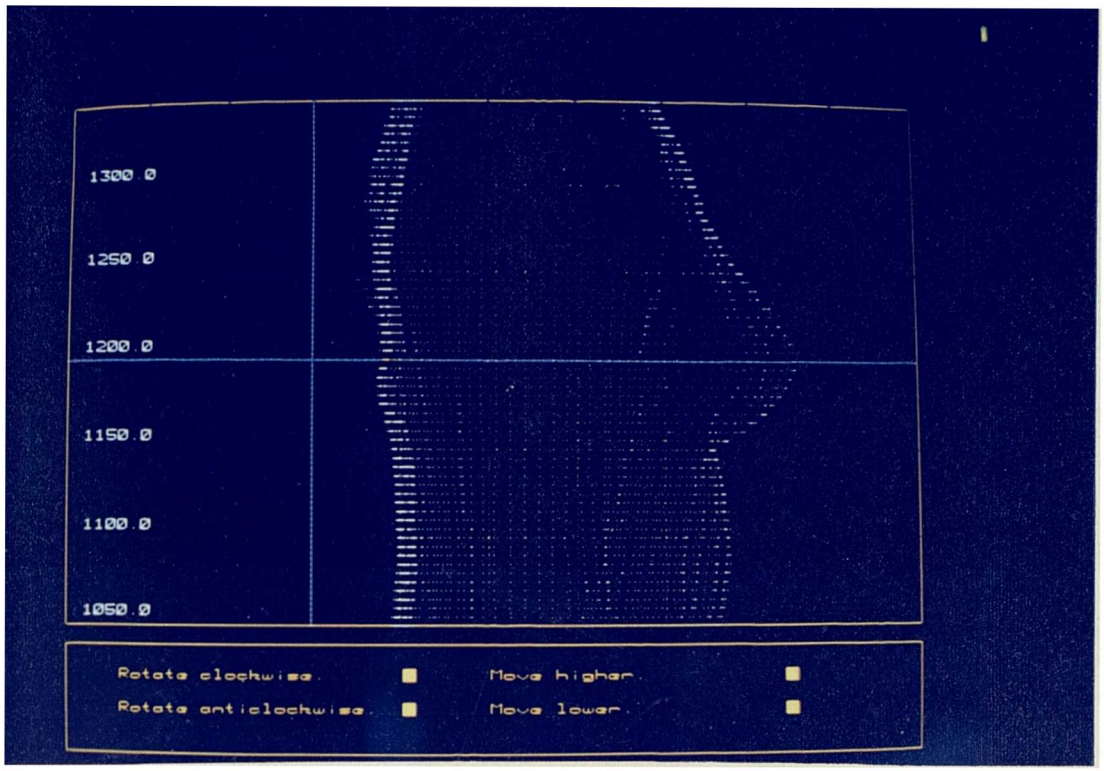


Figure. 11. Appearance of the screen showing a side view plot

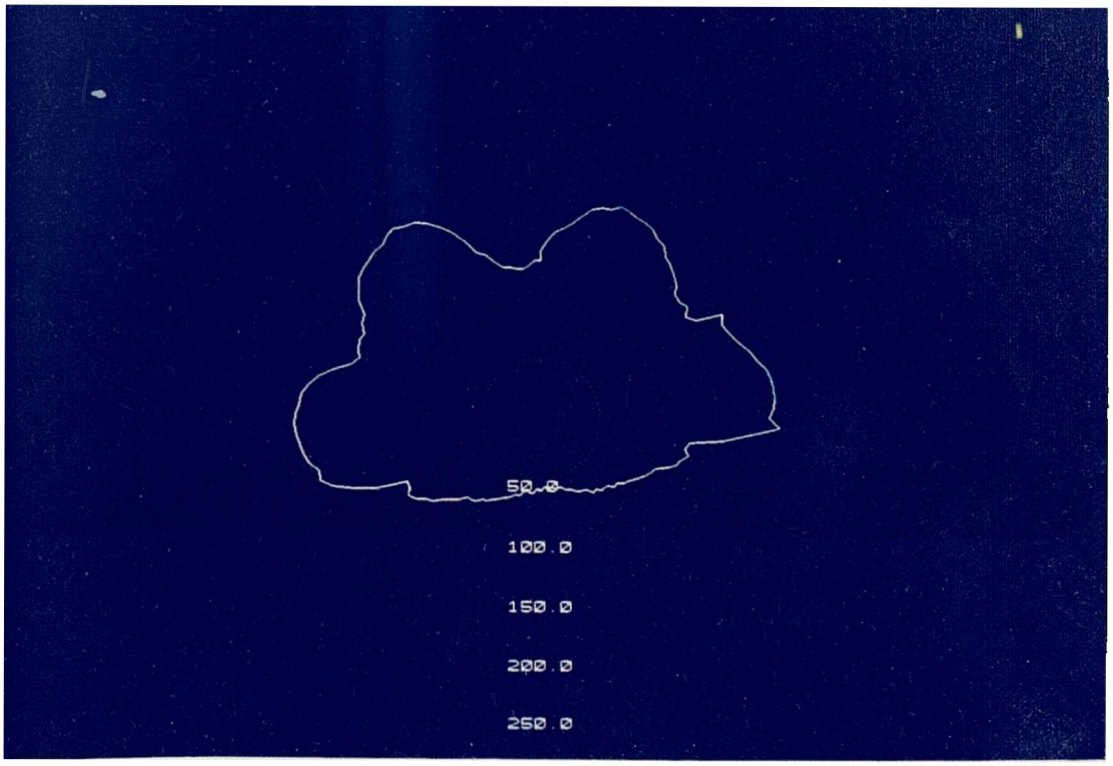


Figure. 12. Appearance of the screen showing a cross-section

CHAPTER 4

The edit/interrogation program

The data from the LASS scanner are of little use in their raw state. If printed out in x,y,z format they would occupy approximately thirty pages of A4 paper, which is too much data for manual manipulation. To make use of the data the Body Edit and Display (BED) program has been written.

This program provides the following :-

1. Visual display of the raw data when translated or rotated in any direction.
2. Simple linear measurement of the raw data.
3. Facilities to allow automatic and manual fitting of Cardinal splines to body sections .
4. Provision to combine the Cardinal splines into three dimensional surface patches.
5. Linear measurement of the body defined as a series of surface patches as well as surface distance or tape measure distance around the whole or part of the perimeter of any horizontal or inclined plane through the body.
6. Provision to load or save raw data to or from the disk memory.
7. Provision to load or save spline data to or from the disk memory.
8. Calculation of surface area between any two horizontal sections.
9. Calculation of volume between any two horizontal sections.
10. Provision to display spline data as a wire frame figure.
11. Provision to display spline data as a 'skinned' figure.
12. Ability to re-create a raw data file from edited spline data.

The computer program to achieve the above is written entirely in the C programming language as defined in Kerninghan and Ritchie 1978.

When the program is run the computer screen is filled by a large window with four smaller windows situated around the edges of the large window. In addition there is a calibrated bar along the bottom of the screen which allows a variable input via the cursor and the mouse. The large main screen displays the data in graphical form while the smaller screens provide for interactive input to change the program or to give information about the data. The window on the left of the screen is arranged to represent a keyboard where the "keys" can be operated by moving the cursor using the mouse to be within the area of a particular key and then clicking the left mouse button. Above the keyboard window in the top left corner of the screen is a small data window which displays the current situation with regard to rotation and translation of the displayed data. It also displays the situation at the time that the "set zero" key was operated and also the difference between the current situation and that at the time when the "set zero" key was operated. This allows simple linear measurements to be made on the displayed data.

In the top right hand corner of the screen the third small window displays cross section information. If the centre button of the mouse is held down while the cursor is moved up the screen an horizontal cross section of any part of the displayed data will appear in the cross section window.

Because the displayed cross section moves with either rotation or translation of the main data display it can be a great help in accurately positioning the main display.

Beneath the cross section window on the right hand side of the screen is a fourth window which serves a dual purpose. When not displaying data it displays a vertical section through the z axis when x is zero. The direction of the axes is displayed by an indicator in the middle of the screen.

Program Options

The "keys" which are used to interact with the program are listed below in order from the top of the keyboard.

Pclear clears the memory array which stores the surface patch data.

Pload loads patch data from the disk.

Psave saves the patch data to the disk.

F/B toggles the display to show either the front view or the back view.

Setzero stores the rotation or translation data in the top left window for use as described above.

Arms removes the arms from the raw data by looking for a sudden change in radius which signals the start or finish of an arm.

Marks displays points on the raw data display which have been pointed to, at the time of data capture by the "wand".

Ssave saves to disk a file containing cubic spline slice data for a number of slices between the limits set by **Set1** and **Set2** and at the spacing input into the program when requested.

Set1 stores the height of the cursor when this key is operated in order to define the lower limit for **Ssave** and **Oblique** and **Vol.**

Set2 stores the height of the cursor when this key is operated in order to define the upper limit for **Ssave** and **Oblique** and **Vol.**

Y1set sets the minimum height of the data displayed on the screen in order to speed up the calculation of the display when rotating etc., The level is set by moving the cursor along the slider displayed at the bottom of the screen after **Y1set** has been activated. Pressing the left mouse button disables the slider input.

Y2set sets the maximum height of the data displayed on the screen in order to speed up the calculation of the display when rotating etc. The level is set by moving the cursor along the slider displayed at the bottom of the screen after **Y2set** has been activated. Pressing the left mouse button disables the slider input.

X1set The raw data are plotted as a series of vertical stripes which represent a line through the radii at a particular angle measured at the centre of rotation of the body being measured. In order to make a clear picture only those vertical stripes that lie within the front 180 degrees are plotted. In other words, only the front is displayed, the back is removed. The angles which are displayed can be adjusted by moving the cursor along the slider after **X1set** has been activated. **X1set** adjusts the angle from which the plot starts.

X2set adjusts the angle from which the plot finishes

Setdy allows the number of slices to be entered so that when fitting splines to slices the correct slice spacing is automatically calculated.

Lines Returns the plot to be a line diagram where vertical strips are plotted.

Skin Assuming that the raw data have been converted to surface patches, then operation of this key, produces a plot which has the appearance of normal white skin.

Transx

Transy

Transz translates the picture by an amount set by moving the cursor along the slider.

Rotx

Roty

Rotz rotates the picture by an amount set by moving the cursor along the slider.

File retrieves the raw data file from the disk.

Curve changes the window to display cross sections and also changes the "key" menu so that cubic splines can be fitted to selected cross sections.

Zoom alters the scale of the plotted picture. Controlled by moving the cursor along the slider.

Lpp plots the patch data in the form of a wire frame.

Quit exits from the program.

Reset sets the rotation and translation parameters to zero.

When the **Curve** key is operated the main window is organised to display cross sections and the key window is shown with a different set of keys as follows.

Patch converts the cubic spline fitted slice data into surface patch data which are stored in a memory array.

Tglzro sets the slice counter to zero.

Smes changes the display of the already selected cross section so that surface distance and tape measure distances can be measured between two pointers. The position of the pointers are controlled by the keys **Dat 1** and **Dat 2**

Dat 1 sets the position of the first pointer for **Smes** as above. The position is set by moving the cursor along the slider.

Dat 2 sets the position of the second pointer for **Smes** as above. The position is set by moving the cursor along the slider.

Vol. Calculates the volume of the curve fitted body between the levels set by **Set1 and Set2.**

Oblique displays an oblique slice of a curve fitted body on the screen which is then available for making surface distance and tape measurements using the **Dat 1** and **Dat 2** keys.

Lines Returns the plot to be a line diagram where vertical strips are plotted. Can be used to exit from the **Oblique** program.

Spline automatically overlays the raw data slice displayed on the screen with a previously cubic spline fitted slice. Some of the measurement information in the vertical slice window is also recalculated as the slices are changed. Slices are selected by moving the cursor up the screen and at the same time holding down the middle mouse button.

Roty rotates the raw data cross section by an amount set by moving the cursor along the slider. The cubic spline section is not moved.

Transx

Transy translates the raw data cross section by an amount set by moving the cursor along the slider. The cubic spline section is not moved.

Recall loads the **standard data format** file from the disk. The **standard data format** is a format used for storing cubic spline data and will be described later.

Save saves the **standard data format** file on the disk.

Zflg toggles zflg between 0 or 1. If zflg equals one then extra columns are included in the **standard data format** file which contain z axis data.

Mirror toggles a flag which decides whether the mirror image of the cubic spline curve is plotted on the screen.

Cntup increments the counter which decides where in the array cubic spline fitted slice data is stored.

Cntdn decrements the counter which decides where in the array cubic spline fitted slice data is stored.

Setu allows the parameter u to be varied by moving the cursor along the slider. The value of u decides the size of the slice which is then created from the master file. The height of the slice is set by the counter controlled by **Cntup** and **Cntdn.**

Mload loads the master file from the disk.

Shocnt displays the value of the counter.

Savtcnt stores the counter value in a temporary memory.

Getcnt retrieves the counter value from the temporary memory.

Tape toggles a flag which decides whether the tape measure calculation is to be calculated for each slice. If the tape reading is not required then the computer works faster.

Tcirc calculates the circumference for each line of the **standard data format** array. If this is followed by a **Save** then the file will have the circumferences appended as the last column.

Redo will re-create a raw data file from edited spline data.

Iris returns the program to the beginning to display raw data.

Pclear clears the array which contains patch data.

Keys changes the key display to an alternative set of keys.

Xyzero removes the effect of **Transx** and **Transy** .

After operating the **Keys** key a different set of keys will be displayed. Some of them are identical to the keys on a previous key panel display and these will not be described again. The new keys are as follows.

Inp stores the current curve fitted slice information in an array at an address set by the slice counter.

Rev. Changes the **Fit** command to fit a cubic spline to the right hand half of the displayed section rather than the left hand half.

Angle gives up to six options for the angles where the automatic cubic spline fit points are located.

Elipse This key modifies an already spline fitted section to produce an elliptical front to the section. The ellipse is intended to represent the chest wall.

Getsl retrieves a curve fitted slice from memory. The particular slice retrieved is indicated by the counter which is controlled by **Cntup** and **Cntdn.**

Down selects the next lower raw data slice.

Up selects the next higher raw data slice.

Fit automatically fits a sixteen point cubic spline curve to the right hand side of the displayed raw data cross section. The points are located at angles set by **Angle.** If **Rev.** is operated the left hand side is fitted.

Updy increases the height from which the displayed slice is selected by an amount equal to (**Set 2** - **Set 1**)/ **Setdy.**

Dpx0 selects the 0 cubic spline control point for movement by the cursor in the x direction only.

Dpy0 selects the 0 cubic spline control point for movement by the cursor in the y direction only.

Dpx15 selects the 15th cubic spline control point for movement by the cursor in the x direction only.

Dpy15 selects the 15th cubic spline control point for movement by the cursor in the y direction only.

Dp1 - Dp14 select the corresponding control point for movement in either the x or the y direction.

Using the Body Edit and Display (BED) program to fit cubic splines

Once the BED program is running click on the **File** button and when requested, enter the file name of the raw data file to be fitted. Referring to Figure 23 it can be seen that the first slice to be fitted is positioned at the crotch. The cursor should now be positioned at this level and the mouse centre button operated to select the slice. Click on **Set 1** to store this value in the memory. Now move the cursor where the maximum hips slice is judged to be and again operate the mouse centre button and also click on **Set 2** to save the upper limit.

From Figure 23 it can be seen that there are five slices between crotch and hips but the crotch slice is repeated twice (to allow the cubic spline interpolation to start correctly) which leaves three incremental positions between those slices. Click on **Setdy** and when requested enter "3" for the three incremental positions. Now click on **Curve** to change windows and then **Spline.** In the right hand small window the values which have previously been entered for **Set 1** and **Set 2** can be seen.

Hold down the mouse centre button and move the cursor up the screen until the height shown in the centre of the screen agrees with **Set 1.** The crotch slice should now be displayed in the centre of the screen with sixteen cubic spline points arranged round the left hand side of it. To manipulate the positions of the spline points click on **Keys** to change the keyboard. On the new keyboard the key **Angle** allows a choice of six sets of positions for the automatic positioning of the spline

points. The sixth position distributes the points around a full three hundred and sixty degrees, while the other positions only cover the left hand side of the picture which is the right hand side of the subject.

In some positions the angle number changes to red which indicates that this position remains stationary when the other points are being automatically fitted. The purpose of this is to make it easier to eliminate arms. If the spline curve has been manually adjusted to exclude the arms then, when the next adjacent slice is chosen, it is easier to judge where to exclude the arms if the previous spline positions have not changed. To re position the spline control points simply position the cursor over the point to be adjusted, click and release the left hand mouse button then, while holding down the left hand mouse button again, drag the spline control point to its new position. The keys **Dp0** to **Dp15** can also be used to select control points. After a key has been operated the control point will follow the cursor until the left hand mouse button is pressed.

Having achieved a satisfactory cubic spline shape the data can be saved by first operating key **Tglzro**, which resets the memory address counter because this is the first slice, and then by operating key **Inp.** The key **Inp** should be operated twice more because it is necessary to store the first slice three times. To get the next slice click on the **Updy** key which will put the next slice on the screen ready for curve fitting. The spline control points are adjusted, as above, and the slices saved by operating key **Inp** as before. Eventually the height of the next slice, as indicated by the red numbers in the middle of the screen, will approximately coincide with the Figure given in **Set 2.** When this happens fit and save the slice as before and then operate the **Iris** key which will return the display to that of the raw data. Immediately operate key **Set 1** to set to the current slice, and then move the cursor to the waist position on the raw data and operate the middle mouse key followed by **Set 2.** As before by reference to Figure 11 notice that the **Setdy** key requires an input of "9". The whole process is then repeated, gradually working up the body until slice 34 is saved. This slice is saved again so that the last slice is duplicated(to allow the cubic spline interpolation to finish correctly). Finally operate the **Save** key to save the results to disk. When curve fitting to a slice it may be noticed that the person is not correctly centred or may need to be rotated so that the fitted curve best

fits the cross section of the body. This can be adjusted using the **Transx, Transy** or **Roty** keys.

The above process fits cubic spline curves to the right side of the body as seen on the left of the screen. To fit the left side of the body it is easier to make use of the right side file already created. If the right side file is not already loaded then use key **Recall** to get it back into the computer and operate key **Rev**, to instruct the computer that the fit is to be on the left side.. Select the appropriate keyboard with **Keys** and to get the first slice operate **Tglzro** and **Getsl**. First check that the red spline fitted curve fits the blue raw data curve correctly. If it does not then adjust using the **Transx, Transy** or **Roty** keys. When this is correct operate key **Fit** and the cubic spline curve will fit to the right side of the picture. The spline curve can now be adjusted as before to achieve the required shape and save using the **Inp** key. By operating the **Inp, Getsl** and **Fit** keys in turn the whole of the left side of the body can be curve fitted at heights which match those of the right side.

The following photographs illustrate the main features of the program.

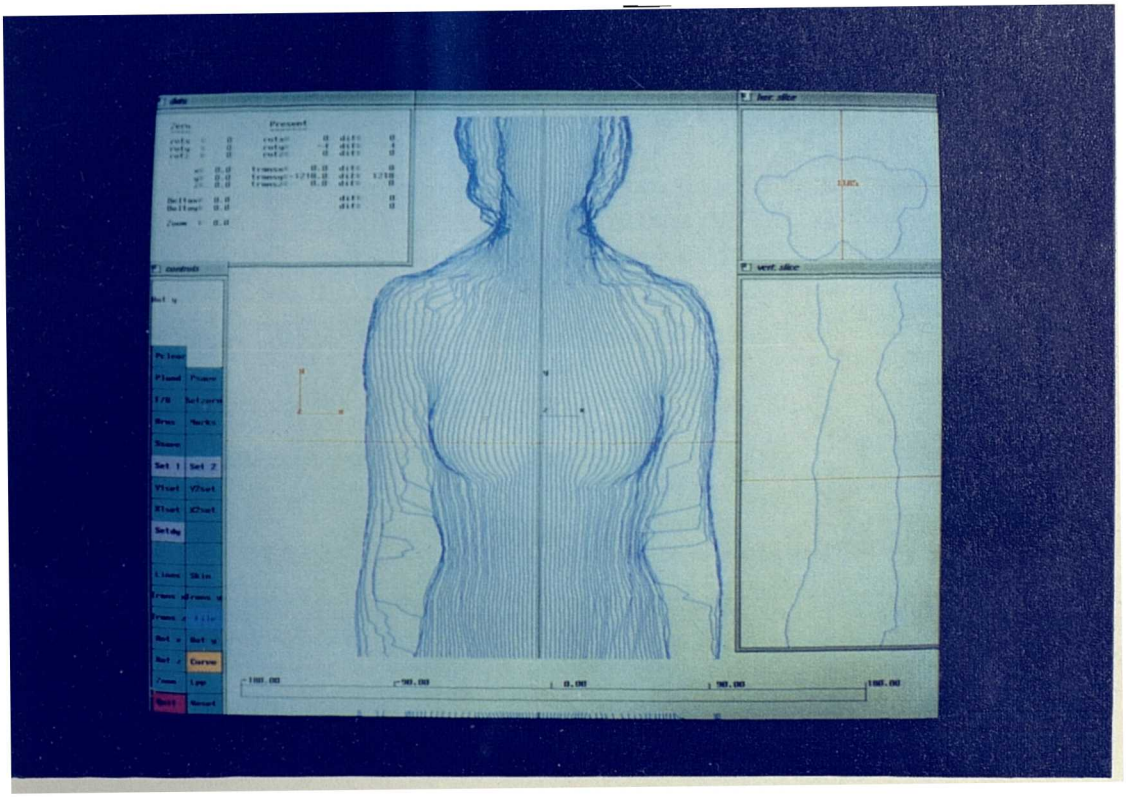


Figure 13. Display of raw data

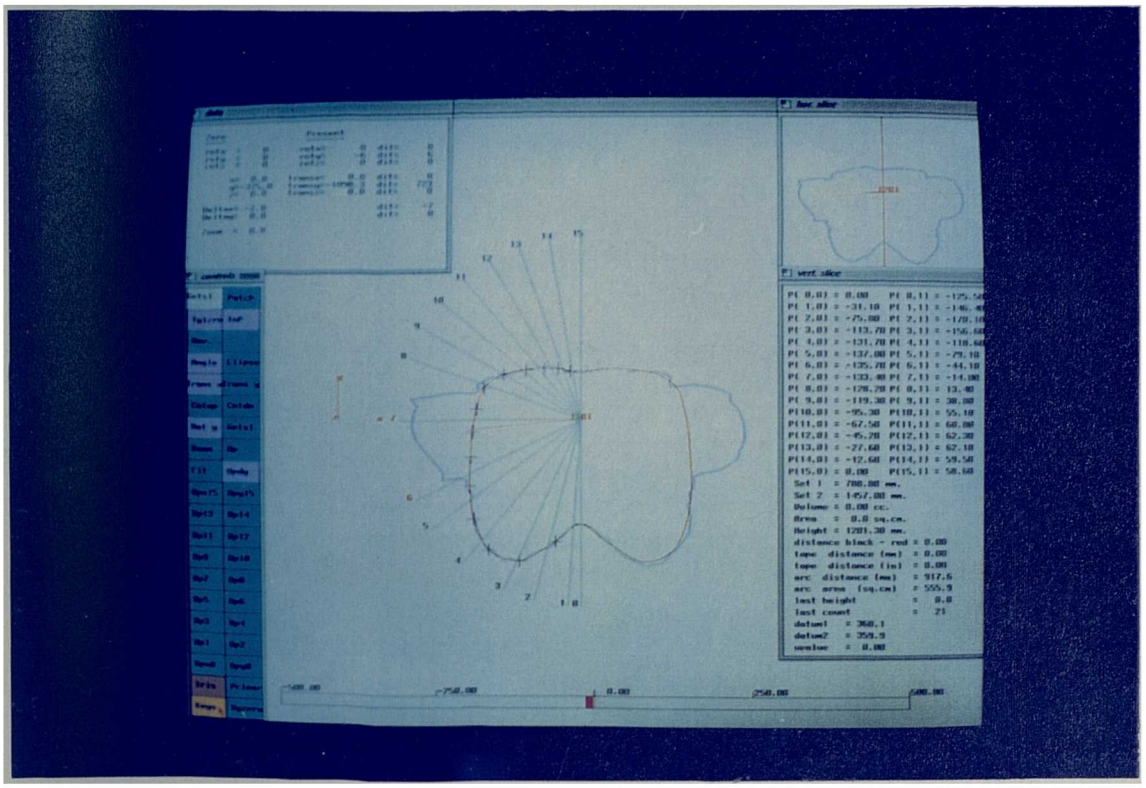


Figure 14. Fitting splines to a cross-section

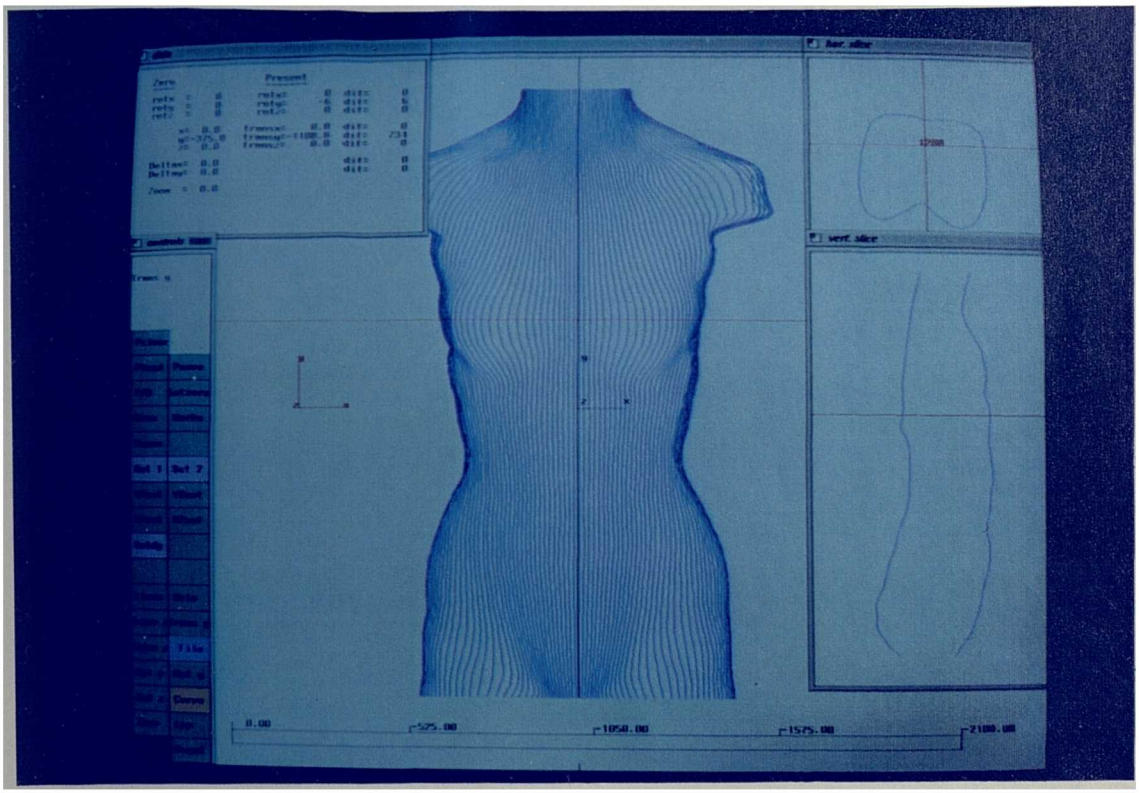


Figure 15. Results of curve fitting and editing

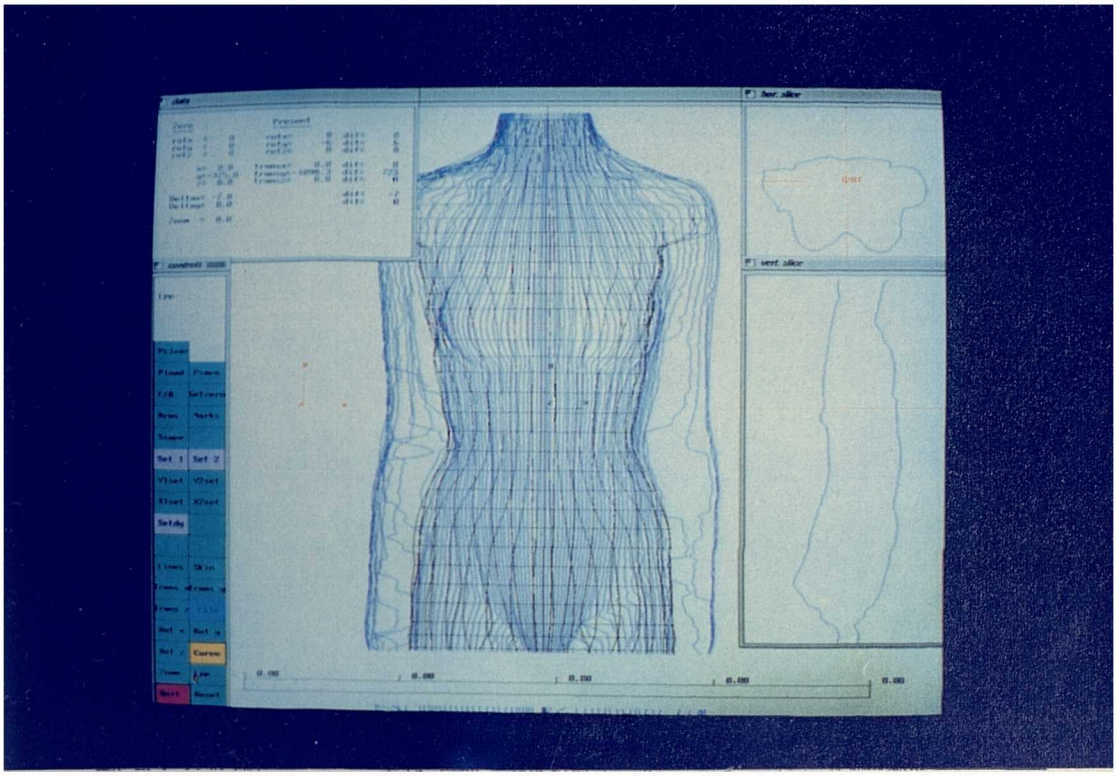


Figure 16. Edited data superimposed on raw data

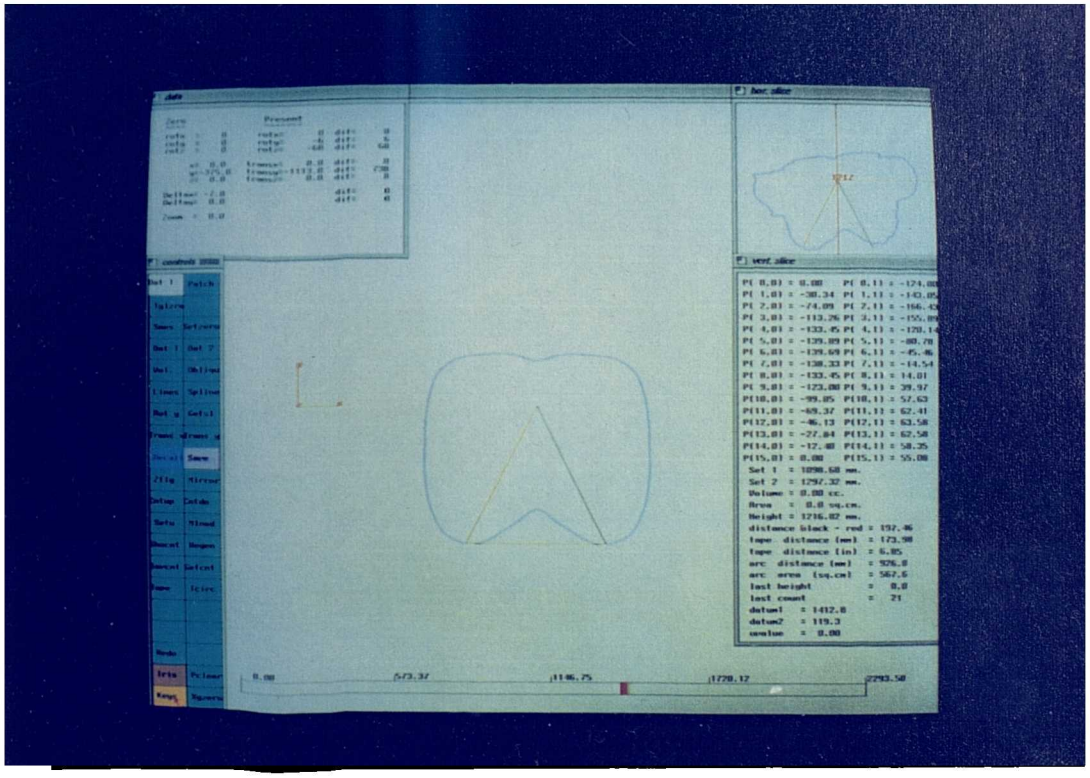


Figure 17. Making measurements on a horizontal slice

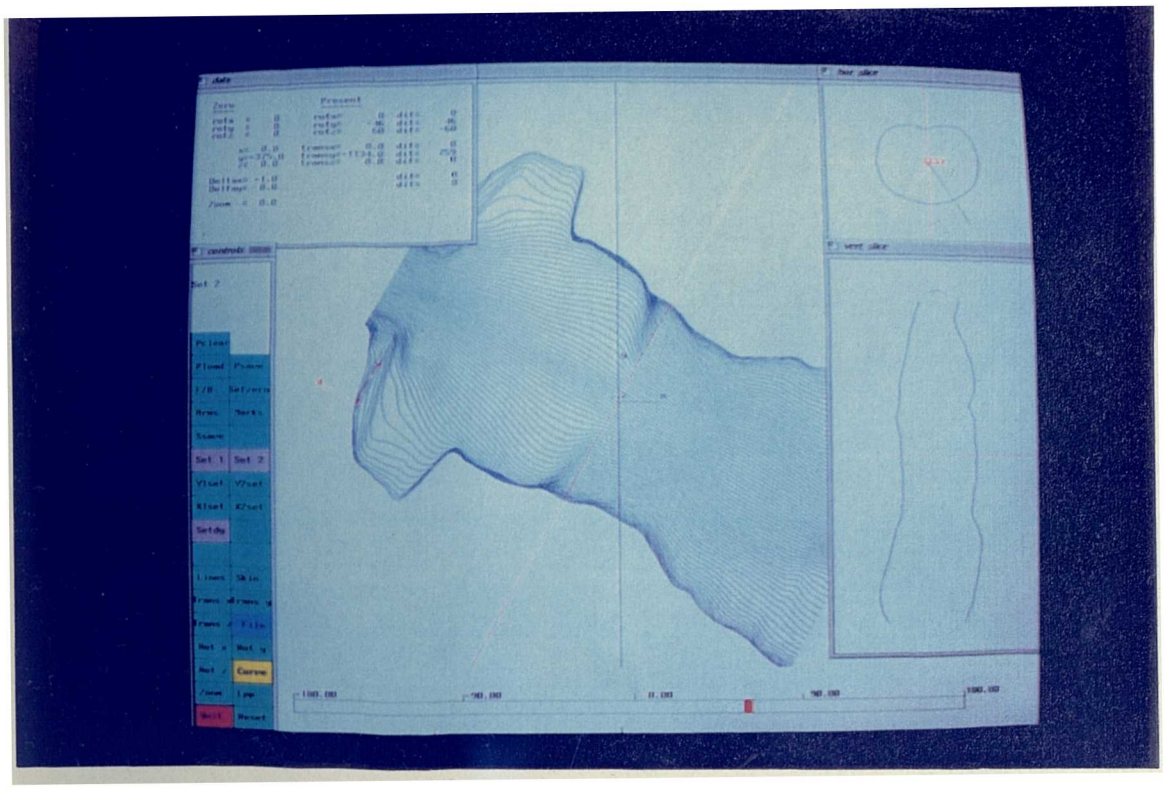


Figure 18. Preparing to take an oblique slice

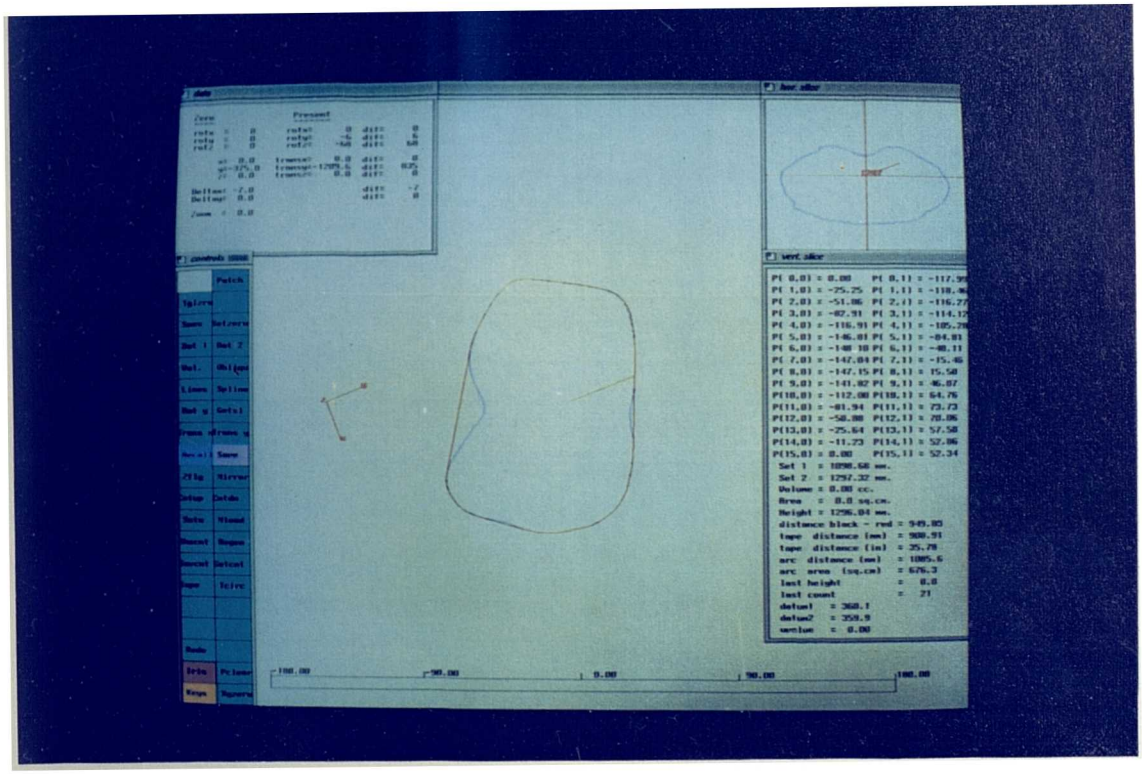


Figure 19. An oblique slice

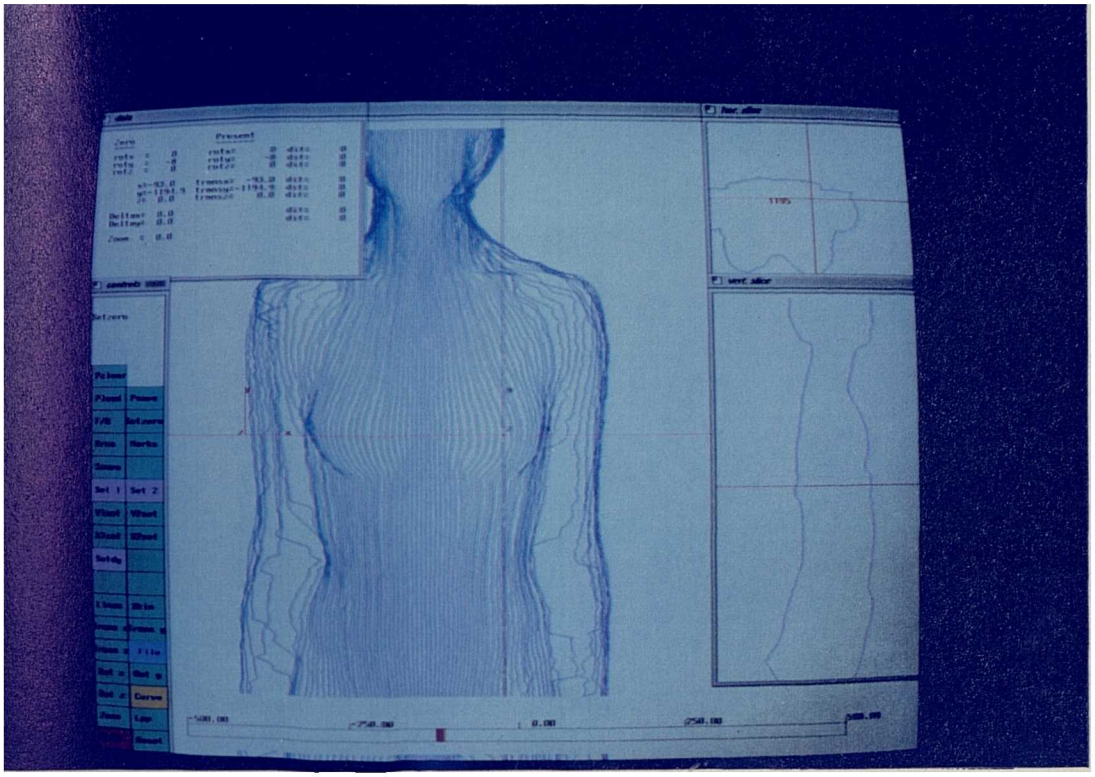


Figure 20. Making measurements on raw data

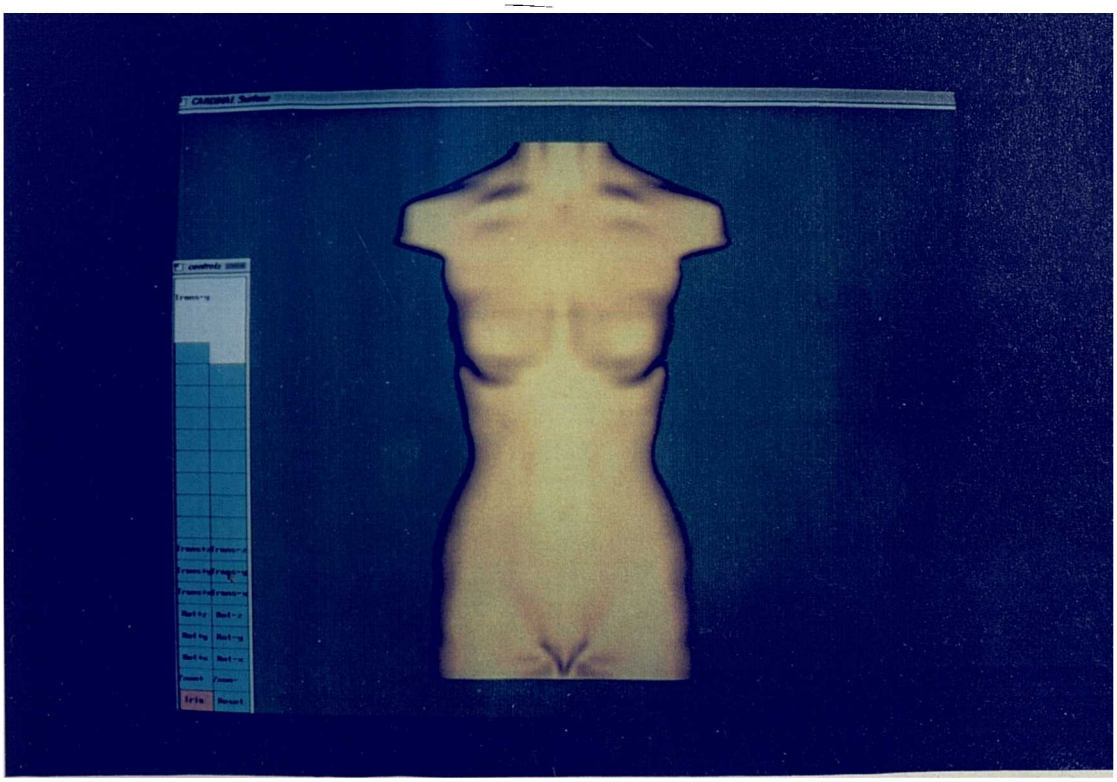


Figure 21. Simulated skin texture

CHAPTER 5

Cubic Splines

Cubic Spline Interpolation

Since the human body shape changes relatively slowly in a series of gentle curves it is not necessary to digitize points on the surface which are very close together. If widely separated points are chosen for digitization then the intermediate points can be obtained by interpolation. The method chosen to calculate the interpolated points used Cardinal splines (a type of parametric cubic curve), as they pass through the data points. This also allowed considerable reduction of data.

Parametric Curves

A parametric cubic curve is one for which x, y, and z are each represented as a third order polynomial of some parameter t.

Hence,

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$
$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y \quad 0 \le t \le 1$$
$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$

Expressed as a vector product,

$$C(t) = at^3 + bt^2 + ct + d$$

$$= [t^3\ t^2\ t\ 1] \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$= TM$$

Where T is the vector $[t^3\ t^2\ t\ 1]$; M is the column vector $\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$

The shape of the curve segment is determined by the coefficients of the vector product, which are stored in the column vector M.

These coefficients can be expressed as a set of four control points. Thus the vector product becomes,

$$C(t) = TM = T(BG)$$

where G is a set of four control points, or the *geometry*, and B is a matrix called the *basis*.

The basis matrix governs the relationship between the control points and the actual cubic curve generated. There are several well known bases used in curve plotting; Bartels, Beatty and Barsky (1987) but the one chosen for this application is the 'Cardinal' basis. This has the property that if four control points are chosen, say a,b,c,d, then the curve produced will terminate at points b and c. If further overlapping control points are chosen, say b,c,d,e, then the next curve will be drawn between c and d. By taking a series of points four at a time, chosen so that they overlap, as above, a continuous smooth curve can be obtained which passes through all of the control points. If the control points are the data points of the body shape then an accurate curve can be obtained which represents that shape and which continuously interpolates between the points.

The equation for plotting a curve using Cardinal splines is,

$$\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1.0 & -2.5 & 2.0 & -0.5 \\ -0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = P(x) \qquad (1)$$

where v_1, v_2, v_3, v_4 are the control vertices for the curve

and $\begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1.0 & -2.5 & 2.0 & -0.5 \\ -0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{bmatrix}$ is the basis matrix (B) which produces a cardinal spline. Because the cardinal spline actually passes through v_2 and v_3 then it is possible to arrange a continuous succession of curves with overlapping control vertices which lie on the data points.

Producing a 3-D Surface

Equation (1) produces data for the perimeter of a single horizontal section. However, a 3-D surface is required. This could be made from a multitude of horizontal sections, but it would be extremely tedious and unnecessary to define and edit the geometry for every horizontal section. The cardinal spline technique can again be used to interpolate vertically between a few horizontal sections. The sections are then taken four at a time to make a surface patch.

Surface Patches

Four horizontal sections can be defined in cubic splines as above, and four other cubic spline curves can be plotted which pass through each of the four control vertices from each horizontal spline. Because these curves are able to interpolate between the control points of the horizontal sections, an infinite number of horizontal sections can be created which will define a three dimensional surface.

To represent this surface in matrix algebra, two parameters are specified (u,w). One traversing the curve in the horizontal plane and the other in the vertical plane. Assuming a matrix of control points (v) then the position of a point P is.

$$P(u,w) = [u^3 + u^2 + u + 1]B\begin{bmatrix} v_{1.1} & v_{1.2} & v_{1.3} & v_{1.4} \\ v_{2.1} & v_{2.2} & v_{2.3} & v_{2.4} \\ v_{3.1} & v_{3.2} & v_{3.3} & v_{3.4} \\ v_{4.1} & v_{4.2} & v_{4.3} & v_{4.4} \end{bmatrix} B^T \begin{bmatrix} w^3 \\ w^2 \\ w \\ 1 \end{bmatrix}$$

where **B** = *basis* matrix.
where $\mathbf{B}^T$ = a transposed *basis* matrix where rows are exchanged for columns.

$$\text{For a Cardinal curve } \mathbf{B} = \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1.0 & -2.5 & 2.0 & -0.5 \\ -0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{bmatrix}$$

In order to create a surface patch going through sixteen points in space ,

$$P = \begin{bmatrix} P_{1.1} & P_{1.2} & P_{1.3} & P_{1.4} \\ P_{2.1} & P_{2.2} & P_{2.3} & P_{2.4} \\ P_{3.1} & P_{3.2} & P_{3.3} & P_{3.4} \\ P_{4.1} & P_{4.2} & P_{4.3} & P_{4.4} \end{bmatrix} \qquad U = \begin{bmatrix} u_{1.1} & u_{1.2} & u_{1.3} & u_{1.4} \\ u_{2.1} & u_{2.2} & u_{2.3} & u_{2.4} \\ u_{3.1} & u_{3.2} & u_{3.3} & u_{3.4} \\ u_{4.1} & u_{4.2} & u_{4.3} & u_{4.4} \end{bmatrix}$$

$$V = \begin{bmatrix} v_{1.1} & v_{1.2} & v_{1.3} & v_{1.4} \\ v_{2.1} & v_{2.2} & v_{2.3} & v_{2.4} \\ v_{3.1} & v_{3.2} & v_{3.3} & v_{3.4} \\ v_{4.1} & v_{4.2} & v_{4.3} & v_{4.4} \end{bmatrix} \qquad W = \begin{bmatrix} w_{1.1} & w_{1.2} & w_{1.3} & w_{1.4} \\ w_{2.1} & w_{2.2} & w_{2.3} & w_{2.4} \\ w_{3.1} & w_{3.2} & w_{3.3} & w_{3.4} \\ w_{4.1} & w_{4.2} & w_{4.3} & w_{4.4} \end{bmatrix}$$

then ,

$$P = UBVB^TW^T$$

If we wish to find the matrix of control points (v) which will fit a surface patch to some data points (p) then given :-

$$P = UBVB^TW^T$$

multiply both sides by $(UB)^{-1}$

$$(UB)^{-1} P = (UB)^{-1} UBVB^TW^T$$

or $$(UB)^{-1} P = VB^TW^T$$

using the identity $(AB)^T = B^TA^T$

$$(UB)^{-1} P = V(WB)^T$$
$$= WBV^T$$

solving for **V**

$$V^T = (WB)^{-1}(UB)^{-1} P$$

or since $(AB)^{-1} = B^{-1}A^{-1}$

$$V^T = (UBWB)^{-1} P$$

This now allows the control points to be calculated for the given set of data points.

CHAPTER 6

Dealing with Data

The raw data output from the LASS scanner is not ideal for many of the uses to which it will be put. One of the main problems is the fact that the scanner produces the radius profile over the arms, while many of the required measurements, waist, hips, bust etc., are measured inside the arms. In many instances for the clothing industry the data are required to be symmetrical about the vertical axis. The raw data must then be edited in some way to remove arms and to achieve symmetry. It is easy to envisage a two dimensional editor operating at the pixel level, but what about three dimensions? Such an editor would be tedious in the extreme to use, and it would be very difficult to edit out arms, for instance, without destroying the general form of the original data.

The method chosen for the editor is to consider the raw data as a series of horizontal cross-sections. These sections are then subjected to a semi-automatic cubic spline curve fitting process which accurately matches the original data, with the exception of the parts subject to change, such as the removal of the arms. Instead of pixel editing, a 'rubber band' technique is used to manipulate the form of the cubic spline. Not every available cross-section needs to be manipulated in this way, only those which define major changes in shape. Cardinal cubic splines were used because they have the property of exactly passing through the control points, which can also be aligned on some of the original data points, so that accuracy is maintained. The choice of the number of control points is based on a compromise between ease of editing and the amount of variation in one body slice. As the human body is generally a rounded shape sixteen control points were found to be sufficient to define the cubic spline representing half of the body's horizontal cross-section. The other half is then considered to be a mirror image of the first half. Having represented each of a number of slices by sixteen control points it is possible to use the same cubic spline method to interpolate vertically between corresponding points of each slice to generate control points for as many intermediate slices as required.

The Choice of Sections

If the height of the sections which are chosen before interpolation are selected arbitrarily then some of the detail may be smoothed out in the interpolation. It is also desirable for the slices of one body to correspond with the slices of another. To achieve this the torso was defined as that part of the body which fills the space between crotch and chin. A check was then made, as follows, to see if people of differing heights were reasonably in the same proportion. The proportional heights of a number of horizontal sections were compared as shown in the table.

Subject	Underbust	Under-arm	Manu-brium	Sternal Notch	Shoulders
1	0.559	0.765	0.847	0.910	0.949
2	0.566	0.752	0.815	0.897	0.917
3	0.587	0.750	0.818	0.876	0.916
4	0.602	0.767	0.805	0.895	0.944
5	0.585	0.805	0.861	0.898	0.923
6	0.556	0.790	0.852	0.888	0.927
7	0.574	0.822	0.856	0.909	0.948
8	0.574	0.796	0.84	0.886	0.987
Average	0.580	0.781	0.837	0.895	0.939
St. Dev.	0.016	0.026	0.021	0.011	0.024
Model 34B	0.578	0.776	0.842	0.896	0.922

Table 1 The height of anatomical points as a proportion of stature.

The following Figure 22 illustrates the proportions as shown in the above table.

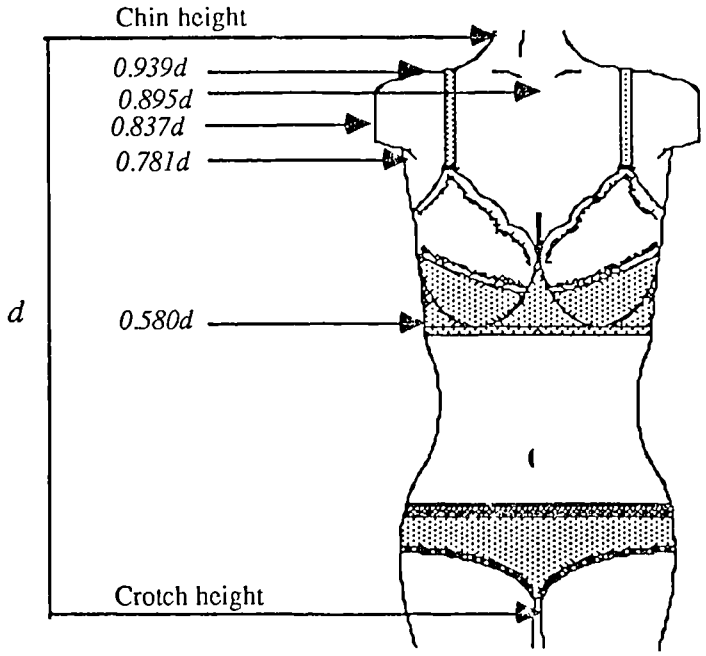


Figure 22. The position of the underbust, underarm, manubrium, sternal notch and shoulders expressed as a fraction of the crotch to chin distance.(d)

Having established that people are essentially proportional a series of major sections were chosen as shown in Figure 22 and numbered to include other sections which will be interpolated between the major sections. Data for the full thirty five sections are contained in a file to be known as the **Shape Matrix.**

The sections are chosen as in Figure 23, and interpolation is used to provide a sufficient number of sections to create the 3-D shape. For the torso thirty two sections are used which are then transformed into surface patches for plotting on the computer screen. As detailed below the data for the thirty two sections are used to make a standard data file or shape matrix.

Standard Data file format

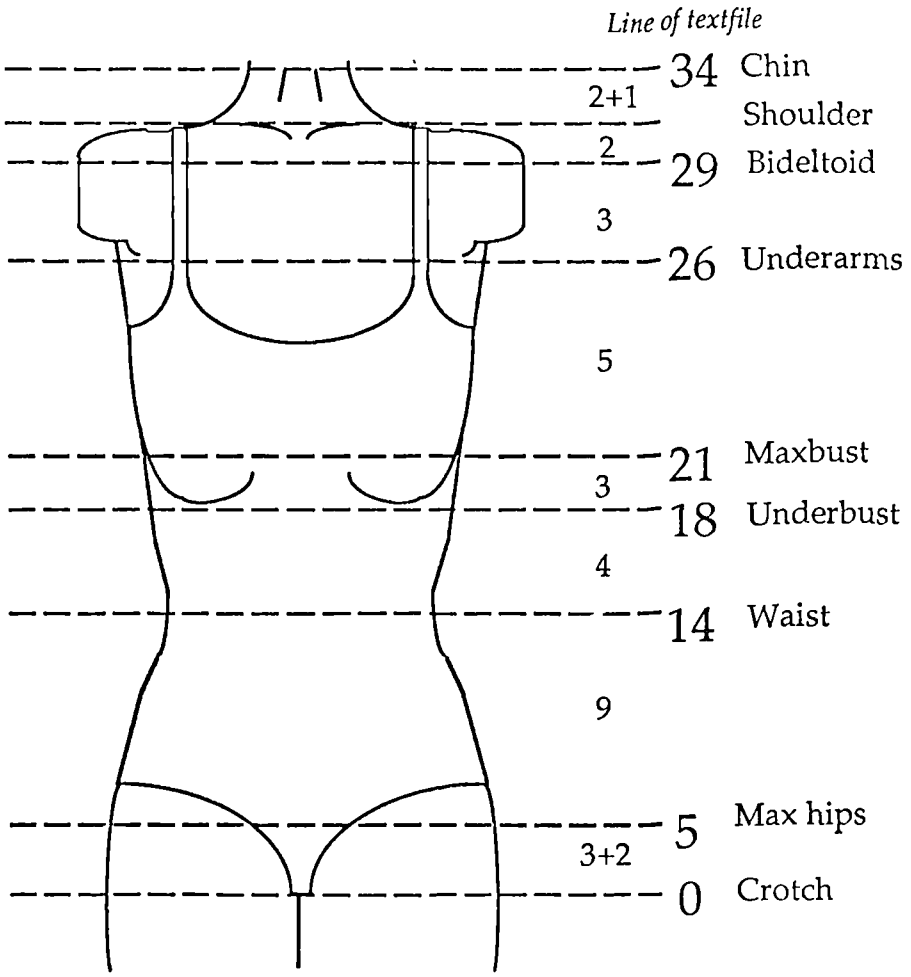


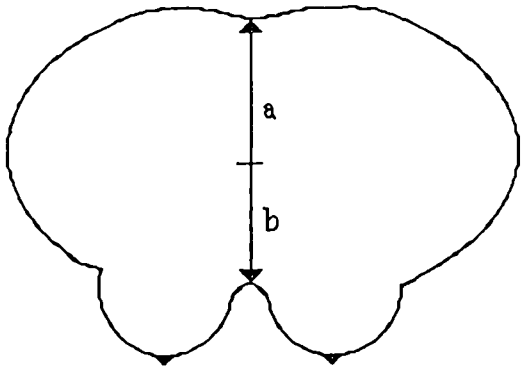
Figure 23. Arrangement of slices in the standard data file/Shape Matrix

Shape Matrix

The Shape Matrix is a computer file containing sixteen x,y co-ordinate values on each of thirty five lines. Each line of data represents sixteen points on the right hand half of one horizontal slice of a torso. The end column of each line contains a value in mms for the height of the slice from the ground. The first three lines are identical and the last two are also identical to allow the vertical interpolation to start and end in an organised way.

Comparing one body with another

When scanned a person does not necessarily stand with the same posture every time. The comparison of one body shape with another is also posture dependent. In many instances it is an advantage to remove the effects of a droopy posture before a comparison of one body with another is made. This is achieved by taking the average z axis (the axis running from front to back.) value of the centre front and the centre back of each slice and subtracting that value from all of the z axis readings of the other points around the slice.



To re-centre the section the radius vectors a & b are replaced by two equal vectors equal to (a + b)/2.

Figure 24. Section showing re-centring method

In addition to this, the various points arising from a cubic spline curve fit, do not necessarily occur at equal angles around the vertical axis of the body. By using cubic splines again it is possible to automatically, in the computer, recalculate the points so they define points at equal angular spacing around the vertical axis. When this has been done the shape matrices of the two bodies should be converted to polar co-ordinates and subtracted from each other. This will directly give the differences of the radii at corresponding parts of each body at corresponding angles. The calculation of the mean and standard deviation of these differences will give a meaningful figure of relative shape. By taking fewer slices the same technique can be used to compare parts of the body.

Having corrected the co-ordinates in this fashion they now represent an array of 16 x 32 points (remembering that the first three and last two lines are duplicates.) which correspond to a similar set of points on another body regardless of height or girth of the different bodies.

The data for the thirty five sections can now be considered as a map of the body surface. Since the vertical scaling, as shown in Figure 23, provides the scaling for differing body heights then each data point corresponds to a particular anatomical point on the body surface. This enables different body data to be compared directly and also allows various 3-D shapes to be averaged by simply taking the numerical average of the co-ordinates of corresponding points, from the data for each shape to be averaged. It is very important with any survey work that the data are reduced in some way since the data for a large number of individuals are of little use unless some kind of average figure can be calculated. An average can easily be calculated for data obtained with a tape-measure but it is believed that the method described above is a novel approach for the averaging of 3-D body data.

Creating shapes from minimum data

With the above method it has been possible to achieve a very large reduction in the amount of data required to specify a body shape, but could the method be usefully taken further in order to create a body shape from a few simple tape-measure measurements?

By making use of the fact that people are generally the same shape, but with the main variation largely dependent on whether they are wide or narrow, it is possible to have a data base ranging from the very narrowest shapes to the very widest and select suitable cross sections from within this range.

The data so far has originated from fitting cubic splines to thirty five cross sections of the body at pre-determined relative positions corresponding to anatomical features. If we consider one cross section, say, through the bust, then the shape will vary considerably between that of a slim person and that of a stocky person, but there will be a smooth transition between the extreme shapes. If we now consider a master file containing the data for the two extreme sections mentioned above, and two further intermediate sized sections. By applying the same cardinal spline interpolation technique as before, but this time to every corresponding point in the four sets of data in the master file, we can create the data for any cross section between the most stocky and slimmest extremes. The parameter which decides where we select in the continuum of shapes is chosen to be the circumference of a

particular section. We are now in a position where we can use traditional circumferential measurements, such as waist, hips, bust etc., to select from a master file containing the accumulated knowledge of the shape of the population at large, to predict the actual shape of that particular cross section of the person to whom the measurements apply. Obviously the master file contains the data for all thirty-five slices required to define the body shape for the two extreme shapes and the two intermediate ones, and so allow the total shape of a torso to be created from a few simple measurements.

To take thirty-five circumferential measurements would still be quite a chore, but again this is unnecessary since the human body changes, in the vertical direction, generally in smooth curves which allows for yet more interpolation. In practice the circumferences of eight horizontal sections, along with the heights at which they are taken is all that is necessary.

The mathematics of creating shapes from minimum data

The shape of a torso can be defined, as above, by means of an array containing the x,y. values of sixteen points for each of thirty-five slices, plus a value for the height of each slice. If a master file is created which combines the above array for the slimmest person, a person intermediate in size between the slimmest and an average person, a person intermediate in size between an average person and the stockiest person and finally a stockiest person to give a total file of one hundred and forty lines long where each line contains the coordinates of one slice. For every defined point on a torso there is in the file four points which range from the narrowest to the widest.

If we consider only one slice at a time then the following array can be obtained, from the master file, such that a_0 - a_{16} represents the slice from the narrowest person b_0 - b_{16} and c_0 - c_{16} from the intermediate persons and d_0 - d_{16} from the widest persons, where a_{16} - d_{16} are the heights of the corresponding slices from the ground.

$$D = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 & b_{10} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} \\ c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} \\ d_0 & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 & d_9 & d_{10} & d_{11} & d_{12} & d_{13} & d_{14} & d_{15} & d_{16} \end{bmatrix}$$

Then the sixteen points which define a new slice

$$= [u^3 \quad u^2 \quad u \quad 1]BD$$

where u is an interpolating parameter, which due to the properties of Cardinal splines will interpolate between the b data and the c data as u varies from 0 to 1.

To interpolate between the a data and the b data the array D is rearranged as follows :-

$$D = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 & p_9 & p_{10} & p_{11} & p_{12} & p_{13} & p_{14} & p_{15} & p_{16} \\ a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 & b_{10} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} \\ c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} \end{bmatrix}$$

A dummy row of data is created such that the x and y co-ordinate values are positioned on the opposite side of the a values at the same distance that the b values are from the a values. i.e.,

$$p = a - (b - a)$$
$$= 2a - b$$

Similarly to interpolate between the c data and the d data

$$D = \begin{bmatrix} b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 & b_{10} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} \\ c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} \\ d_0 & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 & d_9 & d_{10} & d_{11} & d_{12} & d_{13} & d_{14} & d_{15} & d_{16} \\ q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} & q_{11} & q_{12} & q_{13} & q_{14} & q_{15} & q_{16} \end{bmatrix}$$

where :-

$$q = d + (d - c)$$
$$= 2d - c$$

So that the u parameter can uniquely define the interpolated cross-section independent of which of the above conditions apply an arbitrary integer is added as follows :-

For interpolation between a and b 0 is added.

For interpolation between b and c 1 is added.

For interpolation between c and d 2 is added.

By this means the integer part of the u parameter signals which array to use while the fractional part of the u parameter is used in the calculation.

If a different u parameter is known for each of the thirty-five slices then a new file can be created which defines a totally new shape not

necessarily existing in the original data which made up the master file The individual slices are interpolations from the master data, but the combination of slices is new.

What is required now is a means of relating the u parameter to the circumference of a particular slice. The following flow diagram gives an iterative method for achieving this.

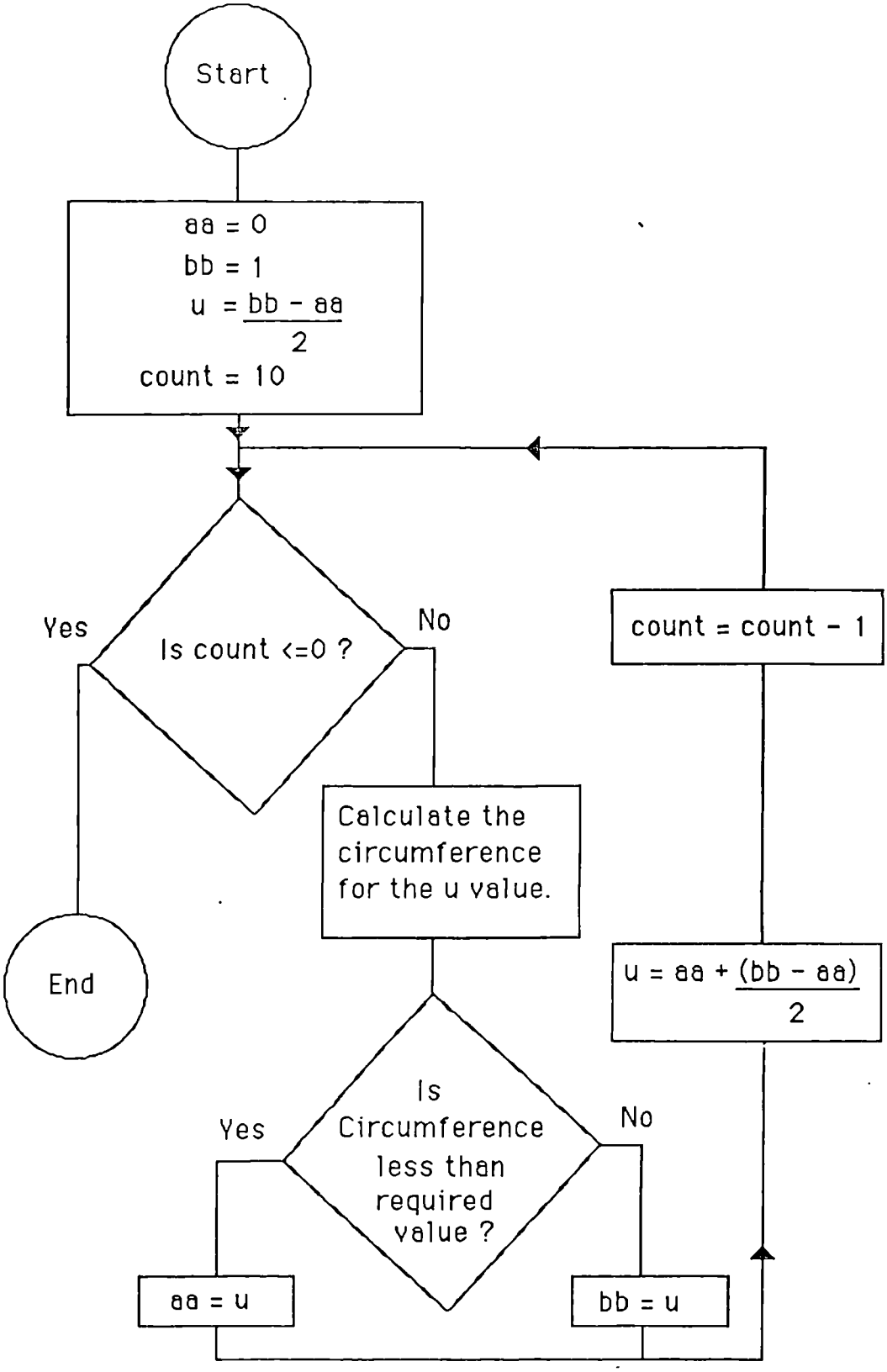


Figure 25. Iterative method to relate u parameter to circumference

The above method is used to select u parameters and then calculate the circumference in a systematic way so that the correct answer can be obtained in ten or less tries. This method still requires a value for the circumference of each of the thirty-five slices yet the input data file has circumference information for only eight slices.

A typical input data file is as follows :-

0	816.2	929.95
9	944.3	856.58
16	1083.8	660.63
18	1125.2	725.06
21	1185.9	904.99
24	1255.0	883.63
28	1346.7	958.04
35	1436.0	361.88

Where the first column is the slice number, the second column is the height of the slice and the third column is the slice circumference. A Cardinal spline interpolation method is used, yet again, to interpolate the data to give circumference values for each of thirty-five slices. Exactly as described previously the data is taken in groups of four, with dummies calculated for the end effect. Interpolation is calculated between the middle values for each group of four and then the groups are re-arranged by leaving out the lower value and including the next higher value for the next interpolation calculation.

CHAPTER 7

Algorithms for Oblique and Tape measure Measurements

Oblique slices

Given a set of discreet data points it is not a simple matter to read data from an arbitrary plane which slices through these data points because, in many instances, the chosen plane could miss many of the data points altogether. Even if the surface is defined as a set of cubic splines, and therefore capable of infinite resolution, there is the compute time to be considered. The following two methods were used to compute an oblique slice. The first when the oblique slice was defined by selecting three points on the slice and the second when the slice was selected visually on the screen by rotating and translating the required slice to the y = 0 plane. The method of translation and rotation is identical in both cases.

Co-ordinate transformations to determine the boundary of an oblique slice through a digitised human being

Assuming that the data are in cylindrical co-ordinates as shown in Figure 26.

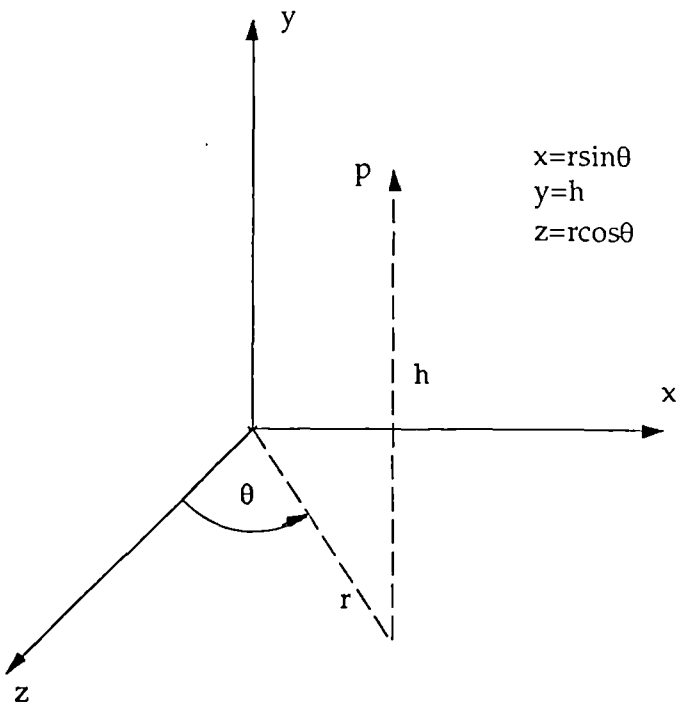


Figure 26. Position of a point in cylindrical co-ordinates.

If the required plane is defined by three points as in Figure 27.

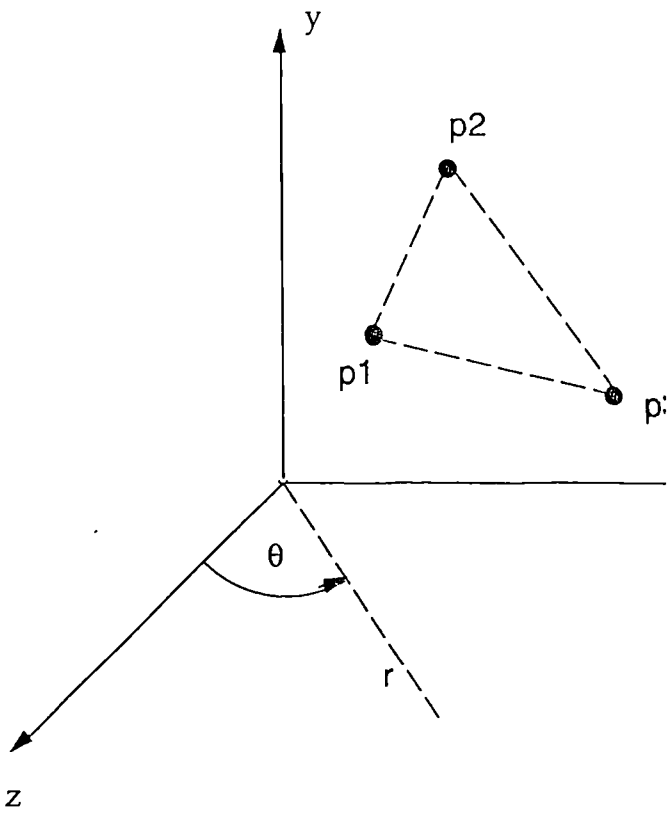


Figure 27. Position of a plane defined by 3 points in cylindrical co-ordinates.

If P_1 ,P_2 and P_3 are three points on the plane defined in terms of r,θ and h then:

To convert to right handed cartesian co-ordinates.

$x_1 = r_1\sin\theta$	$x_2 = r_2\sin\theta$	$x_3 = r_3\sin\theta$
$y_1 = h_1$	$y_2 = h_2$	$y_3 = h_3$
$z_1 = r_1\cos\theta$	$z_2 = r_2\cos\theta$	$z_3 = r_3\cos\theta$

The following manipulations are required to find the rotation functions necessary to rotate the required plane to the z=0 plane.

Translate p1 to origin.

$$\begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 & 0 \\ y_1 & y_2 & y_3 & 0 \\ z_1 & z_2 & z_3 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = A$$

Calculate rotation functions.

$$\sin\phi = \frac{a_{12}}{\sqrt{a_{32}^2+a_{12}^2}} \quad \cos\phi = \frac{a_{32}}{\sqrt{a_{32}^2+a_{12}^2}}$$

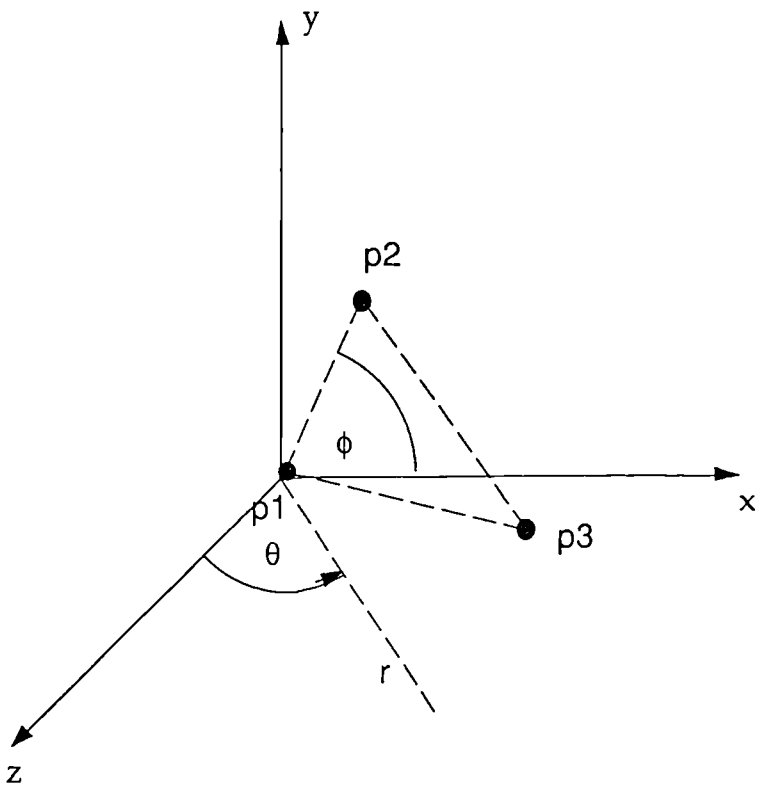


Figure 28. Translation of point P_1 to origin.

Rotate about the z axis. (Figure 29).

$$\begin{bmatrix} \cos\phi & \sin\phi & 0 & 0 \\ -\sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} A = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = B$$

Calculate rotation functions.

$$\sin\alpha = \frac{b_{32}}{\sqrt{b_{12}^2+b_{32}^2}} \quad \cos\alpha = \frac{b_{12}}{\sqrt{b_{12}^2+b_{32}^2}}$$

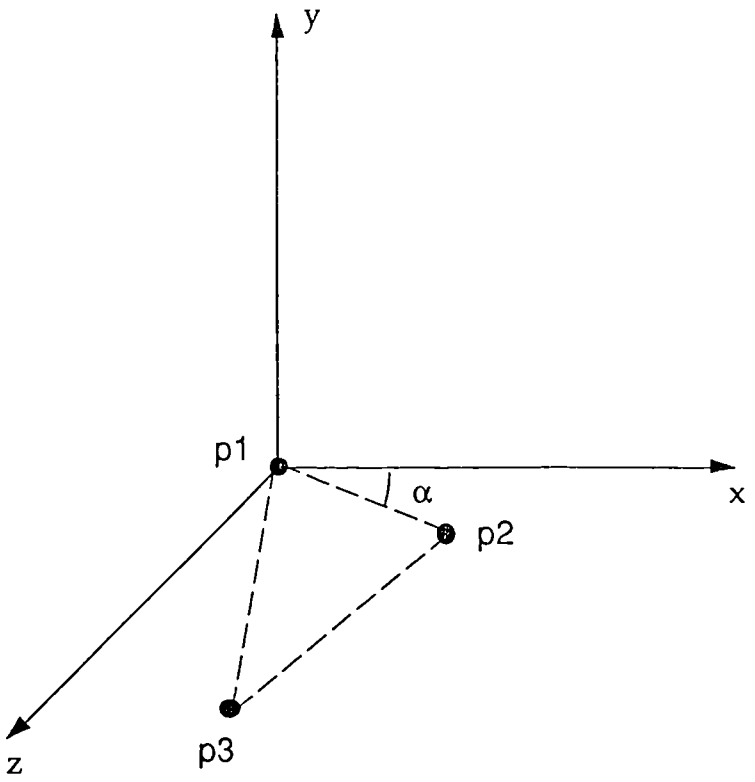


Figure 29. Rotation about the z axis to bring P_2 to the x,z axis.

Rotate about the y axis. (Figure 30).

$$\begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{B} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix} = C$$

Calculate rotation functions.

$$\sin\beta = \frac{c_{23}}{\sqrt{c_{23}^2+c_{33}^2}} \quad \cos\beta = \frac{c_{33}}{\sqrt{c_{23}^2+c_{33}^2}}$$

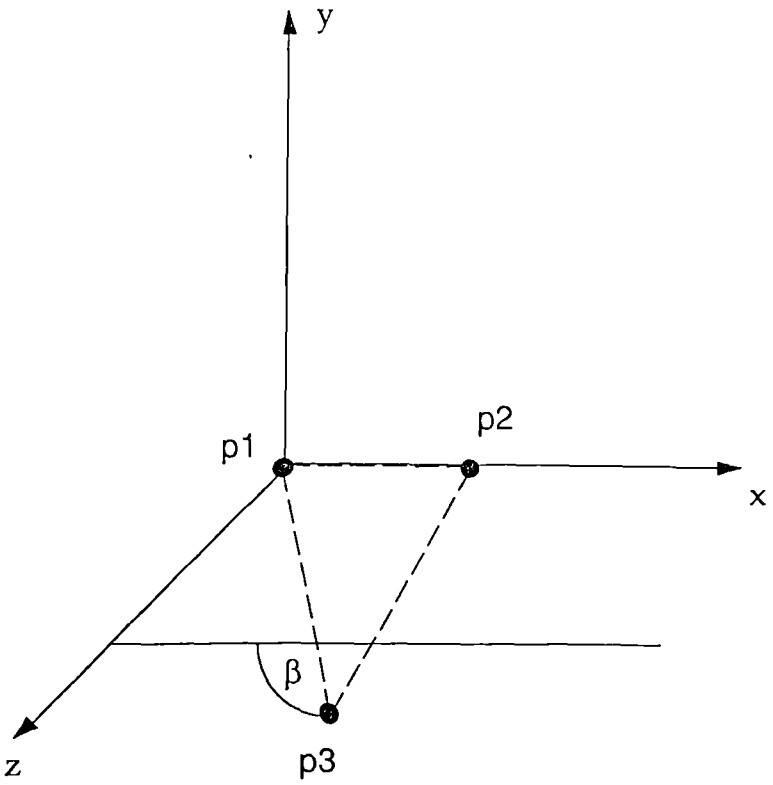


Figure 30. Rotation about the y axis to place p_2 on x axis.

Rotate about the x axis. (Figure 31).

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & \sin\beta & 0 \\ 0 & -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} C = \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \\ d_{41} & d_{42} & d_{43} & d_{44} \end{bmatrix} = D$$

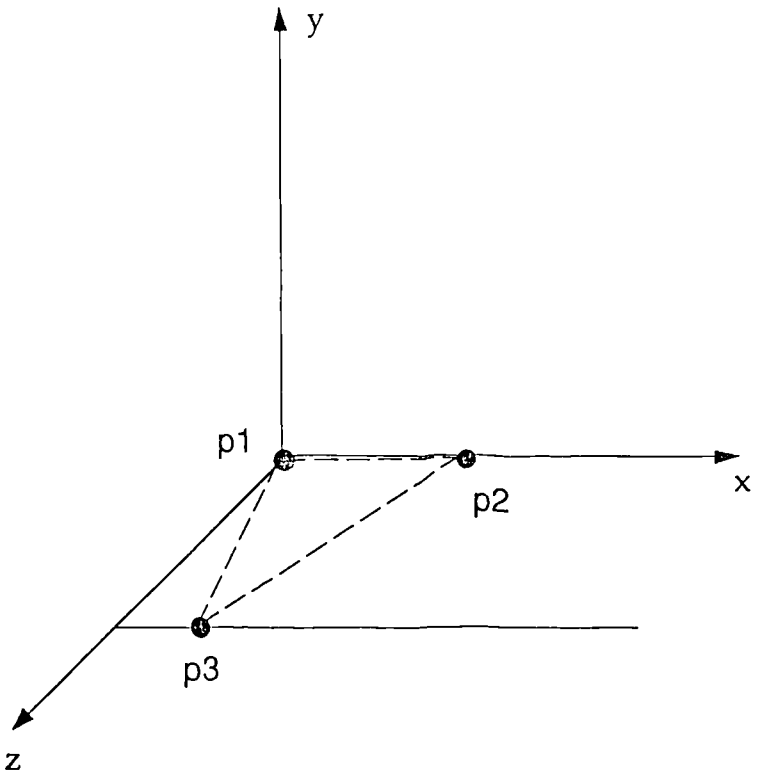


Figure 31. Rotation about the x axis to bring p_3 onto the x,z plane.

Once the rotational matrices have been established then any co-ordinates x,y,z. can be converted to co-ordinates p,q,r. such that the points which lie on the plane defined by p1,p2 and p3 will lie on the y=0 plane as follows:

$$\begin{bmatrix} \cos\phi & \sin\phi & 0 & 0 \\ -\sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & \sin\beta & 0 \\ 0 & -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix} = \begin{Bmatrix} p \\ q \\ r \\ 1 \end{Bmatrix}$$

The 3 rotation matrices can be multiplied together first to give a single matrix and so reduce the calculation time.

If the rotations are performed on a graphic display it may be more convenient to rotate the required plane to the x=0 plane since the rotations are arbitrary.

Selection of points which straddle the x = 0 plane.

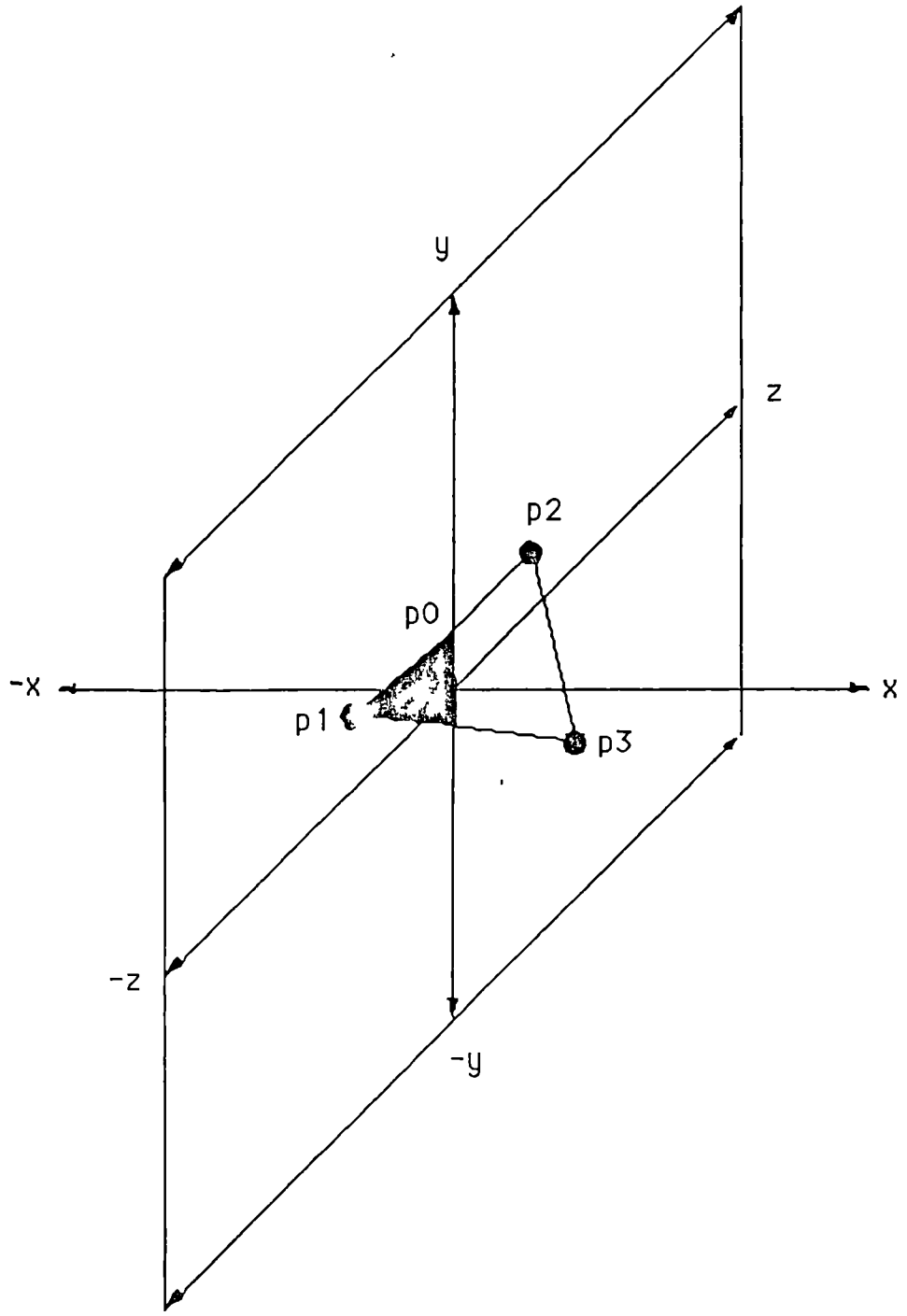


Figure 32. Showing 3 points which straddle the x=0 plane.

If a slice is taken through a series of points representing the shape of a three dimensional object, then in many instances, depending on the position of the slice relative to the array of points, the slice will pass in between the points and so miss them. One solution to this problem is

to move the axis so that the slice being considered is say the x=0 plane. If the data points are now plotted 3 at a time, then when the triangle formed by the three points straddles the x=0 plane Figure 32, one of the points will have the opposite sign to the other two points irrespective of the position of the plane relative to the data.

To determine the co-ordinates x_0,y_0,z_0 where one side of the triangle cuts the x = 0 plane :-

If the co-ordinates of P_1 are x_1,y_1,z_1.
& the co-ordinates of P_2 are x_2,y_2,z_2.
Then the direction cosines are:-

$$l = \frac{(x_2-x_1)}{\sqrt{(x_2 - x_1)^2+(y_2 - y_1)^2+(z_2 - z_1)^2}}$$

$$m = \frac{(y_2-y_1)}{\sqrt{(x_2 - x_1)^2+(y_2 - y_1)^2+(z_2 - z_1)^2}}$$

$$n = \frac{(z_2-z_1)}{\sqrt{(x_2 - x_1)^2+(y_2 - y_1)^2+(z_2 - z_1)^2}}$$

And the length of the vector (p_1,p_2) is :-

$$d = \sqrt{(x_2 - x_1)^2+(y_2 - y_1)^2+(z_2 - z_1)^2}$$

If d_0 is the length of the vector from p1 to the x = 0 plane:

$$x_1 + d_0 l = x_0$$
$$y_1 + d_0 m = y_0$$
$$z_1 + d_0 n = z_0$$

Since $x_0 = 0$ then

$$d_0 = \frac{-x_1}{l}$$
$$y_0 = y_1 + m\left(\frac{-x_1}{l}\right)$$
$$z_0 = z_1 + n\left(\frac{-x_1}{l}\right)$$

Tape measure Measurements

Having collected data which represents the body surface it is a requirement that various measurements can be taken from it. The methods which measure the shortest distance between two points, as in engineering, or the surface distance, as in land surveying, are well known but a method which gives the distance as measured by a tailor using a measuring tape is not so well known. Generally the tape is required to span the hill tops and ignore the valleys

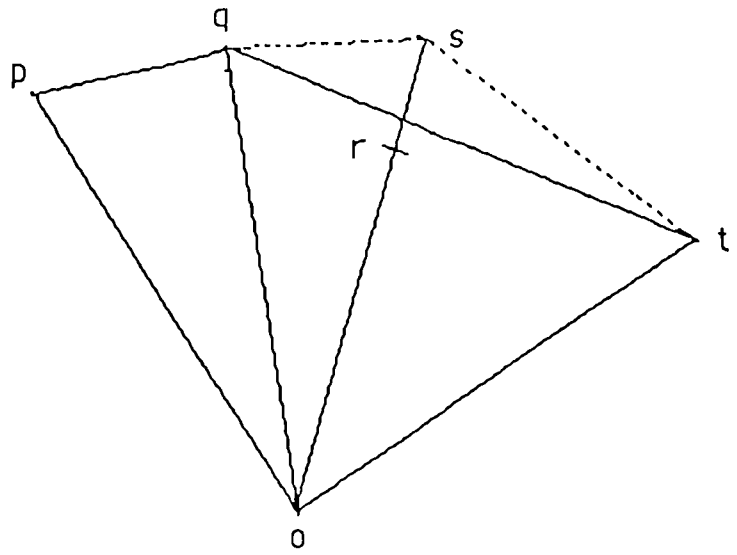


Figure 33. Polar plot of four points to fit tape measure

In the Figure p,q,r and t are points plotted in polar co-ordinates about the centre at o. If a tape is to be stretched round the points from p to t then it would need to miss the point at r. If the points are p,q,s and t then the tape would need to include all the points. What is needed is an algorithm which decides when the line or will cut the line qt.

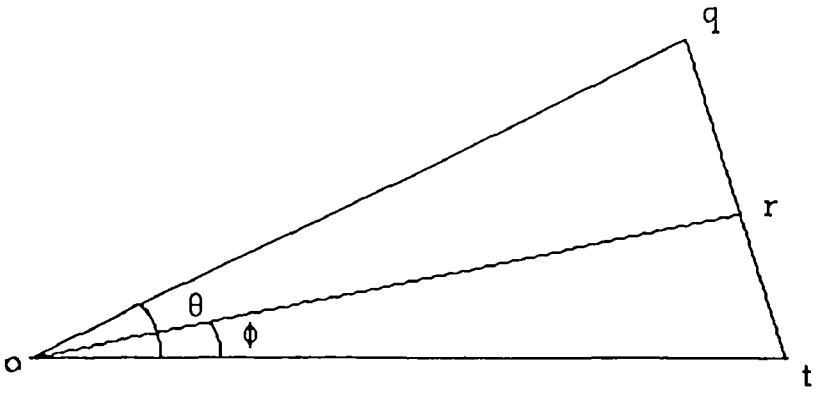


Figure 34. Triangle for calculating tape measure fit

If the triangle in Figure 34 represents the points o,q,t in Figure 33 and or being when r just lies on the line qt. We need to calculate the length of or which just touches the line qt. If the calculated length of o,r is less than the actual length of or then the point r misses the tape measure and conversely if the calculated length of or is greater than the actual length of or then the point r should be included in the tape measure data file. The following is the derivation of the formula for obtaining the calculated length of or.

Using the sine rule on triangle oqt.

$$\frac{qt}{\sin\theta} = \frac{oq}{\sin \angle otq} = \frac{ot}{\sin \angle oqt} \quad \text{.....1}$$

Using the sine rule on triangle otr.

$$\frac{rt}{\sin\phi} = \frac{or}{\sin \angle otq} \quad \text{.....2}$$

From *1*

$$\sin\angle otq = \frac{oq.\sin\theta}{qt}$$

From 2

$$or = \frac{rt.\sin\angle otq}{\sin\phi} = \frac{rt\,.\,oq.\sin\theta}{qt.\sin\phi}$$

In triangle orq

$$\frac{qr}{\sin(\theta - \phi)} = \frac{or}{\sin\angle oqr}$$

$$or = \frac{qr.\sin\angle oqr}{\sin(\theta - \phi)} \quad \text{.....3}$$

But $\angle oqr = \angle oqt$

Therefore from *1*

$$\sin\angle oqr = \frac{ot.\sin\theta}{qt}$$

Substitute in 3

$$or = \frac{qr.ot.\sin\theta}{qt.\sin(\theta - \phi)}$$

$$qr = \frac{or.qt.\sin(\theta - \phi)}{ot.\sin\theta}$$

$$rt = \frac{or.qt.\sin\phi}{oq.\sin\theta}$$

But $qt = qr + rt$

Therefore

$$qt = \frac{or.qt.\sin(\theta - \phi)}{ot.\sin\theta} + \frac{or.qt.\sin\phi}{oq.\sin\theta}$$

$$or = \frac{\sin\theta}{\frac{\sin(\theta - \phi)}{ot} + \frac{\sin\phi}{oq}}$$

$$= \frac{ot.oq.\sin\theta}{oq.\sin(\theta - \phi) + ot.\sin\phi}$$

The computer program calculates the length of o,r for the previous point and if the calculated value is larger than the actual value the point is included in the tape measurement and the next point is selected. If the calculated value is smaller than the actual value then all the previous points are selected in turn until a calculated value larger than an actual value is found. This is because if a new point is higher than a previous point it may now cause the tape to miss one or more of the points which previously made contact with the tape. To provide a sensible starting point a start is made from the highest point (largest radius) and then the tape is fitted in either direction from that.

CHAPTER 8

COMPUTER PROGRAMS

Automated anthropometry relies on the application of computers and the associated software. It is unthinkable to suppose that the amount of data to be processed could be handled in any other way.

The software used in conjunction with the LASS scanner falls into three groups. First the program called "newsnapy" which runs in the ELTEC computer and which controls all the functions of the scanner. To display and manipulate and take measurements from the scanned data there is the Body Edit and Display program (BED) originally called "lass" which is in nine parts "lass1.c" to "lass9.c". The "lass" program is written to run in a "Silicon Graphics" computer. Finally there is a group of ancillary programs used to average scans or take differences between scans and also to manipulate the data. All of the software is written in the "C" language,Kerninghan and Ritchie (1978).

Compilation of "newsnapy" program

Should it be necessary to make changes to the program it will need to be re-compiled in the computer in which it will run. By altering the RESOLUTION flag in the lassdefs file the program can be compiled either to digitise in low resolution or in high resolution. Low resolution means that the data is taken every five degrees around the body and at ten millimetre vertical intervals while for high resolution the data is taken every 2.4 degrees and at five millimetre vertical intervals. As can be seen from the following command file the program is made from several separate source code files.

The command file which will compile the various parts of the program and link them together to make a single object code program is given below:

cc newsnap.c newcamera.c newdrive.c newapalset.r mouse.r newplot.c newfile.c newxyzm.c -x -l=/h0/hirespac/lib/hrg.clib -i -k=21f -f=/h0/cmds/newsnapy

Either the above must be typed as a command or it must be the contents of a command file which can then be run to achieve compilation.
A full listing of the source code is given in Appendix 5.

Body Edit and Display program assembly (lass)

In the same way that the "newsnapy" program is written in parts so the complete lass program is separated into nine sections in order to ease the problem of program development. By altering the RESOLUTION flag in the lassdefs file the program can be compiled either to digitise in low resolution or in high resolution. To match the scanner low resolution means that the data is taken every five degrees around the body and at ten millimetre vertical intervals while for high resolution the data are taken every 2.4 degrees and at five millimetre vertical intervals.

To compile the complete program on the "Silicon Graphics" computer the following text should be made into a file called **makefile** then type "make".

"lass : lass1.o lass2.o lass3.o lass4.o lass5.o lass6.o lass7.o lass8.o lass9.o
cc -O1 lass1.c lass2.c lass3.c lass4.c lass5.c lass6.c lass7.c lass8.c lass9.c -lgl_s -lm -o lass
lass1.o lass2.o lass3.o lass4.o lass5.o lass6.o lass7.o lass8.o lass9.o :lassdefs"

This again will compile and link all the parts of the program to create a single object code usable program.
A full listing of the source code is given in Appendix 6.

ancillary programs

Several ancillary programs have been written in order to process the LASS data. A brief description of each of these follows:

Average

This program will produce a shape matrix which is the average of several other shape matrices. When running the program it prompts for input file names until the **esc** key is typed when the program prompts for an output file name.

Gradsplot3

Using the appropriate shape matrix as the input file the program prompts for input and output file names, title and issue number and then produces a text file written in HP-GL(Hewlett-Packard Graphics Language) which when listed to a suitable plotter will plot the torso to give the true horizontal surface distance plotted from centre front. The plot also includes a standard 34b torso drawn in red for use as a reference. The shape matrix for this called "av34b.xmfac" must be in the same directory as the program when it is run.

Gradsplot4

As gradsplot3 but plots a true horizontal surface distance plot from centre back.

Gradsplotvert

As gradsplot3 or gradsplot4 but plots a true vertical surface distance plot.

Isometric3

This program plots an inclined front view as a wire frame model in order to better recognise surface features which can be related to the surface distance plots.. The wire frame grid corresponds to those on the surface distance plots.

Isometric4

As for Isometric3 above but the program plots an inclined rear view.

Masterfix

In some instances when comparing the shapes of two or more people their posture when they were scanned presents a problem. Generally people stand in a different way every time they are measured. This program considers each horizontal slice in turn and re calculates its centre to be half way between centre front and centre back and creates a new shape matrix.

Newangle

When curve fitting, in order to create a shape matrix, it is common to move some of the sixteen reference points so that a better fit to the desired shape can be achieved. The result of this is that the angular position of the points in the shape matrix do not correspond between bodies. This program recalculates a new set of points to a standard set of angular positions.

Splot

The input to this program is not a shape matrix but a file produced when the "Ssave" key is operated when running the LASS program. The splot program then produces a text file in HP-GL which will cause the plotter to plot as many horizontal cross-sections as were specified in the original file.

Vslice

This program will create a text file in HP-GL from a shape matrix file in order to plot vertical sections. The program prompts for the radial angle measured from the centre front which defines the vertical section.

Vslice2

As for vslice above but the sections are parallel to the vertical or "z" axis

The ancillary source codes are listed in Appendix 5.

CHAPTER 9

Results

Lass calibration data

The LASS measuring system was tested in two ways. A static test was performed using a series of calibrated cylinders positioned at various heights within the 2.1 m vertical measuring range. This checked the ability of the system to measure stationary objects. To ascertain the overall system accuracy, 8 live subjects were measured using both anthropometry and LASS. Brooke-Wavell, Jones, and West (1993).

Static Tests.

The following table gives the result of a calibration check which compares diameters measured in the range 100 - 600 mm. against known diameters. Measurements were taken at discrete heights within the 2.1 metre vertical measuring range as described in Table 3.

Table 2 Results obtained from the measurement of calibrated cylinders.

Height	Width	Depth	Mean	Actual	Actual-Mean
60	508	508	508	500	8
80	497	497	497	500	-3
120	597	594	595.5	600	-4.5
130	596	594	595	600	-5
140	399	399	399	400	-1
190	297	297	297	300	-3
230	497	497	497	500	-3
240	196	196	196	200	-4
280	397	397	397	400	-3
280	196	196	196	200	-4
320	597	596	596.5	600	-3.5
330	296	297	296.5	300	-3.5
380	499	499	499	500	-1
380	198	198	198	200	-2

Table 2 Results obtained from the measurement of calibrated cylinders. (Continued.)

430	400	401	400.5	400	0.5
430	100	101	100.5	100	0.5
470	600	600	600	600	0
480	298	299	298.5	300	-1.5
520	497	498	497.5	500	-2.5
530	201	201	201	200	1
570	102	102	102	100	2
580	401	401	401	400	1
630	297	297	297	300	-3
670	496	495	495.5	500	-4.5
930	299	299	299	300	-1
970	499	499	499	500	-1
980	201	201	201	200	1
1030	402	403	402.5	400	2.5
1030	106	106	106	100	6
1080	600	601	600.5	600	0.5
1080	300	300	300	300	0
1130	498	498	498	500	-2
1130	199	197	198	200	-2
1170	103	103	103	100	3
1180	402	402	402	400	2
1230	300	300	300	300	0
1280	499	501	500	500	0
1280	202	201	201.5	200	1.5
1330	400	400	400	400	0
1330	101	101	101	100	1
1380	600	600	600	600	0
1380	301	300	300.5	300	0.5
1430	200	200	200	200	0
1440	499	500	499.5	500	-0.5
1470	100	98	99	100	-1
1480	401	400	400.5	400	0.5
1530	600	600	600	600	0

Table 2 Results obtained from the measurement of calibrated cylinders. (Continued.)

Height	Width	Depth	Mean	Actual	Actual-Mean
1530	300	300	300	300	0
1680	600	601	600.5	600	0.5
1680	299	300	299.5	300	-0.5
1730	497	500	498.5	500	-1.5
1730	198	197	197.5	200	-2.5
1770	102	102	102	100	2
1780	400	400	400	400	0
1830	302	299	300.5	300	0.5
1840	599	600	599.5	600	-0.5
1870	505	510	507.5	500	7.5
1900	200	201	200.5	200	0.5
1900	500	501	500.5	500	0.5
1940	93	99	96	100	-4
1940	93	99	96	100	-4
1950	403	404	403.5	400	3.5
1950	93	100	96.5	100	-3.5
1960	94	100	97	100	-3
2000	300	301	300.5	300	0.5
2050	201	202	201.5	200	1.5
2090	93	93	93	100	-7

Table 3 Vertical static calibration.

Height	Actual	Height - Actual
450	453	-3
400	403	-3
350	353	-3
300	303	-3
250	253	-3
200	203	-3
1050	1053	-3
1000	1003	-3
950	953	-3
900	903	-3
850	853	-3
800	803	-3
1750	1753	-3
1700	1703	-3
1650	1653	-3
1600	1603	-3
1550	1553	-3

Repeatability Tests

While the wooden cylinders, described above, were an appropriate static test to check the static accuracy of the scanner, a further test is needed to check the overall repeatability of both the scanner and the method of editing the data after the scan. To check the repeatability a person was scanned three times, having stepped off and on the scanner between each scan. The data for each scan was curve fitted and the three shape matrices created. At each of the 16 points on each of the 32 slices within each shape matrix the radius was calculated from the x and y co-ordinate values given. These were subtracted from each other to give a mean and SEM for the difference in radii. In addition the circumference was calculated for each of the thirty two heights in the shape matrix and the differences calculated as shown in Table 4. When considering a comparison of all of the lines in the shape matrix it can

be seen that the SEM is fairly high but if the top seven lines are removed then the SEM is low. An explanation for this is that the repeatability test is also testing the repeatability of the posture of the subject being scanned. Where there are significant horizontal discontinuities, such as at the shoulder or where the arms have been truncated, a small variation in height will make a large difference in radius which shows in the figures in Table 4.

Table 4 Difference in radii between the Shape Matrices of 3 scans.

Shape Matrix difference.	Mean difference (mm)	SEM (mm)
Scan 1 - scan 2 (lines 1-32)	9.6	12.9
Scan 1 - scan 3 (lines 1-32)	8.6	21.2
Scan 2 - scan 3 (lines 1-32)	-1.0	20.9
Scan 1 - scan 2 (lines 1-25)	3.7	5.1
Scan 1 - scan 3 (lines 1-25)	0.2	7.6
Scan 2 - scan 3 (lines 1-25)	-3.5	6.0

Dynamic Tests.

Some of the results from Brooke-Wavell et al (1993) are reproduced here.

Five men and five women were measured by anthropometry and by LASS with the result as shown in table 5. All measurements were made twice by an experienced observer and once by a more experienced observer. Scan data were curve fitted twice by one observer and once by a second observer.

Intra- and inter-observer variability in measurements by anthropometry and LASS is shown in Figures 35 and 36. Intra-observer differences were generally small, the largest being 7 mm for waist circumference by anthropometry.

Inter-observer differences ranged from 3.0 to 13.1 mm for anthropometry and 1.3 to 8.5 mm for LASS.

Table 5 Comparison of anthropometric and LASS measurements at 7 sites in 5 women and 5 men (mean ± SEM)

Women (n=5)	Anthropo-metry	LASS	Difference (LASS - anthropometry)
neck circumference (mm)	325.8 ± 10.0	336.6 ± 11.5	10.9 ± 2.6 *
chest circumference (mm)	904.1 ± 37.2	887.3 ± 42.6	-16.8 ± 5.9 *
waist circumference (mm)	734.6 ± 30.1	745.5 ± 30.7	10.9 ± 4.6
hip circumference (mm)	984.6 ± 39.9	978.8 ± 41.5	-5.8 ± 2.6
waist width (mm)	261.5 ± 9.2	268.0 ± 9.7	5.4 ± 1.3 *
waist depth (mm)	195.6 ± 7.3	204.1 ± 9.1	8.5 ± 2.6 *
waist height (mm)	1028.1 ± 25.5	1021.5 ± 25.5	-6.5 ± 1.5 *
Men (n=5)	Anthropo-metry	LASS	Difference (LASS - anthropometry)
neck circumference (mm)	389.3 ± 12.3	417.1 ± 18.6	27.8 ± 10.6
chest circumference (mm)	973.8 ± 27.1	963.9 ± 30.4	-9.9 ± 4.9
waist circumference (mm)	866.4 ± 36.5	870.8 ± 39.5	4.4 ± 5.5
hip circumference (mm)	986.9 ± 24.6	985.2 ± 23.9	-1.7 ± 3.0
waist width (mm)	307.1 ± 11.5	308.9 ± 11.6	1.8 ± 1.8
waist depth (mm)	224.2 ± 10.2	238.1 ± 11.1	13.9 ± 2.3 *
waist height (mm)	1099.7 ± 28.4	1097.9 ± 29.3	-1.8 ± 2.9

* Anthropometric and LASS measurements significantly different by paired t-test: $p<0.05$.

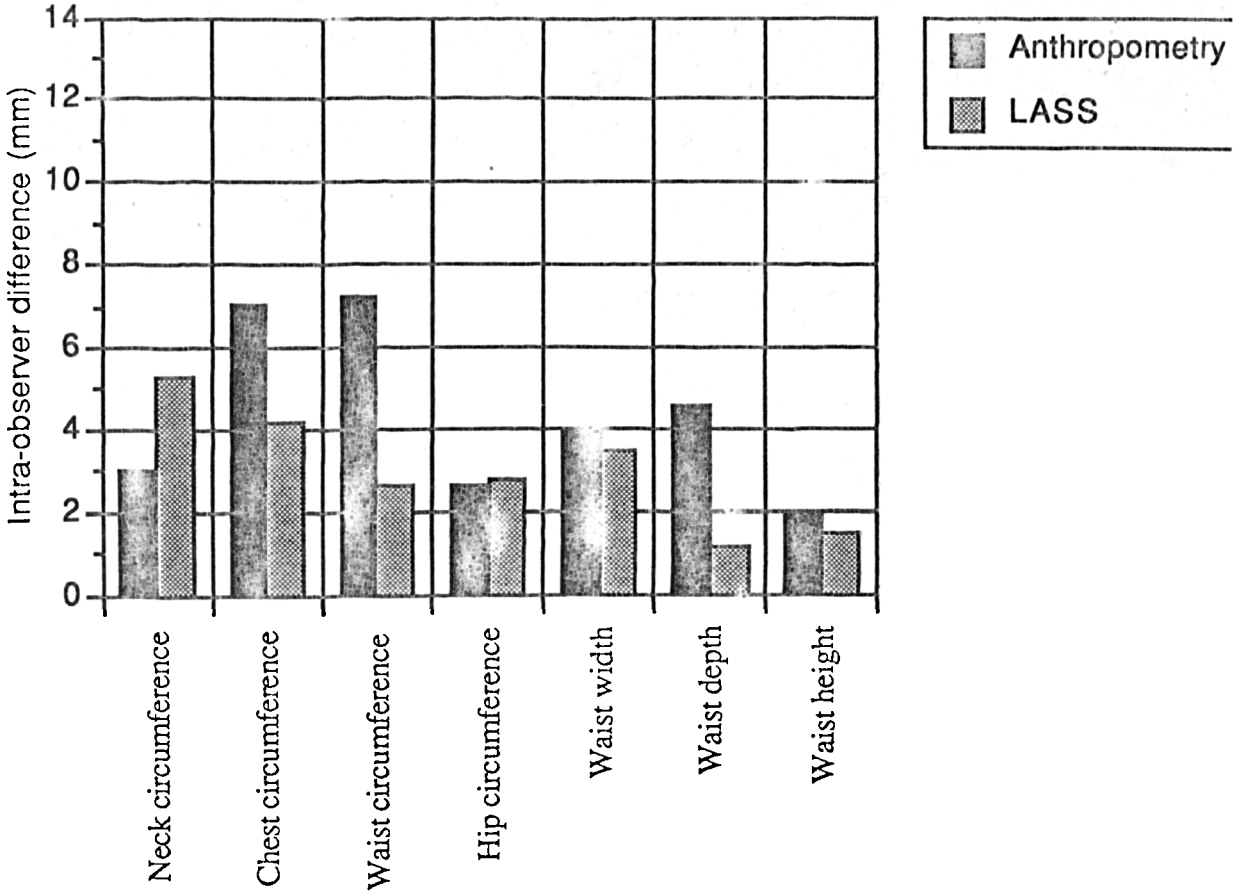


Figure 35 Intra-observer variability in anthropometric and LASS measurements.

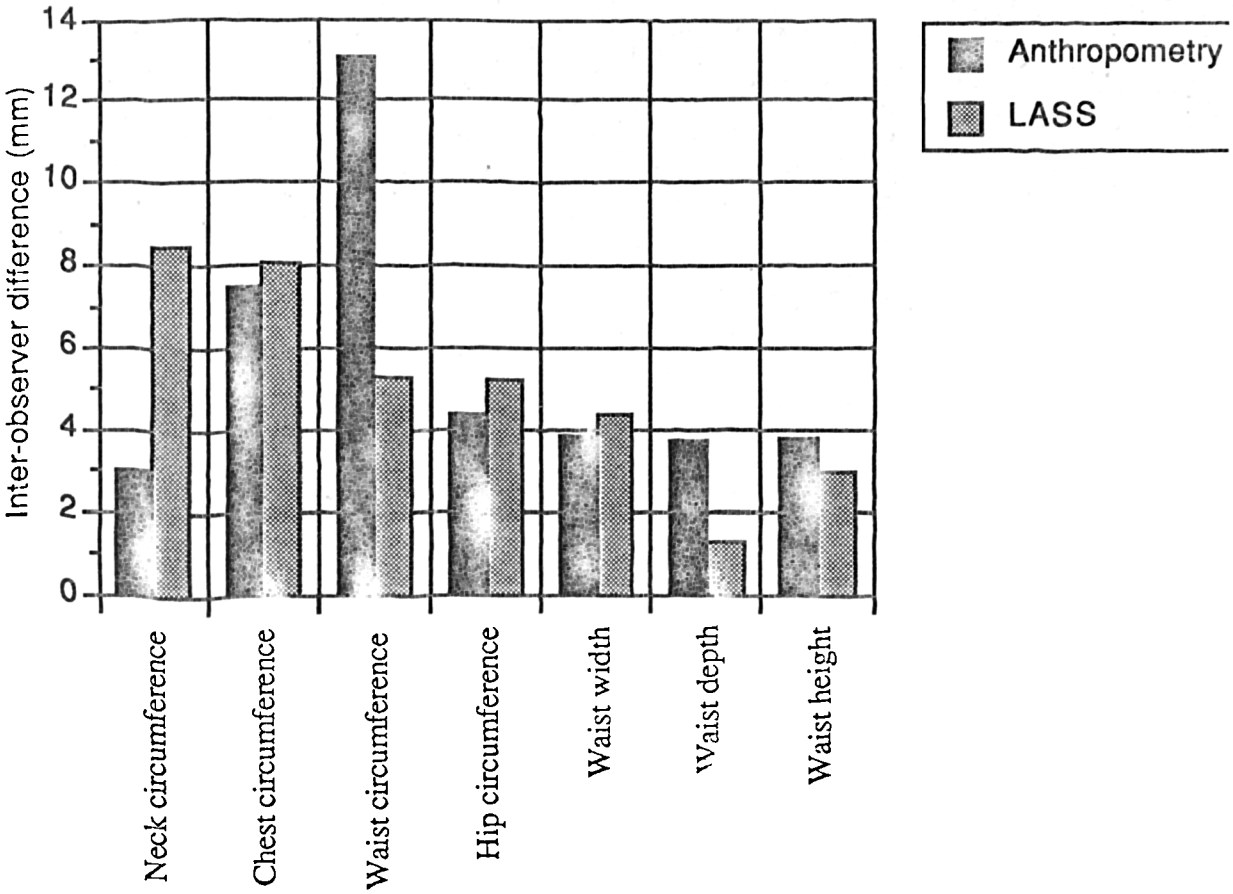


Figure 36 Inter-observer variability in anthropometric and LASS measurements.

As the above results indicate, measurement by the LASS scanner including the considerable data manipulation in creating and using the **shape matrix** gives comparable results to measurements taken using traditional anthropometric methods.

The efficiency of the averaging program

If the computed shape which is the average of many people is to have any relevance to size and shape problems associated with the tailoring trade, then measurements taken from the average person should agree with the average of measurements taken from a number of people. The following table shows a comparison of an average of measurements taken on eight subjects with the same measurements taken on an average of eight subjects.

Filename	Under-bust circumf. (mm)	Maxbust circumf. (mm)	Shoulder to High pt. (mm)	High pt to High pt. (mm)	Underbust height (mm)	Maxbust height (mm)
3b.34b.2.h	743.98	874.22	250.77	161.61	1059	1114
5b.34b.1.h	747.30	900.10	256.37	185.80	1155	1215
6b.34b.1.h	740.40	927.52	260.46	198.06	1202	1254
12b.34b.3.h	781.40	906.67	267.38	188.62	1143	1183
81b.34b.3.h	735.72	907.20	252.80	193.92	1166	1226
88b.34b.1.h	727.79	875.63	267.98	180.50	1121	1167
90b.34b.2.h	726.17	874.75	263.98	186.53	1067	1107
94b.34b.1.h	742.33	914.66	275.49	197.76	1172	1227
MEAN	**743.10**	**897.59**	**261.90**	**186.60**	**1136**	**1187**
34b average	**739.36**	**886.52**	**266.17**	**181.02**	**1133**	**1188**

Table 6 Comparison of Average Measurements with Measurement of an Average.

Discussion.

Measurement of the human form is complex. The method of measurement depends on the measuring data required and the use to which it will be put. If one is making shirts then the data required is significantly different from that required for making gloves. In some instances a whole body measurement is needed and in others the detail of some small part of the body is what is wanted. The ideal solution would be to obtain a whole body scan which was sufficiently detailed that any part could be magnified to give precise measurements of that part with sufficient accuracy for any likely purpose.

Such a scanner has yet to be built and the number of differing approaches to the problem is a measure of the difficulty of such a design. Apart from LASS which is the first successful whole body scanner the other scanners have concentrated on measuring a part of the body for a particular purpose and it would be unreasonable to compare them simply by their accuracy of measurement. Those which scan only one side of the body we might think could be used twice to scan both the front and back views. The two views might then be combined to give an all round view. Turner-Smith(1982), Frobin et al (1983), Gourlay et al (1984), Louis et al (1986). Unfortunately this is not possible unless both views are taken simultaneously because of significant shifts in position and posture between the two scans.

A problem exists for those systems which rotate the subject on a turntable but use only one camera or range finding device to collect the data. Clerget et al (1977), Rioux (1984), Addleman et al (1985), Uesugi (1991). Here the difficulty is the elimination of body sway. Unless the person being measured is restrained in some way the amount of sway will render the measurements too inaccurate for use. At least two cameras or range finders are necessary to measure movement in two orthogonal directions to determine the amount of sway.

A further disadvantage of a single camera system is that it is insufficient to see round horizontal projections such as shoulders, chin or top of the head unless the camera is moved during a scan. This is also true if a single projector is used. While multiple cameras or equivalent scanning devices can be used as in the LASS system, in many cases the cost of specialised cameras is much greater than that of

mass produced television cameras making such an arrangement prohibitively expensive.
Each system must then be compared in the context in which it is used. It is not a fair comparison to compare only the accuracy and resolution of those scanners which measure parts of bodies with the accuracy and resolution of a whole body scanner.

A scanner produces its data in digital form as a series of cylindrical or orthogonal co-ordinates. Because a human body is a generally rounded shape it is sensible to take the data at points which are not too closely spaced and then interpolate between them to provide a continuous surface. A well known technique for this is to use cubic spline interpolation. Several forms of cubic splines have been used such as , Hermite, Bezier, and B-Spline. The latter in the form of Non Uniform Rational B-Splines (NURBS). In the LASS project Cardinal splines were used for interpolation. Cardinal splines have the property that the data points are the control points of the curve and the final curve must therefore pass through the data points. While this can be achieved with the other types of cubic spline the amount of calculation involved is greater.
Fitting the cubic splines to the original scanned data as performed interactively using the BED program is a tedious process. If more cubic spline control points were used than the 16 chosen for the present system then automatic fitting of the points to the data could be achieved with greater accuracy than at present. This would be fine if it was not required to edit out the arms. Since no satisfactory method has been found to determine automatically where an arm starts or finishes the editing of the arms has to be a manual process. The more cubic spline control points used then means that more work is required during the arm editing process in order to move the control points to the required positions. For this reason it was decided that 16 control points was a reasonable compromise.
It might be thought that the editing process because there is an element of human judgement involved, would introduce inaccuracies. The repeatability trials have shown that this is not so and that results can be obtained with accuracy's which are comparable with manual measurements.

It has been shown that the **shape matrix** is a suitable method for the storage of data representing 3-D shapes which are curved in a similar way to the form of the human body. Clearly difficulties would arise if the technique was used for very re-entrant shapes such as those found in engineering. Perhaps the most important application of the method is that it provides the ability to reduce the data from a 3-D scan of a large population to that of an average shape. It also makes possible the measurement of changes of a body when measured at different time intervals even though it was measured in differing positions and postures. Further it provides considerable data reduction which reduces the storage space required for archiving the data.

With the BED computer program it is easy to regenerate data to give any number of cross-sections or any number of points and so facilitate the transfer of data to automatic cloth cutting systems. It may be desirable in the future to represent the traditional manikin in **shape matrix** form as being more suitable for CAD systems.

An important feature is the possibility of using an extension of the method to alter body shapes in a controlled way so that manikins can be made to a size which corresponds to a larger proportion of the population than would an average size. Alternatively bodies can be regenerated from a few simple circumferential measurements. If it is possible in the future to design garments entirely using a computer then the manikin in solid form will no longer be required and its equivalent in the computer could be a **shape matrix** altered as required by regeneration from a few measurements. The figures for the "few simple measurements" could uniquely define a particular size and shape between different manufacturers or between manufacturer and sales outlet.

The LASS scanner while it has proved to be a successful measuring instrument is large and not transportable. This means that for survey work the subjects to be measured must travel to the machine not the other way round, which is a significant disadvantage.

The size of the machine was determined by the need to achieve a sufficient depth of focus by the camera and projector lenses, which is determined by the focal ratio (f number) of the lenses and the lens to subject distance. The only ways of reducing the system size is to either

increase the illumination to allow a higher lens f number to be used or to use mirrors to fold the optical paths.
An alternative is to build a system into a standard container body which, because it is already a rigid structure, could replace the LASS framework. Housed in this way the whole system could then be relatively easily transported.

The prototype LASS scanner uses an electrically driven turntable to rotate the subject being measured. With the current arrangement the turntable rotates in a series of 2.4 degree or 5 degree steps depending on the resolution of the selected scan. This has the effect of slightly twisting the subject as the turntable accelerates and then untwisting them as the turntable decelerates. In practice this has little effect on the measurement since the subject has come to rest immediately before the instant at which the measurement is taken. It would however be more comfortable for the subject and more desirable from a measuring point of view if the turntable were to rotate continuously during a scan. The difficulty in achieving this is the problem of synchronising the turntable with the computer program which controls the collection of the data from the cameras. One method would be to fit the turntable with a rotary shaft encoder so that the computer program could determine the turntable position at any instant during the scan.

Having taken a scan the computer then has to apply the calibration factors to the data in order to build up a total picture from the pictures obtained from the individual cameras. This process takes several minutes to perform and originally was considered to limit the number of people who could be scanned in any one period. A method of speeding up the computation could be to use a number of parallel processors to perform parts of the calculations simultaneously. In practice it was found that the computer could finish its task in the time required for the subject to get undressed and clothed in the close fitting garments ready to be scanned.

The performance of the LASS scanner has generally been very satisfactory in measuring people for tight fitting garments. There are some areas where it does not perform so well. One such area is the scanning of an individual arm. The LASS scanner works by measuring

the radii from points on the surface of the body to the centre of rotation of the turntable. This means that if an all round view of an arm is required then the subject must stand with an arm positioned such that the light stripes projected by the projectors towards the centre of rotation of the turntable lie on the surface of the arm without overlap. The width of the projected lines of light were made sufficiently wide so that the electrical frequency response of the television cameras would not limit the height of the pulse created in response to the camera scanning the light stripe. It is very difficult or impossible with a typical slim arm to maintain a position where the lines of light do not overlap.
Because legs are larger good scans can be obtained by standing the subject with one leg on the centre of rotation of the turntable with the other leg as far away as possible. The masking effect of the other leg is not significant as it subtends a relatively few degrees of the total scan.
Another difficult area is the scanning of feet. This is in the main due to the interaction with the largely horizontal surface of the top of the foot and the 5mm vertical resolution.

In some instances, when making a scan, it is necessary to mark parts of the body so that their positions can be accurately located on the final data when it is displayed on the computer screen. A method was devised whereby a small light on the end of a wand was used to define a location and its position could be recorded by the television cameras. Unfortunately the television cameras without the additional position information derived from the light projectors are only capable of measuring a 2-D scene. This meant that to arrive at a 3-D position the 2-D information has to be mapped onto the 3-D surface obtained from scanning the subject immediately after using the light wand.. This method worked very well for the scanning of static objects but when live subjects were scanned large errors were introduced due to differences in position and posture between the marking phase and the final scan.

Conclusions

The scanner has proved to be reliable over a period of three years since its completion having measured more than 150 people and it is as accurate as traditional tape-measure methods. Its use as a whole body scanner is excelent but it is not intended to measure body parts such as feet and hands. A specialised scanner based on the laser range-finder principle with linear rather than rotational movement would be more appropriate and more versatile for this type of measurement.

When editing the raw data, the method of cubic spline curve fitting gives excellent results and when combined into a **shape matrix** allows for the averaging of several bodies or comparison of one body with another which has not previously been possible. Although the actual process of curve fitting is tedious and takes at least one hour for each body, a more automatic process may be possible in the future.

The re-creation of body shapes from eight circumferences is accurate enough for the manufacture of all but the most tight fitting garments but may be more useful in the future as a replacement for sematotyping. It is perfectly possible to manipulate the eight circumferences to create body torsos of any shape. Conversely a torso of any (almost any!) shape could be defined by eight two digit numbers.

Suggestions for further work.

It is very time consuming to apply the curve fitting algorithm to each subject who has been measured. There is need for a compul r program which will perform this task automatically.
The present scanner is large and unwieldy and not easily transported. The size was largely determined by the depth of focus requirements of the camera. Perhaps it is possible to work with different lenses or a folded optical path to achieve a more transportable device.

REFERENCES

Addleman D., and Addleman L., (1985) Rapid 3D Digitizing. *Computer Graphics World* **8**, 42-44.

Arridge S.R., Moss J.P., Linney A.D., and James D.(1985) Three Dimensional digitization of the face and skull. *Journal of Maxillofacial Surgery* ; **13**, 136-43.

Bartels R.H., Beatty.J.C., and Barsky B.A., (1987) *An Introduction to Splines for use in Computer Graphics and Geometric Modelling.* (Morgan Kaufmann Publishers, Inc. 95 First St., Los Altos, California 94022).

Brooke-Wavell K., Jones P.R.M.J., and West G.M., (1993). Reliability and repeatibility of 3-D body scanner (LASS) measurements compared to anthropometry. *Submitted to Journal of Computer Aided Design.*

Burke P.H., Banks P., Beard L.F.H., Tee J.E., and Hughs C. (1983) Stereophotographicmeasurement of change in facial soft tissue morphology following surgery, *British Journal of Oral Surgery,* **21**, 237-45

Clerget M., Germain F., and Kryze J. (1977) Process and apparatus for optically exploring the surface of a body. *United States Patent 829,936* (United States Patent and Trademark Office Sep.1,.)

Frobin W., and Hierholzer E., (1983) Automatic Measurement of Body Surfaces Using Rasterstereography. *Photogrammetric Engineering and Remote Sensing.,* **49**, 377-384

Gourlay A.R., Kaye G., Dennison D.M., Peacock A,J. and Morgen M.D.(1984) Analysis of an optical mapping technique for lung function. *Computers in Biology and Medicine* **14**, 47-58.

Halioua M, Krishnamurthy R. S., Liu H, and Chiang F. P.(1983) Projection moiré with moving gratings for automated 3-D topography. *Applied Optics* **22**, 850-855.

Ishida A., Mori Y., Kishimoto H., Nakazima T., and Tsubakimoto H. (1987) Body shape measurement for scoliosis evaluation. *Medical & Biological Engineering & Computing.*, **25**, 583-585.

Ito I.(1979) Apparatus for measuring the contour configuration of articles.*U.K.Patent G.B 2030286 b. Jul. 20* (British Patent Office London.)

Kerninghan and Ritchie (1978.) *The C Programming Language* (Prentice-Hall).

Leventhal L.A., Hawkins D., Kane G., and Cramer W.D.,(1986) *68000 Assembly Language Programming.* (2nd Edition Osbourne McGraw-Hill, Berkeley,California.)

Lewis J.R.T. and ,Sopwith T.(1986) Three dimensional surface measurement by computer. *Image and Vision Computing* ,**4**, 159-166.

Magnant D.(1985) Capteur tridemensional sans contact. *Proceedings of the Society of Photo-Optical Instrumentation Engineers,* **602**, 18-22.

Meadows D.M., Johnson W.O., and Allen J.(1970). Generation of surface contours by Moiré patterns. *Applied Optics,* **9**, 942-947.

McCartney J., and Hinds B. K .(1992) Computer aided design of garments using digitised three-dimensional surfaces. *Proc Instn Mech Engrs* **206**, 199-206

Reid G. T., Rixon R. C., Marshall S. J., and Stewart H. (1986). Automatic on-line measurements of three-dimensional shape by shadow casting moiré topography. *Wear,* **109**, 297-304.

Rioux M. (1984). Laser range finder based on synchronised scanners. *Applied Optics,* **23**, 3837-3844.

Sheldon W.H., Stevens S.S., and Tucker W.B.,(1940). *The Varieties of Human Physique.* (Harper & Brothers New York and London.)

Smith N.S.H.,Jones P.R.M.J., and West G.M.(1990) 3-D Scanning: A new tool in the study of human body composition. *Ann. of Human Biol.*,**17**, 341-351.

Takada M. and Esaki T.(1981). Method and apparatus for measuring human body or the like *U.K. Patent G.B. 2069690 B. Jan 26.* (British Patent Office, London.)

Turner-Smith A.R.(1982) Television scanning technique for topographic body measurements. *Proceedings of the Society of Photo-Optical Instrumentation Engineers,* **361**, 1-5.

Uesugi M. (1991) Three-Dimensional Curved Shape Measuring System Using Image Encoder. *Journal of Robotics and Mechatronics* ,**3**, 190 - 95.

Vietorisz T.(1964). Improvements in or relating to the scanning of objects to provide indications of shape. *U.K. Patent 1,078,108 Dec. 16.* (British Patent Office, London.)

West G.M.(1987) Loughborough Anthropometric Shadow Scanner. *M.Phil thesis* . (Loughborough University of Technology. Leics. U.K.)

West G.M. and Jones P.R.M.(1985) Making Measurements on a body *British Patent 85.24473. Oct. 4.* (British Patent Office, London. 1985.)

APPENDIX 1

LASS Specification

1.	Maximum height of subject		2.1 m
2.	Maximum radius of subject		0.3 m
3.	Angular increment between measurements		2.4 degrees
		a.	5.0 degrees
	or	b.	
4.	Vertical spacing between measurements		5.0 mm
		a.	10..0 mm
	or	b.	
5.	Maximum radial error taken on a static subject anywhere within the vertical distance	2 s.d.	2.1 m 1.96 mm

Camera Specification

Make	Panasonic
Camera type number	WV-1550/B
Power source	240V AC, 50Hz, 9W
Scanning:	625 lines/50 fields/25 frames
Synchronizing: Internal:	CCIR standard or vertical line locked (switchable internally) External (Gen-lock) Composite video signal Composite sync signal Horizontal drive(HD) and vertical drive (VD) pulse.
Video output level	1.0Vp-p composite / 75 ohms (BNC)
Horizontal resolution at centre	650 lines
Signal to noise:	More than 44 db (AGC off)
Pick-up tube:	2/3" separate mesh Newvicon Type S4102

Required illumination	minimum	0.03 footcandle(0.3 lux) with F1.4 lens under incandescent light
Ambient temperature	operating	-18 ~ +50 degrees celsius
Lens Mounting		Standard C mount

Lens Specification (Camera)

Make	FUJINON-TV
Focal length	50mm
Focal ratio	1:1.4
Focal ratio setting	1:4.0

Lens Specification (Projector).

Make	FUJINON-TV
Focal length	16mm.
Focal ratio	1:1.4
Focal ratio setting	1:2.0

Projector Lamp

WOTAN XENOPHOT HLX 64627 A1/231
12 volt 100 watt

Computer hardware

Make	ELTEC elektronic mainz GmbH Galileo-Galilei-Str. 11 Postfach 65 D-6500 Mainz 42 , West Germany
Bus	VME

The following computer cards were used to build the required system:

EUROCOM 5-1	Processor with 16MIIz 68020 CPU, 1Mbyte RAM.
RAM 512-2/68K	2 Mbyte additional RAM
SL20-1.2/68K	Slave processor with 16 MHz 68020 CPU 1Mbyte RAM + MC68881 Maths Co-processor
HRG-2	High resolution graphics interface
APAL-1.2/68K	2X24 I/O ports & two MC68230 timer units
IPIN Ethernet Controller	Interface for Ethernet.
80 Mbyte Hard disk unit	
3 1/2 inch Floppy disk unit	
Tape streamer	
Mouse-100	Serial Mouse
Televideo 905 Terminal	
Panasonic TX-2002 High resolution colour CRT display	

Computer software.

OS-9/68K Operating system Version 2.1
OS-9/68K Assembler/Linker/Debugger Version 2.8
OS-9/68K C Compiler Revision G June 1987.

Photometer

Supplier:	Ealing Electro-Optics plc., Greycaine Road, Watford WD2 4Pw
Digital portable radiometer	part no. 28-0974
Flat response filter/detector	part no. 28-0966
Variable bandpass filter set	part no. 35-2567

Turntable.

Supplier:	Unimatic Engineers Ltd., 122 Granville Road, Cricklewood, London NW2 2LN
Rotary table	RTL 250 NES
Stepping motor	4266
Drive unit	CD20
Power supply	PM1200
Transformer	TO92
Rack	RA12/232
RS232 interface	IF1

Trap calling Routines.

```
            NAM  newapalset.a
            ttl  trap call prog.
            use      /dd/defs/oskdefs.d

            psect

trapno      equ  5
intname     dc.b     "qtrapo",0

gset:       movem.l  d4/a0/a1,-(a7)
            move.w   #trapno,d0          trap number to
                                         assign
            move.l   #0,d1               no optional memory
                                         override
            lea      ((intname).l,pc),a0 get address of int.
                                         prog.
            os9      F$TLink             link to trap handler
            bcs.s    wrong               check if ok
            movem.l  (a7)+,d4/a0/a1
            moveq    #0,d1
            rts
gcall:      movem.l  a2/a5,-(a7)
            movea.l  d0,a5               get array address
            lea      ((CAMINFO).l,a6),a2 get camera list
                                         address
            tcall    trapno,0            trap to prog
            bcs.s    wrong1
            movem.l  (a7)+,a2/a5
            rts
ccall       movem.l  a2/a5,-(a7)
            movea.l  d0,a5               get array address
            lea      ((CAMINFO).l,a6),a2 get camera list
                                         address
            tcall    trapno,1            trap to prog
```

```
         bcs.s     wrong1
         movem.l   (a7)+,a2/a5
         rts
scall:   tcall     trapno,2          trap to prog
         bcs.s     wrong1
         rts
pcall:   tcall     trapno,3          trap to prog
         bcs.s     wrong1
         rts
c2call:  lea       ((port2c).l,a6),a2   get address of data
                                        space
         tcall     trapno,4          trap to prog
         bcs.s     wrong1
         rts
wrong:
         movem.l   (a7)+,d4/a0/a1
wrong1:  os9       F$Exit
unpack:  movem.l   a4/a5,-(a7)
         movea.l   d0,a5             address of byte to
                                     unpack
         lea       ((BYTES).l,a6),a4 get address of
                                     unpack list
         move.l    (a5),d1           get data
         rol.l     #8,d1             move next bits to
                                     lsb
         rol.l     #2,d1             move next bits to
                                     lsb
         move.l    d1,d0
         andi.l    #$3ff,d0          mask to 10 bits
         move.l    d0,(a4)+          save radius
         rol.l     #8,d1             move next bits to
                                     lsb
         rol.l     #1,d1             move next bits to
                                     lsb
         move.l    d1,d0
         andi.l    #$1ff,d0          mask to 9 bits
         move.l    d0,(a4)+          save line count
```

```
            rol.l       #8,d1               move next bits to
                                            lsb
            move.l      d1,d0
            andi.l      #$ff,d0             mask to 8 bits
            move.l      d0,(a4)+            save slice count
            rol.l       #5,d1               move next bits to
                                            lsb
            move.l      d1,d0
            andi.l      #$1f,d0             mask to 5 bits
            move.l      d0,(a4)             save camera info
            movem.l     (a7)+,a4/a5
            rts
packcart:   movem.l     d5/a4/a5,-(a7)
            movea.l     d0,a5               address of byte to
                                            pack
            lea         ((BYTES).l,a6),a4   get address of
                                            unpack list
            move.l      (a4)+,d0            get x
            andi.l      #$3ff,d0            mask to 10 bits
            move.l      d0,d5               put in d5
            rol.l       #6,d5               make room for next
                                            bits
            rol.l       #6,d5               make room for next
                                            bits
            move        (a4)+,d0            get y
            andi.l      #$fff,d0            mask to 12 bits
            or.l        d0,d5               insert y into d5
            rol.l       #5,d5               make room for next
                                            bits
            rol.l       #5,d5               make room for next
                                            bits
            move.l      (a4)+,d0            get z
            andi.l      #$3ff,d0            mask to 10 bits
            or.l        d0,d5               insert z into d5
            move.l      d5,(a5)
            movem.l     (a7)+,d5/a4/a5
            rts
unpkcart:   movem.l     a4/a5,-(a7)
```

```
            movea.l   d0,a5              address of byte to
                                         unpack
            lea       ((BYTES).l,a6),a4  get address of
                                         unpack list
            bfexts    (a5){0:10},d0      unpack x
            move.l    d0,(a4)+           put inlist
            bfexts    (a5){10:12},d0     unpack y
            move.l    d0,(a4)+           put in list
            bfexts    (a5){22:10},d0     unpack z
            move.l    d0,(a4)+           put in list
            movem.l   (a7)+,a4/a5
            rts
packpola:   movem.l   d5/a4/a5,-(a7)
            movea.l   d0,a5              address of byte to
                                         pack
            lea       ((BYTES).l,a6),a4  get address of
                                         unpack list
            move.l    (a4)+,d0           get r
            andi.l    #$7ff,d0           mask to 11 bits
            move.l    d0,d5              put r in d5
            rol.l     #5,d5              make room for next
                                         bits
            rol.l     #5,d5              make room for next
                                         bits
            move.l    (a4)+,d0           get theta
            andi.l    #$3ff,d0           mask to 10 bits
            or.l      d0,d5              put theta in d5
            rol.l     #6,d5              make room for next
                                         bits
            rol.l     #5,d5              make room for next
                                         bits
            move.l    (a4)+,d0           get h
            andi.l    #$7ff,d0           mask to 11 bits
            or.l      d0,d5              put h in d5
            move.l    d5,(a5)
            movem.l   (a7)+,d5/a4/a5
            rts
unpkpola:   movem.l   a4/a5,-(a7)
```

```
        movea.l   d0,a5              address of byte to
                                     unpack
        lea       ((BYTES).l,a6),a4  get address of
                                     unpack list
        bfextu    (a5){0:11},d0      unpack r
        move.l    d0,(a4)+           put in list
        bfextu    (a5){11:10},d0     unpack theta
        move.l    d0,(a4)+           put in list
        bfextu    (a5){21:11},d0     unpack h
        move.l    d0,(a4)+           put in list
        movem.l   (a7)+,a4/a5
        rts
        vsect
buffer  ds.b 64
        ends
```

Trap Handling Routines to get Camera Data

```
nam Qtrap trap handler
use /h0/defs/oskdefs.d
use portadd

PSRR1          equ    $00
PSRR2          equ    %00000111
INITP1         equ    $00
INITP2         equ    %00000000
SE1REQ         equ    $00
SE2REQ         equ    $00
PACNTRL1       equ    $70
PACNTRL2       equ    %10000000
PBCNTRL1       equ    $70
PBCNTRL2       equ    %10110000
PCCNTRL1 equ   $00
DOUT           equ    $FF
DIN            equ    $00
A1DATA         equ    %00100000
B1DATA         equ    $00
C1DATA         equ    $02
```

```
A2DATA          equ             $00
B2DATA          equ             $00
PCDIR1          equ             %00001111
PCDIR2          equ             %00000000

type equ (TrapLib<<8)+Objct
revs equ (ReEnt+SupStat<<8)+0
edit            equ             0
stack           equ             0
psect  Qtrap_a,type,revs,edit,stack,Qtrap
dc.l initial
dc.l terminat
_sysedit: equ edit edition number of module

initial     adda.l    #8,a7
            movem.l   a3/a4,-(a7)
            move.l    #PIT1,a3          point to port 1
            move.l    #PIT2,a4          point to port 2
            move.b    #INITP1,PGCNTRL(a3)
            move.b    #PSRR1,PRQSER(a3)
            move.b    #DC UT,PADATD(a3)
            move.b    #DOUT,PBDATD(a3)
            move.b    #PCDIR1,PCDATD(a3)   set direction of c
                                           output  move.b
            #PACNTRL1,PACNTRLR(a3)
            move.b    #PBCNTRL1,PBCNTRLR(a3)
            move.b    #PCCNTRL1,TCNTRL(a3)
            move.b    #A1DATA,PADATR(a3)
            move.b    #B1DATA,PBDATR(a3)
            move.b    #C1DATA,PCDATR(a3)
            move.b    #INITP2,PGCNTRL(a4)
            move.b    #PSRR2,PRQSER(a4)
            move.b    #DIN,PADATD(a4)
            move.b    #DIN,PBDATD(a4)
            move.b    #PCDIR2,PCDATD(a4)   set direction of c
                                           output
            move.b    #PACNTRL2,PACNTRLR(a4)
            move.b    #PBCNTRL2,PBCNTRLR(a4)
```

```
            move.b    #PCCNTRL1,TCNTRL(A4)

            move.b    #PSRR2,PRQSER(a4)
            movem.l   (a7)+,a3/a4
            moveq.l   #0,d1
            rts
terminat    adda.l    #8,a7
            rts

* The stack looks like this:
* +8 callers return PC
* +6 vector #
* +4 func code
* +0 callers a6 reg.

            org       0                stack offset
                                       definitions
S.d0        do.l      1                caller's d0 reg
S.d1        do.l      1                caller's d1 reg
S.d2        do.l      1                caller's d2 reg
S.d3        do.l      1                caller's d3 reg
S.d4        do.l      1                caller's d4 reg
S.d5        do.l      1                caller's d5 reg
S.d6        do.l      1                caller's d6 reg
S.d7        do.l      1                caller's d7 reg
S.a0        do.l      1                caller's a0 reg
S.a1        do.l      1                caller's a1 reg
S.a2        do.l      1                caller's a2 reg
S.a3        do.l      1                caller's a3 reg
S.a4        do.l      1                caller's a4 reg
S.a5        do.l      1                caller's a5 reg
S.a6        do.l      1                caller's a6 reg
S.func      do.w      1                trap function code
S.pc        do.l      1                return pc

* address of average list passed in a1
* address of camera list passed in a2
* address of data list passed in a5
```

```
* slice count passed in d1

Qtrap    movem.l  d0-d7/a0-a5,-(a7)
         movea.l  #PIT1,a3          point to port 1
         movea.l  #PIT2,a4          point to port 2
         move.w   S.func(a7),d1     get function code
         cmp.w    #1,d1
         beq      Ctrap             branch if function
                                    code #1
         cmp.w    #2,d1
         beq      PAtrap            branch if function
                                    code #2
         cmp.w    #3,d1
         beq      PBtrap            branch if function
                                    code #3
         cmp.w    #4,d1
         beq      PCtrap            branch if function
                                    code #4
* here to get data from all cameras in list
         move.l   (a2)+,d3          get camera count
         move.l   S.d1(a7),a0       get slice count
         move     sr,d5             save sr
         move.l   d5,-(a7)          push sr on to stack
         ori      #$700,sr          switch off
                                    interrupts
cloop    move.l   (a2)+,d4          get camera info
                                    from list
         move.l   d4,d5             get edge info
         andi.l   #$ff00,d4         get camera no
         ror.l    #8,d4             move to end of byte
         move.l   d4,a1             save camera no in
                                    a1
         move.l   d5,d4             restore d4
         andi.l   #$f0,d4           camera no. only
         andi.l   #$f,d5            edge no. only
         move.b   PADATR(A3),d0
         andi.l   #$c,d0            get shutter info
         or.l     d0,d4
```

```
        move.l   d4,d7
        move.w   #312,d2          set line count
Qtrap1  btst.b   #4,PSTATR(a4)    bit test h1 frame
                                  sync flag
        beq.s    Qtrap1
        btst.b   #4,PSTATR(a4)    bit test h1 frame
                                  sync flag
        beq.s    Qtrap1
        btst.b   #4,PSTATR(a4)    bit test h1 frame
                                  sync flag
        beq.s    Qtrap1
Qtrap2  btst.b   #4,PSTATR(a4)    bit test h1 frame
                                  sync flag
        bne.s    Qtrap2
Qtrap3  move.l   d7,d4            camera info
         or.l    #$1,d4           mode=1,preset
                                  enable=0
        move.b   d4,PADATR(a3)    select camera
Qloop1  btst.b   #6,PSTATR(a4)    bit test h3 line sync
                                  flag
        beq.s    Qloop1
        btst.b   #6,PSTATR(a4)    bit test h3 line sync
                                  flag
        beq.s    Qloop1
adummy  btst.b   #6,PSTATR(a4)    bit test h3 gone
        bne.s    adummy
        movep.w  PADATR(a4),d6    get address counter
                                  reading
        ror.l    #8,d6
        ror.l    #2,d6
        andi.l   #$f,d6           mask to 4 bits
        move.b   d5,PCDATR(a3)    address edge
        or.l     #$3,d4           mode=1,preset
                                  enable=1
        move.b   d4,PADATR(a3)    select camera
        movep.w  PADATR(a4),d0    get data
        cmpi.l   #5,d6            has it counted 4
                                  edges ?
```

```
        beq.s     Qtrap4
*       cmpi.l    #3,d6          has it counted at
                                 least 2 edges ?
*       bge.s     Qtrap4
        move.l    #0,d0          make dud data
                                   equal 0
Qtrap4  andi.l    #$3ff,d0       mask to 10 bits
        move.l    d0,d1          insert into d1
        rol.l     #8,d1          make room for next
                                 bits
        rol.l     #1,d1          make room for next
                                 bits
        move.w    d2,d0          get line count
        andi.l    #$1ff,d0       mask to 9 bits
        or.l      d0,d1          insert into d1
        rol.l     #8,d1          make room for next
                                 bits
        move.l    a0,d0          get slice count
        andi.l    #$ff,d0        mask to 8 bits
        or.l      d0,d1          insert into d1
        rol.l     #5,d1          rotate to make
                                 room for next byte
        ror.l     #4,d4          get camera number
                                 to lo byte
        move.l    a1,d0          camera number to
                                 d0
        andi.l    #$1f,d0        mask to 5 bits
        or.l      d0,d1          insert into d1
        move.l    d1,(a5)+       store data (a5)
        sub.w     #1,d2          dec. loop count
        bne       Qtrap3
        sub.l     #1,d3          dec camera count
        bne       cloop
        move.l    (a7)+,d5       get sr from stack
        move      d5,sr          restore sr
        movem.l   (a7)+,d0-d7/a0-a5/a6-a7
        rts
* here to get data from 1 camera
```

```
* address of camera list passed in a2
* address of data list passed in a5
* camera number in list passed in d1

Ctrap     movea.l   #PIT1,a3          point to port 1
          movea.l   #PIT2,a4          point to port 2
          move.w    #312,d2           set line count
          move.l    a2,d6             save camera list
                                      address
          move.l    S.d1(a7),d3       get camera number
                                      in list
          move.l    #0,a0             get slice count
          move      sr,d5             save sr
          ori       #$700,sr          switch off
                                      interrupts
ctrap1    btst.b    #4,PSTATR(a4)     bit test h1 frame
                                      sync flag
          beq.s     ctrap1
          btst.b    #4,PSTATR(a4)     bit test h1 frame
                                      sync flag
          beq.s     ctrap1
          btst.b    #4,PSTATR(a4)     bit test h1 frame
                                      sync flag
          beq.s     ctrap1
ctrap2    btst.b    #4,PSTATR(a4)     bit test h1 frame
                                      sync flag
          bne.s     ctrap2
ctrap3    btst.b    #6,PSTATR(a4)     bit test h3 line sync
                                      flag
          beq.s     ctrap3
          btst.b    #6,PSTATR(a4)     bit test h3 line sync
                                      flag
          beq.s     ctrap3
          move.l    (a2,d3*4),d4      get camera info
                                      from list
          move.b    d4,PADATR(a3)     select camera
          movep.w   PADATR(a4),d0     get data
cdummy    btst.b    #6,PSTATR(a4)     bit test h3 gone
```

```
        bne.s     cdummy
        andi.l    #$3ff,d0          mask to 10 bits
        move.l    d0,d1             insert into d1
        rol.l     #8,d1             make room for next
                                    bits
        rol.l     #1,d1             make room for next
                                    bits
        move.w    d2,d0             get line count
        andi.l    #$1ff,d0          mask to 9 bits
        or.l      d0,d1             insert into d1
        rol.l     #8,d1             make room for next
                                    bits
        move.l    a0,d0             get slice count
        andi.l    #$ff,d0           mask to 8 bits
        or.l      d0,d1             insert into d1
        rol.l     #5,d1             rotate to make
                                    room for next byte
        ror.l     #8,d4             get camera number
                                    to lo byte
        move.l    d4,d0             camera number to
                                    d0
        andi.l    #$1f,d0           mask to 5 bits
        or.l      d0,d1             insert into d1
        move.l    d1,(a5)+          store data
        sub.w     #1,d2             dec. loop count
        bne.s     ctrap3
        move      d5,sr             restore sr
        movem.l   (a7)+,d0-d7/a0-a5/a6-a7
        rts
PAtrap  movea.l   #PIT1,a3          point to port 1
        move.b    d0,PADATR(a3)     transfer d0 to port
        movem.l   (a7)+,d0-d7/a0-a5/a6-a7
        rts
PBtrap  movea.l   #PIT1,a3          point to port 1
        move.b    d0,PBDATR(a3)     transfer d0 to port
        movem.l   (a7)+,d0-d7/a0-a5/a6-a7
        rts
PCtrap  movea.l   #PIT2,a3          point to port 1
```

```
move.b    PCDATR(a3),d0
move.b    d0,(a2)              transfer port to (a2)
                               address
movem.l   (a7)+,d0-d7/a0-a5/a6-a7
rts
ends
```

APPENDIX 3

Projector Calculations.

Calculations for a 16 mm f 1.4 lens to project light 1,250 mm with a depth of focus of +/- 125 mm.

Let u = image distance
v = object distance
f = focal length
O = object size
I = image size

Then using standard optical formulae

$$\frac{1}{u}+\frac{1}{v}=\frac{1}{f}$$

$$\frac{1}{1250}+\frac{1}{v}=\frac{1}{16}$$

$$v=\frac{1250 \times 16}{1250 - 16}$$

$$= 16.2 \text{ mm.}$$

Magnification.

$$\frac{O}{I}=\frac{u}{v}=\frac{1250}{16.2} = 77.16$$

For a projected slit height O = 666 mm.

$$I=\frac{666}{77.16} = 8.64 \text{ mm.}$$

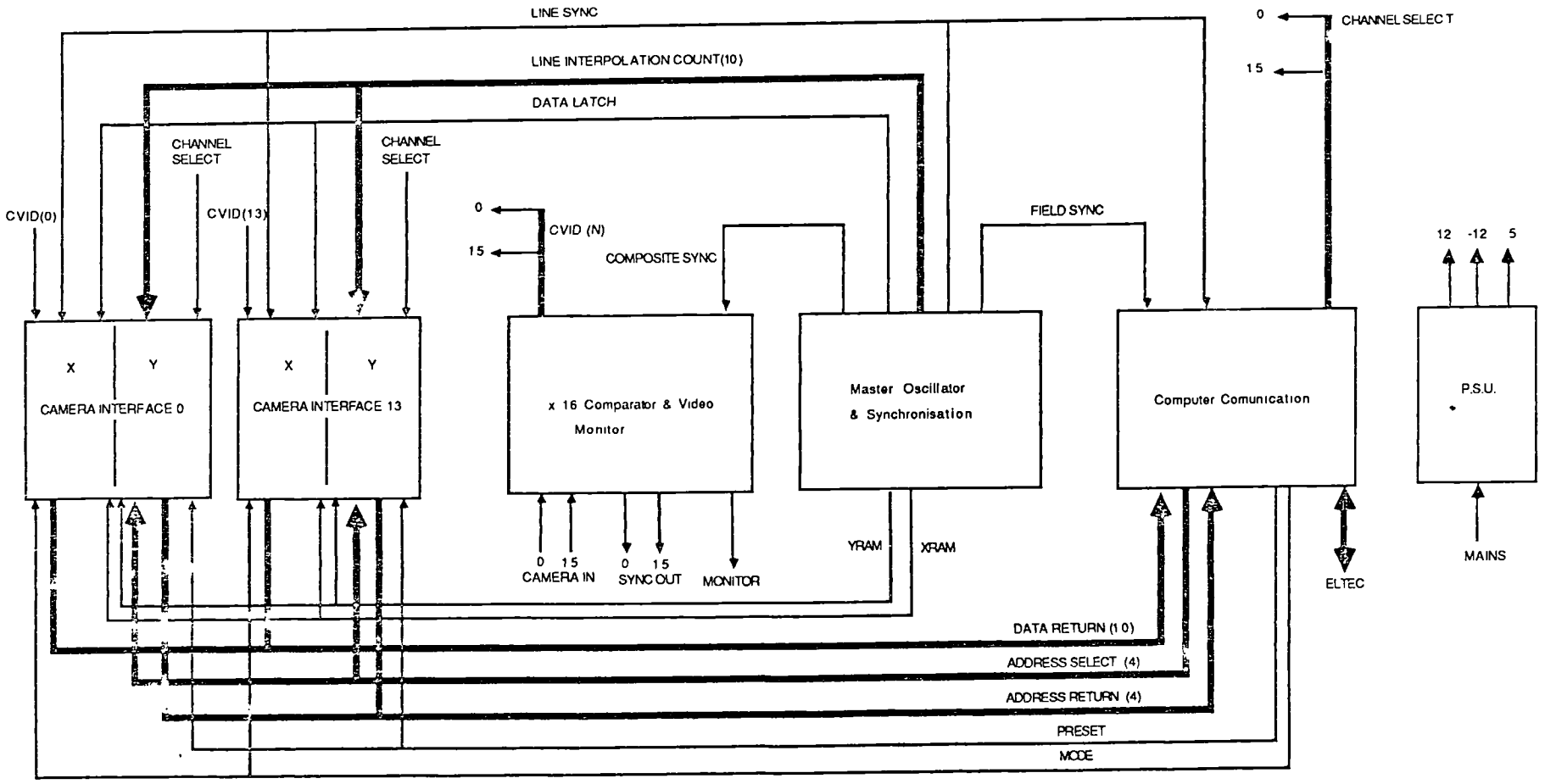
At f = 1.4

Aperture $A = \frac{16}{1.4} = 11.42$ mm.

Circle of confusion @ 125 mm $=\frac{11.42 \times 125}{1250} = 1.14$ mm.

APPENDIX 4

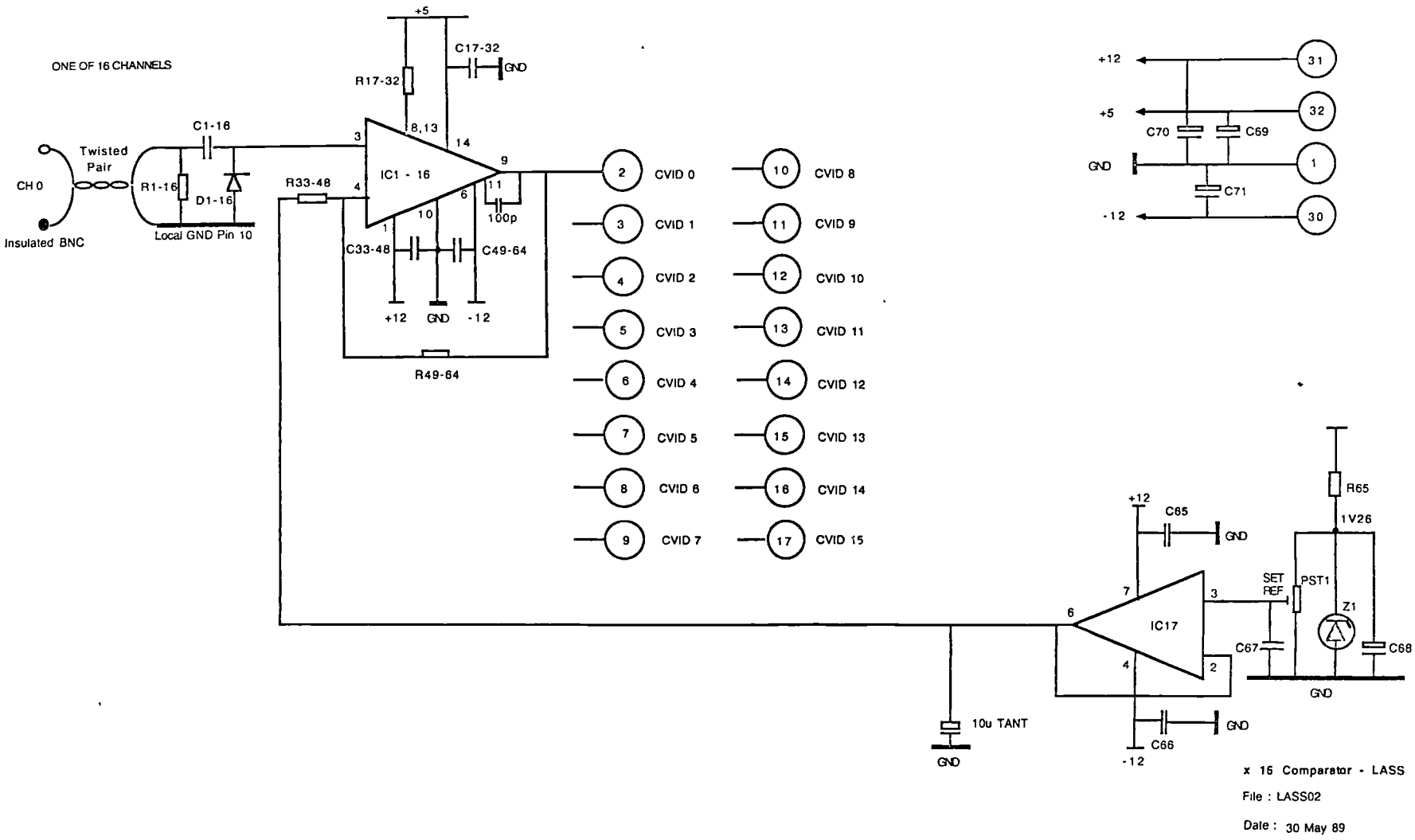
Circuit Diagrams



x 16 Channel Interface - System Schematic

File : LASS01

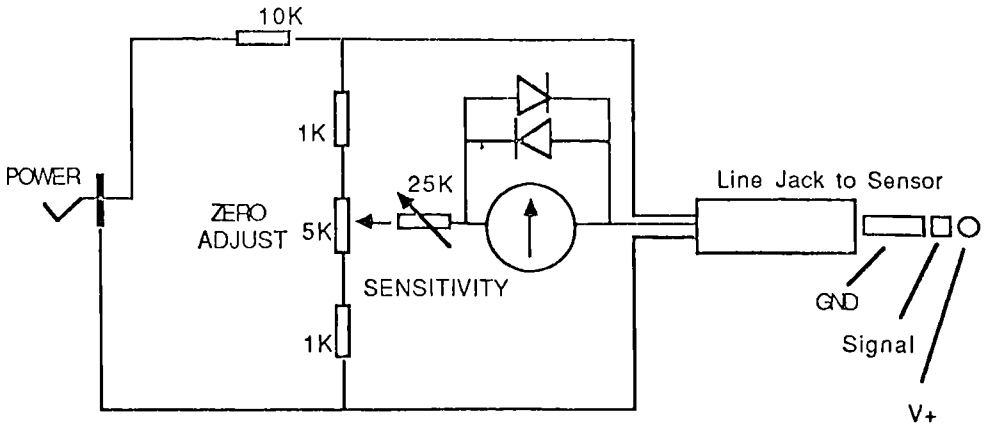
Date : 5 June 89



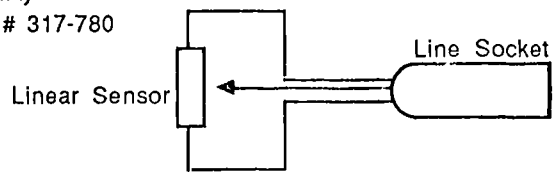
x 16 Comparator - LASS

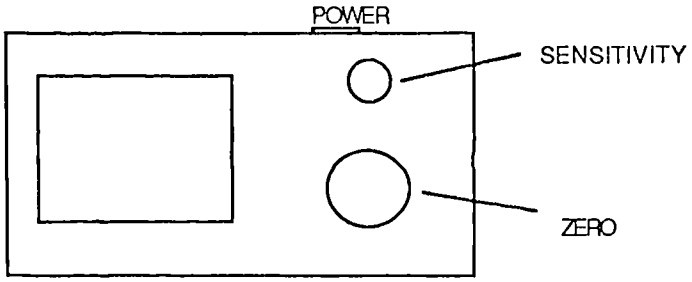
File : LASS02

Date : 30 May 89



NOTES: 1. Meter - 50/0/50 uA
2. Diodes - Shottky
3. Sensor - RS # 317-780

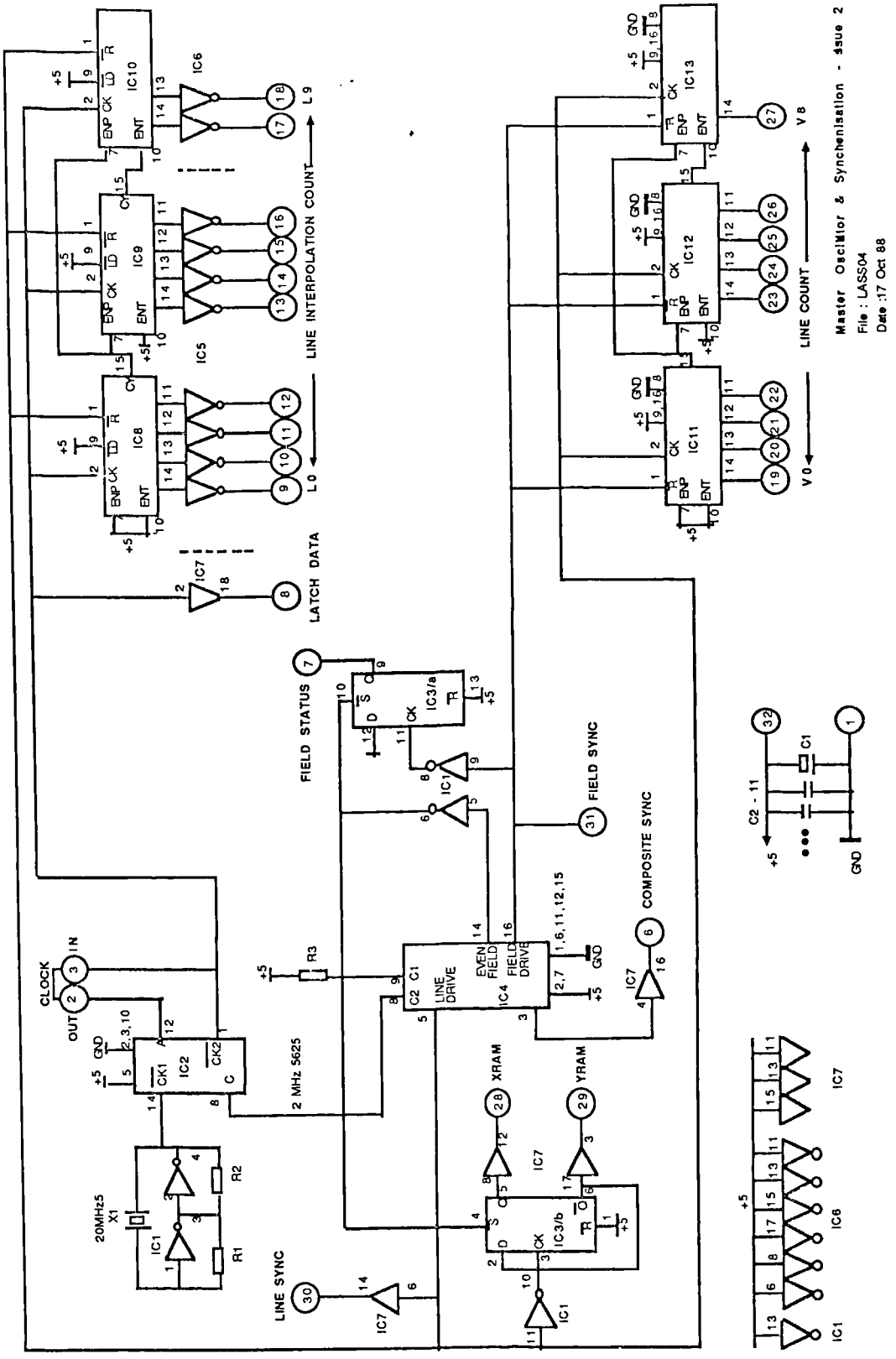




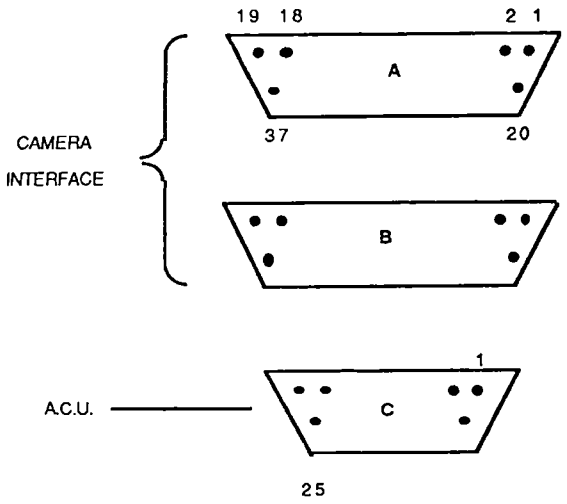
ECCENTRICITY METER - LASS

File : LASS03

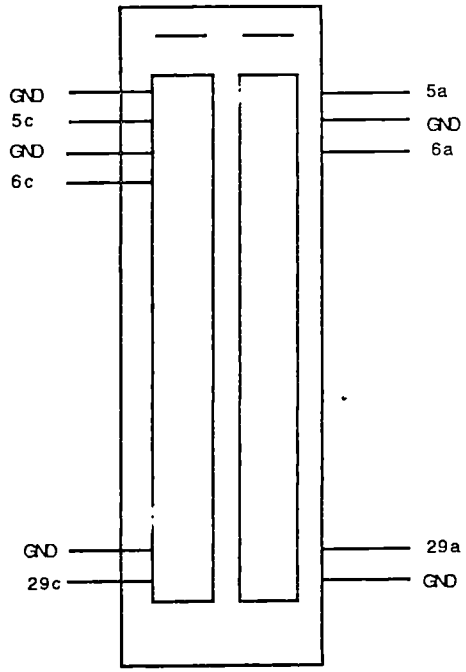
Date : 18 Aug 88


Master Oscillator & Synchronisation - Issue 2
File : LASS04
Date :17 Oct 88
LINE INTERPOLATION COUNT
LINE COUNT
LATCH DATA
FIELD STATUS
FIELD SYNC
COMPOSITE SYNC
LINE SYNC
XRAM
YRAM
2 MHz 5625
20MHz5
CLOCK
IC1
IC2
IC3/a
IC3/b
IC4
IC5
IC6
IC7
IC8
IC9
IC10
IC11
IC12
IC13
R1
R2
R3
C1
C2 - 11
+5
GND

Rear Panel Connectors - Socket View



Eltec I/O Board Rear Connector



GROUNDED

A20 - 37

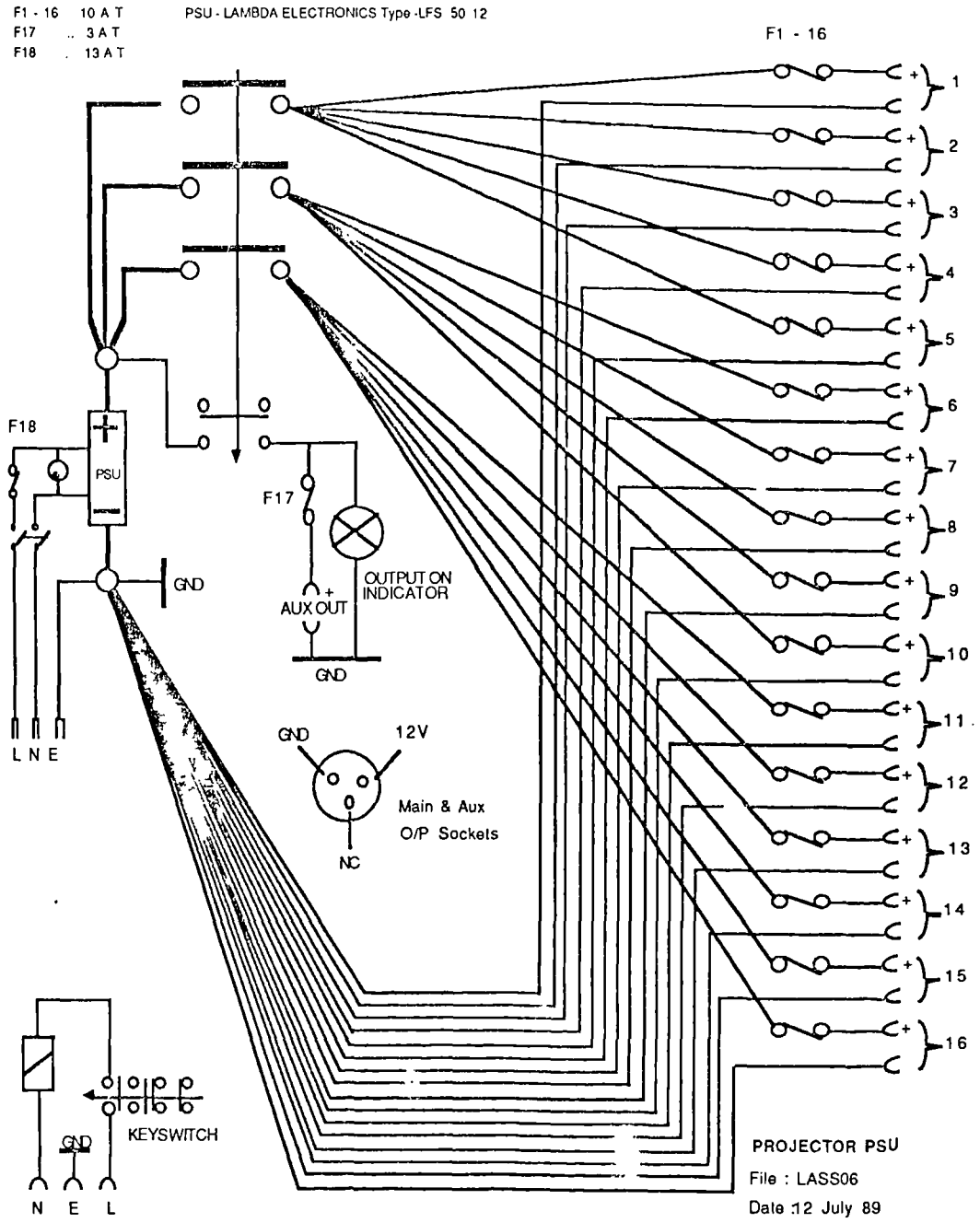
B20 - 37

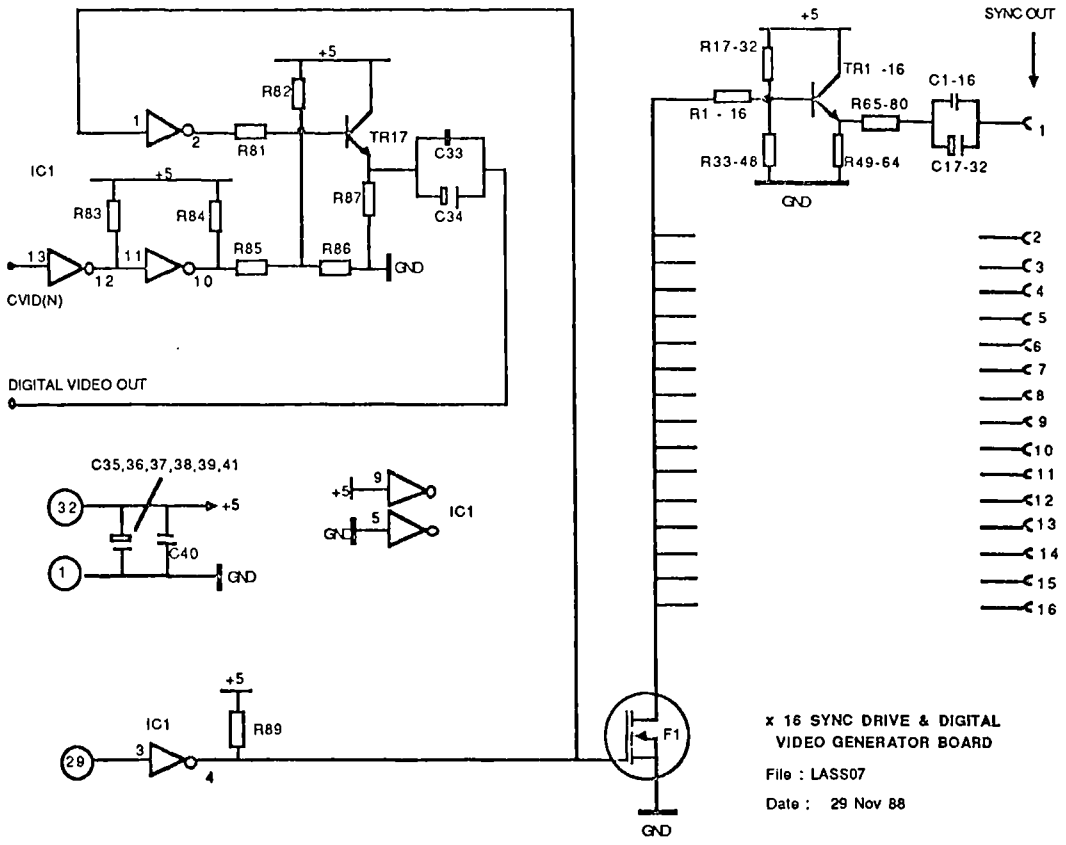
C25

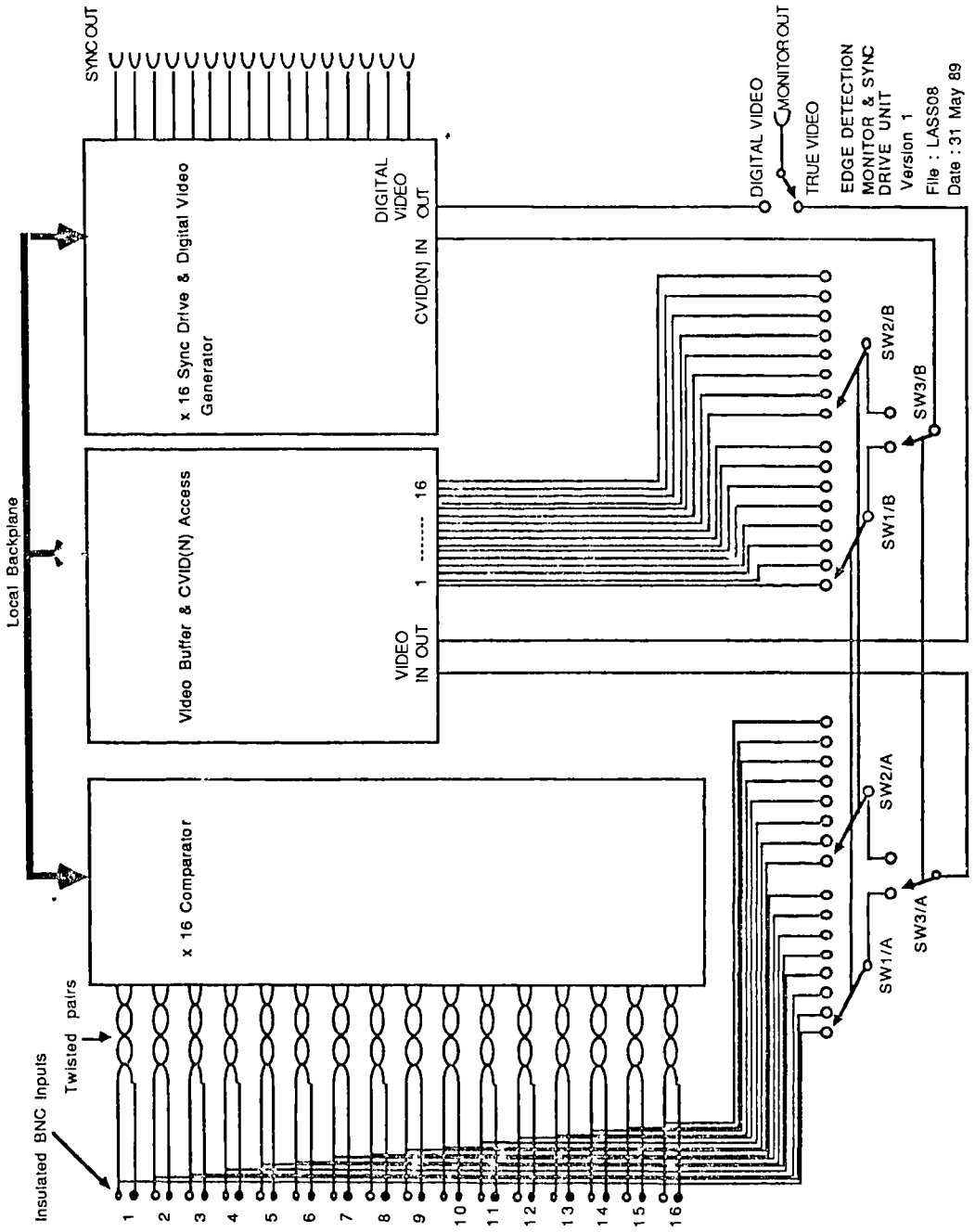
ELTEC I/O ASSEMBLY - LASS

File : LASS05

Date : 25 Aug 88


F1 - 16 10 A T
F17 .. 3 A T
F18 . 13 A T
PSU - LAMBDA ELECTRONICS Type -LFS 50 12
F1 - 16
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
F18
PSU
F17
GND
OUTPUT ON INDICATOR
AUX OUT
GND
L N E
GND
12V
Main & Aux
O/P Sockets
NC
KEYSWITCH
GND
N E L
PROJECTOR PSU
File : LASS06
Date :12 July 89


+5
R82
1
2
R81
TR17
C33
IC1
+5
R83
R84
R87
C34
13
11
R85
R86
GND
12
10
CVID(N)
DIGITAL VIDEO OUT
C35,36,37,38,39,41
32
+5
C40
1
GND
9
+5
IC1
GND
5
+5
IC1
R89
29
3
4
F1
GND
+5
R17-32
TR1 -16
C1-16
SYNC OUT
R1 - 16
R65-80
1
R33-48
R49-64
C17-32
GND
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
x 16 SYNC DRIVE & DIGITAL
VIDEO GENERATOR BOARD
File : LASS07
Date : 29 Nov 88


SYNC OUT
x 16 Sync Drive & Digital Video Generator
DIGITAL VIDEO OUT
CVID(N) IN
Video Buffer & CVID(N) Access
VIDEO IN OUT
1 ------ 16
x 16 Comparator
Local Backplane
Insulated BNC Inputs
Twisted pairs
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
DIGITAL VIDEO
MONITOR OUT
TRUE VIDEO
EDGE DETECTION
MONITOR & SYNC
DRIVE UNIT
Version 1
File : LASS08
Date : 31 May 89
SW1/A
SW2/A
SW3/A
SW1/B
SW2/B
SW3/B

ELTEC I/O CHANNEL 1

INTERNAL SIGNAL NAME		WIRE ID	EXTERNAL SIGNAL NAME			ACCESS CONNECTOR
Port A Bit	0	5a	MODE			A9
	1	6a	PRESET ENABLE			A10
	2	7a	SHUTTER 0			C7
	3	8a	1			C8
	4	9a	CHANNEL SELECT		0	A5
	5	10a			1	A6
	6	11a			2	A7
	7	12a			3	A8
Port B Bit	0	13a	ACU	LED	0	C1
	1	14a			1	C2
	2	15a			2	C3
	3	16a			3	C4
	4	17a			4	C5
	5	18a		MAINS	0	C6
	6	19a		MAINS	1	C11
	7	20a		MAINS	2	C12
H1		21a	*			
H3		22a	*			
Port C Bit	2	23a	ADDRESS SELECT		2	A3
	1	24a			1	A2
H2		25a	*			
H4		26a	*			
Port C Bit	3	27a	ADDRESS SELECT		3	A4
	0	28a			0	A1

ELTEC I/O CHANNEL 2

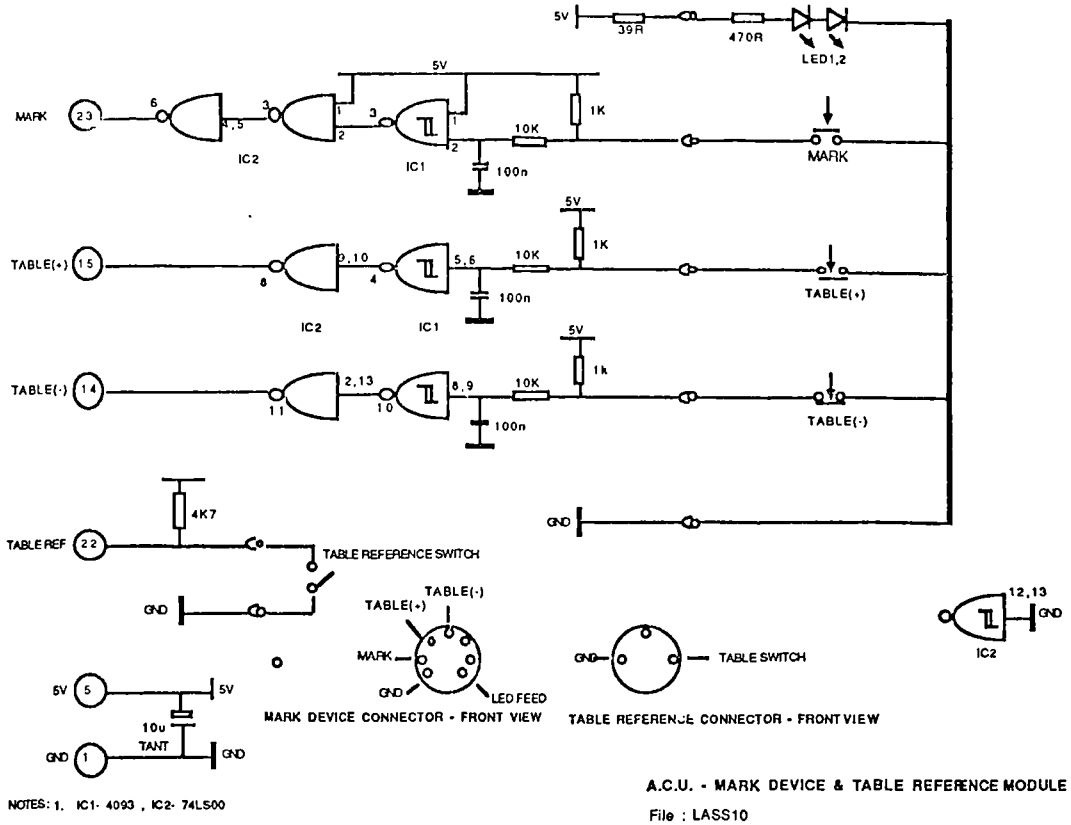
INTERNAL SIGNAL NAME		WIRE ID	EXTERNAL SIGNAL NAME		ACCESS CONNECTOR
Port A Bit	0	29c	DATA RETURN	8	B9
	1	28c		9	B10
	2	27c	ADDRESS RETURN	0	B11
	3	26c		1	B12
	4	25c		2	B13
	5	24c		3	B14
	6	23c	*		
	7	22c	*		
Port B Bit	0	21c	DATA RETURN	0	B1
	1	20c		1	B2
	2	19c		2	B3
	3	18c		3	B4
	4	17c		4	B5
	5	16c		5	B6
	6	15c		6	B7
	7	14c		7	B8
H1		13c	FIELD SYNC		B15
H3		12c	LINE SYNC		B16
Port C Bit	2	11c	ACU - TABLE (+)		C13
	1	10c	ACU-MARK SWITCH		C9
H2		9c	*		
H4		8c	*		
Port C Bit	3	7c	ACU - TABLE (-)		C14
	0	6c	ACU-TABLE REFERENCE		C10

* = Not Used

ELTEC INPUT / OUTPUT DETAIL

File : LASS09

Date : 31 May 89



NOTES: 1. IC1- 4093 , IC2- 74LS00

A.C.U. - MARK DEVICE & TABLE REFERENCE MODULE

File : LASS10

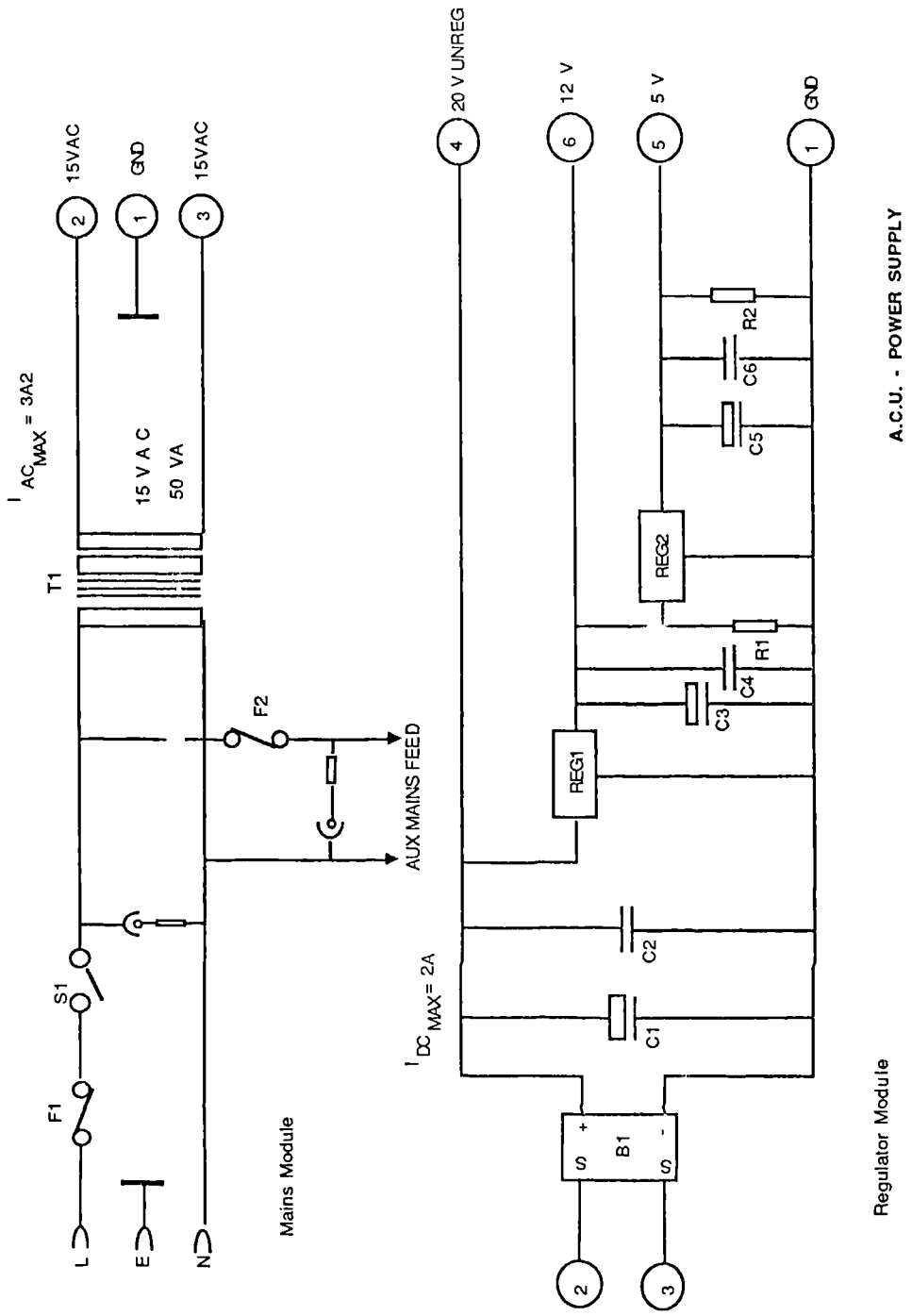
Date : 2 JUL 89

BACKPLANE BUS #	DEFINITION	DATA I/O CONNECTOR (25 way D)	FUNCTION
1	GND	1	GND
2	15VAC		
3	15VAC		
4	20V UNREGULATED		
5	5V		
6	12V		
7	LED DRIVE REGULATOR - FEED		
8	- ADJUST		
9	- OUTPUT		
10			
11			
12			
13			
14	D17	19	TABLE(-)
15	D16	18	TABLE(+)
16	D15	17	MAINS 2
17	D14	16	MAINS 1
18	D13	15	MAINS 0
19	D12	14	SHUTTER 1
20	D11	13	SHUTTER 0
21	D10	12	
22	D9	11	TABLE REFERENCE
23	D8	10	MARK LED
24	D7	9	LED 7
25	D6	8	LED 6
26	D5	7	LED 5
27	D4	6	LED 4
28	D3	5	LED 3
29	D2	4	LED 2
30	D1	3	LED 1
31	D0	2	LED 0
32			

A.C.U. - Internal Connection Detail

File : LASS11

Date : 23 Jun 89


A.C.U. - POWER SUPPLY
File : LASS12
Date : 28 Sep 88
Mains Module
Regulator Module
15VAC
GND
15VAC
15 V A C
50 VA
T1
F1
S1
F2
AUX MAINS FEED
20 V UNREG
12 V
5 V
GND
REG1
REG2
B1
C1
C2
C3
C4
C5
C6
R1
R2
L
E
N

APPENDIX 5

Source code for "newsnapy" program.

```
/* lassdefs */
#include <stdio.h>
#include <math.h>
#define RESOLUTION  1 /* high=1 low=0 */
#if RESOLUTION
#define ANGLE 2.4 /* angular increment */
#define STEPS 150 /* 360/ANGLE steps per rev */
#define VSTEP 5   /* vertical step height */
#define NARRAY 126000 /* NFRAME*STEPS*2 size of normalised
data array */
#define SLSPACE 30 /* STEPS/VSTEP */
#define NFRAME 420 /* no. of vertical intervals in 2.1 metres
2100/VSTEP */
#define PRPLOT printf("\033n\033T20")  /* sets line feed scale on
printer */
#define DENS 2 /* density function for patches */
#define ARMTOL 15 /* limit for arm removal */
#else
#define ANGLE 5.0 /* angular increment */
#define STEPS 72 /* 360/ANGLE steps per rev */
#define VSTEP 10 /* vertical step height */
#define NARRAY 30240 /* NFRAME*STEPS*2 size of normalised data
array */
#define SLSPACE 15 /* STEPS/VSTEP */
#define NFRAME 210 /* no. of vertical intervals in 2.1 metres
2100/VSTEP */
#define PRPLOT printf("\033n\033T40")  /* sets line feed scale on
printer */
#define DENS 1 /* density function for patches */
#define ARMTOL 20 /* limit for arm removal */
```

```
#endif
#define ARRAY FRAME*CAM*2*STEPS /* memory required for raw
data */
#define CAM 14      /* no. of cameras */
#define CAMDIST 2900 /* camera to centre distance */
#define FRAME 312   /* no of lines per frame */
#define TOPBLANK 25 /* no of hidden lines at top of screen */
#define STEPREV 72000 /* no of steps per rev */
#define ARDIM (FRAME-20)*2
#define ZERO 75 /*57*/
#define XMAX 500 /* max measuring range in X direction. */
#define YMAX 400 /* max measuring range in Y direction. */
#define CALBLK 140 /* size of calibration block per camera */
#define THETA1 CAL1[121+CALBLK*(camsl-1)] /* angle of leading
shadow */
#define THETA2 CAL1[122+CALBLK*(camsl-1)] /* angle of trailing
shadow */
#define SINE1  CAL1[123+CALBLK*(camsl-1)] /* sine of THETA1 */
#define SINE2  CAL1[124+CALBLK*(camsl-1)] /* sine of THETA2 */
#define COS1   CAL1[125+CALBLK*(camsl-1)] /* cos of THETA1 */
#define COS2   CAL1[126+CALBLK*(camsl-1)] /* cos of THETA2 */
#define ZERO1  CAL1[127+CALBLK*(camsl-1)] /* zero of shadow 1
#define HEIGHT CAL1[129+CALBLK*(camsl-1)] /* height of lowest
calibration light */
#define KK CAL1[130+CALBLK*(camsl-1)]     /* camera distance */
#define NPOINT (angl*NFRAME+hite)*2 /* point to array */
#define XCOUNT 590 /* scale range in X direction. */
#define YCOUNT 312 /* scale range in Y direction. */
#define CZERO 180   /* center zero offset */
#define PI 3.141592654
#define RAD PI/180
#define HEADSIZE 140 /* no of elements in file header */
#define SCALE 50 /* window to plot scale factor */
#define PLUSX 10 /* window scale */
#define MINUSX -10 /* window scale */
#define PLUSY 7.5 /* window scale */
#define MINUSY -7.5 /* window scale */
#define PLUSZ 10 /* window scale */
```

```
#define MINUSZ -10 /* window scale */
#define BLANK 0
#define TRANSX 1
#define TRANSY 2
#define TRANSZ 3
#define ROTX 4
#define ROTY 5
#define ROTZ 6
#define ZOOM 7
#define RESET 8
#define XFILE 9
#define EXITTRANS 10
#define CURVE 11
#define XYZERO 12
#define LINES 13
#define IRIS 14
#define IRIS2 15
#define SKIN 16
#define VOLUME 17
#define X1SLCT 19
#define X2SET 20
#define Y1SET 21
#define Y2SET 22
#define FB 23
#define SETZERO 24
#define SSAVE 25
#define PPLOT 26
#define ARMS 27
#define MARKS 28
#define OBLIQUE 29
#define SET1 30
#define SET2 31
#define DPX0 34
#define DPY0 35
#define DP1 36
#define DP2 37
#define DP3 38
#define DP4 39
```

```
#define DP5 40
#define DP6 41
#define DP7 42
#define DP8 43
#define DPX9 44
#define DPY9 45
#define UP 47
#define UP5 48
#define FITP 49
#define DOWN 50
#define SAVE 52
#define PSAVE 53
#define PLOAD 54
#define RECALL 55
#define DELTAX 56
#define DELTAY 57
#define PPLOT2 58
#define PMPRNT 59
#define DPZERO 60
#define CFIT 61
#define SPLINE 62
#define PCLEAR 64
#define DAT1 65
#define DAT2 66
#define SMES 67
#define PBPLOT 68
#define SET3 69
#define SET4 70
#define TGL 71
#define INP 72
#define PTCH 73
#define LPP 74
#define LPSAVE 75
#define LPLOAD 76
#define LCLEAR 77
#define ROTPX 78
#define ROTPY 79
#define ROTPZ 80
```

```
#define ROTMX 81
#define ROTMY 82
#define ROTMZ 83
#define TRANSPX 84
#define TRANSPY 85
#define TRANSPZ 86
#define TRANSMX 87
#define TRANSMY 88
#define TRANSMZ 89
#define ZOOMP 90
#define ZOOMM 91
#define SPANG 92
#define MIRROR 93
#define GETSL 94
#define COUNT 95
#define PORT_INIT
```

/* snap.c */

```
#include "lassdefs"

        remote unsigned   MEMSP[ARRAY]; /*main
                                  data storage area */
        short NDATA[NARRAY];        /* normalised data
                                          array */
        extern int mouse_port,mouse_x,mouse_y;
        extern char mouse_buttons;
        int BYTES[4];         /* common for passing parms. */
        int CAMINFO[2*CAM+1];
        double TAPE[STEPS+1][2];        /* temp store for tape
                                        measurements */
        float HEADER[HEADSIZE];         /* file header info */
        double CAL1[CALBLK*CAM];    /* calibration array
                                        X,Y,Xslope,Yslope */
        extern int *hrgbase;          /* graphics pointer */
        extern char *font_0;          /* graphics character font */
        double sintheta,costheta; /*theta is angle for current
                                  shadow */
```

```
double w_scale,x_scale,y_scale; /* window scale factors */
int linetype,colour,wndow,zoom; /* graphics parameters */
double rad,offset,kk,smooth;
int camera,height;
int write(path,buffer,count);
char *buffer,pt;
int avflag,armflag,linesp,steps,cflag,listflg,pltflg;
int po;
unsigned char port2c;
double xsum,ysum,zsum,xyzcount,anglcount;
float data;
unsigned count;
int slicearray[STEPS+5][2];
double slicexy[4*NFRAME][2];
```

main()

```
{
int gset(),camsl,c;
extern char pc;
hrgbase = (int*)0xFF800000;

printf("LASS-Control program.\n");
 usefnt(font_0);
gset();         /* set up interface prog */
listflg=1;      /* select camera list at start up */
selcam();
printf("Switch selector fully anticlockwise.\n");
printf("Switch motor drive off and on.\n");
printf("Type space <cr>\n");
while((c=getchar()) != ' ');
tstart();
camera=0;
linesp=1;
w_scale=1.67;       /* aspect ratio of screen */
x_scale=((double)600/400); /* real space x axis 0-400 */
```

```
    y_scale=((double)450*w_scale/300); /* real space y axis 0-300
                                                                */
    window(1);
    window(2);
    window(3);
    window(4);
    window(5);
    kk=CAMDIST;       /* camera distance */
    rad=PI/180;
    lodcal();       /* load calibration */
    cflag=1;
    avflag=1;
    armflag=0;
    anglcount=0;
    smooth=1;
    pt=0xff;        /* cal light interface byte */
    pintoff(255);
    soff(12);
    init_mouse();
    select();
    finito_mouse();
    }


table()
    {
    printf("0 = Not used.\n");
    printf("1 = List one frame.\n");
    printf("2 = Read mean slice.\n");
    printf("3 = Plot cartesian.\n");
    printf("4 = Flat plot 1 frame & clear.\n");
    printf("5 = Flat plot 1 frame without clear.\n");
    printf("6 = Select table control.\n");
    printf("7 = List directory of /h0/newdata.\n");
    printf("8 = Print array.\n");
    printf("9 = Print multiple slices.\n");
    printf("a = Plot front view.\n");
    printf("b = Plot slice & clear.\n");
    printf("c = Plot slice without clear.\n");
```

```
        printf("d = Print calibration.\n");
        printf("e = Use wand.\n");
        printf("f = Toggle correction flag.\n");
        printf("g = Collect data & process later.\n");
        printf("h = Print slice data to a file.\n");
        printf("i = Wand print.\n");
        printf("j = Save data file.\n");
        printf("k = Load data file.\n");
        printf("l = Not used.\n");
        }


page2()
        {
        printf("m = Set height.\n");
        printf("n = Select camera.\n");
        printf("o = Select no. of cameras in the system.\n");
        printf("p = Control calibration lights.\n");
        printf("q = Enter shadow angle.\n");
        printf("r = Toggle arm flag.\n");
        printf("s = Save data to a text file.\n");
        printf("t = List data.\n");
        printf("u = Toggle the projector.\n");
        printf("v = Set zero.\n");
        printf("w = Toggle average flag.\n");
        printf("x = Plot on printer.\n");
        }


select()
        {
        int f,angl,hite,c,sliceget(),arprnt(),cograb1(),xx,camsl;
        int projflg,temp;
        double angletemp,zro;
        extern int zercal1,zercal2;
        table();
        while((c=getchar()) != ' '){
        switch (c) {
         case '0':
        break;
```

```
case '1':
printf("0=allow for angle.1=direct measurement.Enter
                                                    0/1.");
scanf("%d",&f);
if(f==1)f=-3;
temp=(CAMINFO[camera] & 12);        /* get projector
                                                    bits.*/
pinton(32);
son(temp);
delay(20);
initarry();
sliceget(0);
pintoff(32);  /* switch projector off */
if(cflag==1){
angl=conorm(0,f);
grid(3);
colour=4;
plot1(angl,-1);
printa(angl,f);
        }
 else{
        grid(4);
 colour=4;
 plot2(0,0,0,-1);
 arprnt(0);
        }
break;
 case '2':
temp=(CAMINFO[camera] & 12);        /* get projector
                                                    bits. */
pinton(32);  /* switch projector on */
son(temp);
delay(20);
        f=-3;
camsl=camera/2+camera%2;     /* find camera number
                             not list number */
hite=HEIGHT/VSTEP;
initarry();
```

```
angl=cograb1(f);
        grid(3);
 colour=4;
 plot1(angl,f);
printa(angl,f);
pintoff(32);  /* switch projector off */
 xx=meansl(angl,hite);
break;
 case '3':
 clrall();
        grid(6);
cartplot();
mousexyz();
break;
 case '4':
pinton(32);  /* switch projector on */
delay(20);
if(cflag==1){
        grid(3);
initarry();
angl=cograb1(0);
pintoff(32);  /* switch projector off */
 colour=4;
 plot1(angl,0);
        }
 else{
sliceget(0);
pintoff(32);  /* switch projector off */
 colour=4;
plot2(0,0,0,-1);
        }
break;
 case '5':
pinton(32);  /* switch projector on */
delay(20);
if(cflag==1){
angl=cograb1(1);
pintoff(32);  /* switch projector off */
```

```
 colour=4;
 plot1(angl,1);
        }
 else{
sliceget(0);
pintoff(32);   /* switch projector off */
 colour=4;
plot2(0,0,0,-2);
        }
break;
 case '6':
        select3();
break;
 case '7':
system("dir /h0/newdata");
break;
 case '8':
 printa(steps);
break;
 case '9':
  printm();
break;
 case 'a':
 clrall();
  grid(1);
  grid(5);
xyplot2(0);
mousepnt();
break;
 case 'b':
  printf("Enter height.");
  scanf("%d",&hite);
hite=hite/VSTEP;
grid(2);
splot(hite);
break;
 case 'c':
  printf("Enter height.");
```

```
 scanf("%d",&hite);
hite=hite/VSTEP;
splot(hite);
break;
 case 'd':
 if(camera==0)camsel();
 calpr();
break;
 case 'e':
 wand1();
break;
 case 'f':
if(cflag==0)cflag=1;
else cflag=0;
printf("Correction flag = %d\n",cflag);
break;
 case 'g':
 raw150();
break;
 case 'h':
slicesav();
break;
 case 'i':
 wandprnt();
break;
 case 'j':
newsavcal();
break;
 case 'k':
newlodcal();
 cflag=1;
 break;
 case 'l':
 pintoff(64);
break;
 case 'm':
camsl=camera/2+camera%2;     /* find camera number
                             not list number */
```

```
 printf("Height=%d\n",height);
 printf("Enter height.");
 scanf("%d",&height);
 HEIGHT=height;
break;
 case 'n':
 camsel();
break;
 case 'o':
 selcam();
break;
 case 'p':
 calite();
break;
 case 'q':
camsl=camera/2+camera%2;     /* find camera number
                              not list number */
printf("Theta=%g\n",CAL1[122-
camera%2+CALBLK*(camsl-1)]/rad);
printf("Enter new angle.");
scanf("%E",&angletemp);
angletemp=angletemp*rad;
CAL1[122-camera%2+CALBLK*(camsl-1)]=angletemp;
if(angletemp>PI)angletemp=angletemp-PI;
CAL1[124-camera%2+CALBLK*(camsl-1)]     =sin(angletemp);
CAL1[126-camera%2+CALBLK*(camsl-1)]     =cos(angletemp);
break;
 case 'r':
 if(armflag==0){
 armflag=1;
  printf("armflag=1\n");
 }
 else {
 armflag=0;
  printf("armflag=0\n");
 }
 break;
  case 's':
```

```
 textsav();
break;
 case 't':
listdata();
break;
 case 'u':
if(projflg!=0){
pintoff(32);  /* switch projector off */
soff(12);     /* switch shutters off */
projflg=0;
        }
else{
pinton(32);  /* switch projector on */
 son(12);     /* switch shutters on */
projflg=1;
        }
break;
 case 'v':
camsl=camera/2+camera%2;     /* find camera number
                                not list number */
printf("Zero=%g\n",CAL1[128-camera%2+   CALBLK*(camsl-
1)]);
printf("Enter new zero value.");
scanf("%E",&zro);
CAL1[128-camera%2+CALBLK*(camsl-1)]=zro;
break;
 case 'w':
 if(avflag==0)avflag=1;
 else avflag=0;
  printf("avflag=%d\n",avflag);
break;
 case 'x':
 plotprint();
break;
 case 'y':
        table();
break;
 case 'z':
```

```
            page2();
        break;
            default:
         printf("Type 'y' for page 1.& 'z' for page 2 of the      menu.\n");
        break;
        }
        }
        }

calselect()
        {
        int cc,hite;

        while((cc=getchar()) != ' '){
            switch (cc) {
         case '0':
            cal1();
        break;
         case '1':
         savcal();
        break;
         case '2':
         lodcal();
            cflag=1;
        break;
         case '3':
         shadow();
        break;
            default:
        caltable();
        break;
            }
            }
            }

caltable()
        {
        printf("0 = Calibrate.\n");
```

```
        printf("1 = Save calibration.\n");
        printf("2 = Load calibration.\n");
        printf("3 = Calibrate shadow angle.\n");
        printf("Type space to return to the main program.");
        }
arzero()
        {
        int i;
        unsigned *ap;

        ap=&MEMSP[0];
        for(i=0;i<ARRAY;i++){
        *ap=0;
        ap++;
        }
        }
/* here to set camera selection */
selcam()
{
         int lstno;

         if(listflg<1 || listflg>13)lstno=2;
         else if(listflg==1)lstno=1;
         else if(listflg==6 || listflg==7)lstno=2;
         else if(listflg==8 || listflg==9)lstno=3;
         else if(listflg==10 || listflg==11)lstno=4;
         else if(listflg==4 || listflg==5)lstno=5;
         else if(listflg==12 || listflg==13)lstno=6;
         else if(listflg==2 || listflg==3)lstno=7;
         printf(" List number is %d \n",lstno);
         printf(" Select camera list (1-6).\n");
         printf(" 1=14 cameras X 2 projectors alternate from
                        bottom, for calibration only.\n");
         printf(" 2=Levels 0 to 1\n");
         printf(" 3=Levels 2 to 3\n");
         printf(" 4=Levels 4 to 6\n");
         printf(" 5=Levels 0 to 4\n");
         printf(" 6=Levels 2 to 6\n");
```

```
        printf(" 7=Levels 0 to 6\n");
        scanf("%d",&lstno);
        if(lstno==1)listflg=1;
        else if(lstno==2)listflg=6;
        else if(lstno==3)listflg=8;
        else if(lstno==4)listflg=10;
        else if(lstno==5)listflg=4;
        else if(lstno==6)listflg=12;
        else if(lstno==7)listflg=2;
        if(listflg==1){
        camset(1);
        printf(" There are %d cameras in the system.\n",
       CAMINFO[0]);
        }
        else camset(listflg);
}
/* print calibration array */

calpr()
        {
        int i,camsl;
        double *cl1;

        camsl=camera/2 + camera%2;    /* get camera number
                                         not list no. */
        cl1 = &CAL1[CALBLK*(camsl-1)]; /* point to final
                                         calibration array. */
         for (i=0;i<120;i=i+4)
        printf("%6.6f        %6.6f  %6.6f
        %6.6f\n",*(cl1+i),*(cl1+i+1),*(cl1+i+2),*(cl1+i+3));
        printf("cc=%6.2f Theta1=%6.2f
        Theta2=%6.2f\n",CAL1[120+CALBLK*(camsl-1)],
        CAL1[121+CALBLK*(camsl-1)]*
        180/PI,CAL1[122+CALBLK*(camsl-1)]*180/PI);
        printf("Sine1=%6.2f Sine2=%6.2f ",
        CAL1[123+CALBLK*(camsl-1)],CAL1[124+CALBLK*(camsl-1)]);
```

```
        printf("Cos1=%6.2f
        Cos2=%6.2f\n",CAL1[125+CALBLK*(camsl-1)],
        CAL1[126+CALBLK*(camsl-1)]);
        printf("Zero1=%6.2f Zero2=%6.2f ",
        CAL1[127+CALBLK*(camsl-1)], CAL1[128+CALBLK*(camsl-1)]);
        printf("Height=%6.2f KK=%6.2f\n",HEIGHT,KK);
        }
/* print array */

arprnt(f)
        int f;
        {
        int i,j,k,l;
        int n;
        unsigned *np;
        l=0;
        j=slicepoint(0,0);
        k=slicepoint(1,0);
        for(i=j;i<k;i=i+4*linesp){
        if(l%2==0)printf("\n");
        unpack(i);
        printf("%6d",BYTES[0]);
        printf("%6d",BYTES[1]);
        printf("%6d",BYTES[2]);
        printf("%6d ",BYTES[3]);
        l++;
                }
        printf("\n");
                }
/* get system clock */

clock()
        {
        int _sysdate(format,time,date,day,tick);
        int time,date,tick;
        unsigned short day,tock;

        _sysdate(3,&time,&date,&day,&tick);
```

```
        tock =          tick;
        return(time*100+tock);
        }

delay(hunsecs)
        int hunsecs;
        {
        int time,finish;
        finish=clock()+hunsecs;
        do{time=clock();}while(time<=finish);
        }
/* call timer with "timer(clock()+delay)" delay is in 1/100 secs */

timer(endtime)
        int endtime;
        {
        int time;
        do{time=clock();}while(time<=endtime);
        }
/* here to set slice width */

linespace()
        {
        extern int linesp;
        printf("Enter line spacing.");
        scanf("%d",&linesp);
        }
/* get 150 slices of raw data & then process later */

raw150()
        {
        int pp,angl,camsl,psel,dd;
        int i,interval;
        double theta;

        pp=32;          /* switch projector on */
        pinton(pp);
```

```
        arzero();
        initarry();
        if(camera==0)dd=14;
        else dd=15;
        dd=15;
        for(i=0;i<STEPS;i++){
        printf("%4d\n",i);
        sliceget(i);
        interval = tstep(dd);
        timer(clock()+interval);
        }
        pintoff(pp);  /* switch projector off */
        printf("Now normalising data.\n");
        for(i=0;i<STEPS;i++){
        conorm(i,i);
        }
        }
/* here to return mean of 1 vertical slice */

meansl(angl,hite)
                int angl,hite;
        {
                double xsum,xyzcount;
                double xx,l;
                int radius,htmax,camsl;
                short *np;

        xsum=0;
        xyzcount=0;
        htmax=hite+YMAX/VSTEP-15;
        for(hite=hite+15;hite<=htmax;hite++){        /* do rows */
        np=NPOINT;        /* point to array */
        xx=*np;
        l=*(np+1);
        if(l>0){
        xx=xx/l;
        xsum=xsum+xx;
        xyzcount++;
```

```
        }
        }
        radius=xsum/xyzcount;
        printf("mean radius=%d\n",radius);
        return(radius);
        }
/* here to initialise NDATA[] */

initarry()
        {
        int i;
        for(i=0;i<=NARRAY;i++)NDATA[i]=0;
        }
/* her · to print multiple slices of array */

printm()
        {
        int start,finish,i;
        printf("Enter starting slice.");
        scanf("%d",&start);
        printf("Enter finishing slice.");
        scanf("%d",&finish);
        for(i=start;i<=finish;i++){
        printf("Slice no.=%d\n",i);
        printa(i,0);
        plot1(i,-1);
        }
        }

printa(angl,f)
         int angl,f;
         {
         int r,n,hite,htmax,camsl,avrg;
         double phi,theta;
         short *np;

        if(f<0)f=0;
```

```
        hite=height/VSTEP;
        htmax=hite+YMAX/VSTEP;
        for(;hite<=htmax;hite++){       /* do rows */
        np=NPOINT;      /* point to array */
        r=*np;
        n=*(np+1);
        if(hite%3==0)printf("\n");
        if(n!=0)avrg=r/n;
        else avrg=0;
        printf("%5d %5d %5d %5d       ",hite*VSTEP,r,n,avrg);
         }
          printf("\n");
         }


bell0
{
        char bb;

        bb='\007';
        putchar(bb);
      }
```

```
/* camera.c */
#include "lassdefs"

        extern remote unsigned MEMSP[ARRAY];    /* main
                                        data storage area */
        extern short  NDATA[NARRAY];   /* normalised data
                                            array */
        extern int BYTES[4];                /* common for passing
                                                parms. */
        extern int CAMINFO[2*CAM+1];
        extern int camera,height;
        extern int avflag,cflag,colour,linesp,listflg;
        extern double rad,offset,kk,smooth;
        extern double CAL1[CALBLK*CAM];  /* calibration
                                    array X,Y,Xslope,Y,slope */
```

```
        extern double sintheta,costheta; /* theta is angle for
                                            current shadow */
        extern double anglcount;
        extern unsigned char port2c;
        int CALCAM[CAM+1][2];
        extern float HEADER[HEADSIZE]; /* file header info */


/* here to grab and correct using CAL1[] table */


conorm(slice,f)
        int slice,f;
        {
        int i,j,k,l,r,xx,yy,aa,xc,yc,camsl,camno;
        int step,row,hite,angl,caloff;
        double *cl1,x,y,z,ycorr,zro;
        double theta,denom,costheta,sintheta;
        short *np;

   printf("%4d\n",slice);
        xc=(XCOUNT-ZERO)/6;
        yc=(YCOUNT-TOPBLANK)/6;
        caloff=xc/2;            /* offset of calibration frame */
        j=slicepoint(slice,0);
        k=slicepoint(slice+1,0);
          for(i=j;i<k;i=i+4){
          unpack(i);                    /* get raw data */
          r =  BYTES[0]-ZERO-caloff;    /* uncorrected radius. */
          l = BYTES[1];                   /* original line no. */
                if(l==YCOUNT){
                camno=BYTES[3];  /* camera list number. */
                printf("%d\n",camno);
                camsl=camno/2 + camno%2; /* get camera number
                                                    not list no. */
                cl1= &CAL1[CALBLK*(camsl-1)]; /* point to
                                          correction block */
                height=HEIGHT;
                theta=CAL1[122-camno%2+CALBLK*(camsl-1)];
```

```
                /* from 121 or 122 according to camera */
        sintheta=CAL1[124-camno%2+CALBLK*(camsl-1)];
                /* from 123 or 124 according to camera */
        costheta=CAL1[126-camno%2+CALBLK*(camsl-1)];
                /* from 125 or 126 according to camera */
              zro=CAL1[128-camno%2+CALBLK*(camsl-1)];
                /* from 127 or 128 according to camera */
               if(f==-3){
               theta=0;
               sintheta=0;
               costheta=1;
               }
              angl=slice+theta/rad/ANGLE;
                                        /* integer angular step */
              if(angl<0)angl=0;
              if(angl>=STEPS)angl=angl-STEPS;
              }
        else if(l<=(YCOUNT-TOPBLANK) && l>0 && r>0){
        yy=l/yc;                                /* box y value */
        y=l%yc;                                 /* y value in box */
        xx=r/xc;                                /* box x value */
        x=r%xc;                                 /* x value in box */
        aa=24*xx+4*(5-yy);          /* pointer to box data */
/*printf("x=%g,y=%g,xs=%g,ys=%g\n",*(cl1+aa),*(cl1+aa+1),*(cl1+aa+2
),*(cl1+aa+3));*/
        y=*(cl1+aa+1)+*(cl1+aa+3)*y;
                                /* new y value in real space */
        x=*(cl1+aa)+*(cl1+aa+2)*x-CZERO-zro;
                                /* new x value in real space */
        if(camno%2==1 && x<0){
        continue;
        }
        else if(camno%2==0 && x>0){
        continue;
        }
          denom=(double)kk*costheta+x*sintheta;
         x=kk*x/denom; /* x now = real radius */
         if(x<0 && f==-3)x=x*-1;
```

```
        z=-x*sintheta;          /* Z value */
        ycorr=(y-YMAX/2)*(kk+z)/kk;
              /* calculate error due to being nearer to camera */
        if(y >-1  && angl>-1 && angl<=STEPS){
        y=YMAX/2+ycorr+height;
        hite=y/VSTEP; /* integer vertical inc.(from zero) */
        np=NPOINT; /* point to array */
         if(np>&NDATA[0] && np<&NDATA[NARRAY-1]){
        if(x>0){
         *np=*np+x;                          /* R value */
         np++;
         *np=*np+1;                        /* increment counter */
        }
        }
        }
        }
       }
       return(angl);
       }
/* here to set up calibration array */
    .
cal1()
       {
       unsigned i,k,l;
       int f,j,m,n,o,p,r,rf,t,x,xc,yc;
       int xx1,xx2,yy1,yy2,camsl,caloff;
       int rr,nn,cc,temp,camra;
       unsigned *xk;
       double *cl1;
       double rad[2],lin[2];
       double xslope,yslope,xorigin,yorigin;
          double x_scale,y_scale;
          double ht;
          int window;
          int a,b,c,d;

       ht=0;
       xc=(XCOUNT-ZERO)/6;
```

```
yc=(YCOUNT-TOPBLANK)/6;
caloff=xc/2;            /* offset of calibration frame */
camset(1);  /* select sequential camera list. */
printf("Enter end to calibrate. (0 or 180) degrees.");
scanf("%d",&temp);
if(temp==0)calset(0);
else calset(1);
printf("Enter camera to calibrate. (0=bottom 6=top)");
scanf("%d",&temp);
  for(o=temp;o<=6;o++){
 cflag=0;
 x=0;
 j=0;
 t=0;
 clrall();
 grid(4);
 colour=7;
 setcol(colour);
 camera=CALCAM[o][0];
 height=CALCAM[o][1];
 camra=CAMINFO[camera]/16;
 camra=camra & 0017;
 printf(" Camera no. is %d(%d)\n",o,camra);
 printf("   at height=%d\n",height);
       while(x<5){
        if(j!=0){
        t++;
         printf("Retry no. %d\n",t);
              if(t>5){
              printf("Bad calibration ! Repeat column %4d,
                              camera %d(%d)",x,o,camra);
       printf("\nEnter column to calibrate (0-4).5=quit.");
              scanf("%d",&x);
              t=0;
              }
        }
         pintoff(31);                  /* switch off lights */
         switch(x){
```

```
case 0:
pintoff(31);                    /* switch off lights */
pinton(1);
break;
case 1:
pintoff(31);                    /* switch off lights */
pinton(2);
break;
case 2:
pintoff(31);                    /* switch off lights */
pinton(4);
break;
case 3:
pintoff(31);                    /* switch off lights */
pinton(8);
break;
case 4:
pintoff(31);                    /* switch off lights */
pinton(16);
break;
default:
break;
}
xx2=x*80;           /* x coordinate of ref 2. */
xx1=xx2+40;                 /* x coordinate of ref 1. */
f=1;
sliceget(0);
k = slicepoint(0,0);        /*  point to original data. */
l= slicepoint(1,0);
ccall(k,camera);
plot1(f,f);
l=k+FRAME;
camsl=camera/2 + camera%2;
                /* get camera number not list no. */
cl1 = &CAL1[CALBLK*(camsl-1)];
                /* point to final calibration array. */
cl1=cl1+24*x;       /* move address up 1 block.*/
```

```
/*printf("camsl=%d camera=%d camra=%d
      x=%d\n",camsl,camera,camra,x);*/
            j=12; /* number of calibration points to look for.*/
            m=5;                /* count rows */
            rr=0;               /* total radii */
            nn=0;               /* total lines */
            cc=0;               /* point counter */
            p=0;          /* array pointer */
                for(i=k;i<l;i=i+4){
                unpack(i);
                r = BYTES[0]-ZERO;
                                /* get uncorrected radius */
                n = BYTES[1];
/*printf("x*xc+caloff=%d r=%d\n",x*xc+caloff,r);*/
            if(r>(x*xc+caloff) && r<((x+1)*xc+caloff)){
                        /* found a calibration point */
                        if(cc==0){
                        nn=n;   /* line no. of first point */
                        rr=r;   /* radius of first point */
                        rf=r;   /* save first point */
                        cc=1;   /* count adjacent points */
                        }
                  else if(r>=(rf-3) && r<=(rf+3)){
            nn=nn+n;/* sum lines nos of adjacent points */
                   rr=rr+r;     /* sum radii of adjacent points */
                   cc++;        /* count adjacent points */
                   }
                }
                else if(cc>0){ /* next point not adjacent */
                rad[p]=rr/cc-(x*xc+caloff);
                /* get average radius - corner of square */
                lin[p]=nn/cc-(m*yc);
                /* get average line no - corner of square */
/*printf("j=%d m=%d x=%d p=%d cc=%d rr=%d r=%d rf=%d
          nn=%d  n=%d\n",j,m,x,p,cc,rr,r,rf,nn,n);*/
                p++;
                rr=0;
                nn=0;
```

```
                    cc=0;
                    j--;
/*
    printf("rad[0]=%6.6f  rad[1]=%6.6f\n",rad[0],rad[1]);
    printf("lin[0]=%6.6f  lin[1]=%6.6f\n",lin[0],lin[1]);
*/
                    if(p>=2 && m>=0){
                    yy2=m*60;  /* y coordinate of ref 2. */
                    yy1=yy2+45;         /* y coodinate of ref 1. */
                        if(rad[0]-rad[1]==0){
                        printf("Radius=zero problem ?\n");
                        break;}
                         if(lin[0]-lin[1]==0){
                          printf("Line=zero problem ?\n");
                         break;}
                         xslope=40/(rad[0]-rad[1]);
                         yslope=45/(lin[0]-lin[1]);
                         xorigin=xx2-rad[1]*xslope;
                         yorigin=yy2-lin[1]*yslope;
                          printf("cl1=%d\n",cl1);
                         *cl1=xorigin;
                         cl1++;
                         *cl1=yorigin;
                         cl1++;
                         cl1++;
                         *cl1=yslope;
                         cl1++;
                         p=0;
                         m--;
                         }
                        }
                    }
                    if(j==0){
                    x++;
                        if(x>5){
                        printf("\n");
                        break;
                        }
```

```
                    t=0;
                    }
                  }
                pintoff(31);
      HEIGHT=height; /* height of lowest calibration light */
                KK=CAMDIST;     /* camera distance */
                cflag=1;
      printf("End of camera %d(%d) calibration.\n",o,camra);
   printf("Do you want to continue calibration?(1=yes 0=no)");
                scanf("%d",&temp);
                if(temp !=1)break;
                }
              }
/* here to determine the angle of the shadow. */


shadow()
      {
      double strad,r,x,y,inc;
      double kk,theta,phi;
      int f,i,lc,o,camsl,q,angl,hite,temp;

 if(cflag==0){
      printf("Correction flag not set.\n");
      return;
      }
      initarry();
      camset(1);  /* select sequential camera list. */
      printf("Enter quadrant.(1-4).");   /* input quadrant no. */
      scanf("%d",&q);
                  switch(q){
                  case 0:
                  printf("Wrong quadrant.\n");
                  break;
                  case 1:
                  printf("1st Quadrant.\n");
                  calset(0);
                  break;
                  case 2:
```

```
                printf("2nd.. Quadrant.\n");
                calset(0);
                break;
                case 3:
                printf("3rd. Quadrant.\n");
                calset(1);
                break;
                case 4:
                printf("4th. Quadrant.\n");
                calset(1);
                break;
                default:
                printf("Wrong quadrant.\n");
                break;
                }
printf("Set standard radius of 282 mm. to quadrant
                                        %d\n",q);
strad=282;
printf("Enter camera to calibrate. (0=bottom 6=top)");
scanf("%d",&temp);
camset(1);  /* select sequential camera list. */
kk=CAMDIST;
lc=linesp;
linesp=1;
f=0;
 for(o=temp;o<=6;o++){
 camera=CALCAM[o][0];
 height=CALCAM[o][1];
 if(q==2 || q==4)camera=camera+1;
 camsl=camera/2 + camera%2;
                        /* get camera number not list no. */
 CAL1[122-camera%2+CALBLK*(camsl-1)]=0;
                /* into 121 or 122 according to camera */
 pinton(32);  /* switch projector on */
 temp=(CAMINFO[camera] & 12);/*get projector bits.*/
 son(temp);
 delay(100);
 angl=cograb1(-3);
```

```
pintoff(32); /* switch projector off */
printf("angl=%d  height=%d\n",angl,height);
theta=1;
inc=0.00001;
hite=height/VSTEP;
x=meansl(0,hite);
printf("meansl= %g\n",x);
r=100;
      for(i=0;i<100000;i++){
       if((r-strad)<=0.001 && (r-strad)>=-0.001)break;
        theta=theta-(r-strad)*inc;
        r=kk*x/(kk*cos(theta)+x*sin(theta));
            if(i==9999){
            printf("Equation not solved!\n");
      printf("theta=%6.2f r=%6.6f\n",theta*180/PI,r);
            theta=0;
            break;
            }
       }
            switch(q){
            case 0:
            break;
            case 1:
            break;
            case 2:
            theta=PI-theta;
            break;
            case 3:
            theta=PI+theta;
            break;
            case 4:
            theta=2*PI-theta;
            break;
            default:
            break;
            }
       linesp=lc;
```

```
                printf("Angle of the shadow is %6.2f
                        degrees.\n",theta*180/PI);
            CAL1[122-camera%2+CALBLK*(camsl-1)]=theta;
          /* into 121 or 122 according to camera */
                if(theta>PI)theta=theta-PI;
                /* allow for 2nd camera 180 deg. round */
      CAL1[124-camera%2+CALBLK*(camsl-1)]=sin(theta);
                /* into 123 or 124 according to camera */
      CAL1[126-camera%2+CALBLK*(camsl-1)]=cos(theta);
                /* into 125 or 126 according to camera */
            KK=kk;
      printf("End of camera %d calibration.\n",camera);
printf("Do you want to continue calibration?(1=yes 0=no)");
          scanf("%d",&temp);
          if(temp !=1)break;
          }
       }
/* here to select camera from list. */


camsel()
   {
      camset(1);                        /* select calibration list */
      printf(" Camera list no. is %d\n",camera);
      printf(" Camera no. is %d\n",CAMINFO[camera]/256);
      printf(" Enter new camera list no.");
      scanf("%d",&camera);
      printf(" New camera no. is %d\n",
                              CAMINFO[camera]/256);
      }
/*
Here to list camera control codes in the order that the cameras
will be accessed. First byte is length of list.high half bite is chanel no.
low half bite -bit 0=mode, bit 1=preset enable.  0=last edge,7=2nd
edge,b=3rd edge,3=1st edge */
camset(order)
      int order;
      {
/* sequential list for calibration */
```

```
        if(order==1){
        CAMINFO[0]=CAM*2;   /* length of list */
        CAMINFO[1]=0x1dc;   /* last edge */
        CAMINFO[2]=0x2df;   /* 2nd edge */
        CAMINFO[3]=0x3cc;   /* last edge */
        CAMINFO[4]=0x4cf;   /* 2nd edge */
        CAMINFO[5]=0x5bc;
        CAMINFO[6]=0x6bf;
        CAMINFO[7]=0x7ac;
        CAMINFO[8]=0x8af;
        CAMINFO[9]=0x99c;
        CAMINFO[10]=0xa9f;
        CAMINFO[11]=0xb8c;
        CAMINFO[12]=0xc8f;
        CAMINFO[13]=0xd7c;
        CAMINFO[14]=0xe7f;
        CAMINFO[15]=0xf6c;
        CAMINFO[16]=0x106f;
        CAMINFO[17]=0x115c;
        CAMINFO[18]=0x125f;
        CAMINFO[19]=0x134c;
        CAMINFO[20]=0x144f;
        CAMINFO[21]=0x153c;
        CAMINFO[22]=0x163f;
        CAMINFO[23]=0x172c;
        CAMINFO[24]=0x182f;
        CAMINFO[25]=0x191c;
        CAMINFO[26]=0x1a1f;
        CAMINFO[27]=0x1b0c;
        CAMINFO[28]=0x1c0f;
        }
/* 2 lists giving full height starting at bottom for each end */
        else if(order==2){
        CAMINFO[0]=14;   /* length of list */
        CAMINFO[1]=0x1d4;   /* last edge */
        CAMINFO[2]=0x2d2;   /* 2nd edge */
        CAMINFO[3]=0x5b4;
        CAMINFO[4]=0x6b2;
```

```
        CAMINFO[5]=0x994;
        CAMINFO[6]=0xa92;
        CAMINFO[7]=0xd74;
        CAMINFO[8]=0xe72;
        CAMINFO[9]=0x1154;
        CAMINFO[10]=0x1252;
        CAMINFO[11]=0x1534;
        CAMINFO[12]=0x1632;
        CAMINFO[13]=0x1914;
        CAMINFO[14]=0x1a12;
        }
        else if(order==3){
        CAMINFO[0]=14;   /* length of list */
        CAMINFO[1]=0x3c4;   /* last edge */
        CAMINFO[2]=0x4c2;   /* 2nd edge */
        CAMINFO[3]=0x7a4;
        CAMINFO[4]=0x8a2;
        CAMINFO[5]=0xb84;
        CAMINFO[6]=0xc82;
        CAMINFO[7]=0xf64;
        CAMINFO[8]=0x1062;
        CAMINFO[9]=0x1344;
        CAMINFO[10]=0x1442;
        CAMINFO[11]=0x1724;
        CAMINFO[12]=0x1822;
        CAMINFO[13]=0x1b04;
        CAMINFO[14]=0x1c02;
        }
/* 2 lists giving levels 0 - 4 for each end */
        else if(order==4){
        CAMINFO[0]=8;   /* length of list */
        CAMINFO[1]=0x1d4;   /* last edge */
        CAMINFO[2]=0x2d2;   /* 2nd edge */
        CAMINFO[3]=0x5b4;
        CAMINFO[4]=0x6b2;
        CAMINFO[5]=0x994;
        CAMINFO[6]=0xa92;
        CAMINFO[7]=0xd74;
```

```
        CAMINFO[8]=0xe72;
        }
        else if(order==5){
        CAMINFO[0]=8;   /* length of list */
        CAMINFO[1]=0x3c4;   /* last edge */
        CAMINFO[2]=0x4c2;   /* 2nd edge */
        CAMINFO[3]=0x7a4;
        CAMINFO[4]=0x8a2;
        CAMINFO[5]=0xb84;
        CAMINFO[6]=0xc82;
        CAMINFO[7]=0xf64;
        CAMINFO[8]=0x1062;
        }
/* 2 lists giving levels 0 - 1 for each end */
        else if(order==6){
        CAMINFO[0]=4;   /* length of list */
        CAMINFO[1]=0x1d4;   /* last edge */
        CAMINFO[2]=0x2d2;   /* 2nd edge */
        CAMINFO[3]=0x5b4;
        CAMINFO[4]=0x6b2;
        }
        else if(order==7){
        CAMINFO[0]=4;   /* length of list */
        CAMINFO[1]=0x3c4;   /* last edge */
        CAMINFO[2]=0x4c2;   /* 2nd edge */
        CAMINFO[3]=0x7a4;
        CAMINFO[4]=0x8a2;
        }
/* 2 lists giving levels 2 - 3 for each end */
        else if(order==8){
        CAMINFO[0]=4;   /* length of list */
        CAMINFO[1]=0x994;
        CAMINFO[2]=0xa92;
        CAMINFO[3]=0xd74;
        CAMINFO[4]=0xe72;
        }
        else if(order==9){
        CAMINFO[0]=4;   /* length of list */
```

```
        CAMINFO[1]=0xb84;
        CAMINFO[2]=0xc82;
        CAMINFO[3]=0xf64;
        CAMINFO[4]=0x1062;
        }
 /* 2 lists giving levels 4 - 6 for each end */
        else if(order==10){
        CAMINFO[0]=6;   /* length of list */
        CAMINFO[1]=0x1154;
        CAMINFO[2]=0x1252;
        CAMINFO[3]=0x1534;
        CAMINFO[4]=0x1632;
        CAMINFO[5]=0x1914;
        CAMINFO[6]=0x1a12;
        }
        else if(order==11){
        CAMINFO[0]=6;   /* length of list */
        CAMINFO[1]=0x1344;
        CAMINFO[2]=0x1442;
        CAMINFO[3]=0x1724;
        CAMINFO[4]=0x1822;
        CAMINFO[5]=0x1b04;
        CAMINFO[6]=0x1c02;
        }
/* 2 lists giving levels 2 - 6 for each end */
        else if(order==12){
        CAMINFO[0]=10;   /* length of list */
        CAMINFO[1]=0x994;
        CAMINFO[2]=0xa92;
        CAMINFO[3]=0xd74;
        CAMINFO[4]=0xe72;
        CAMINFO[5]=0x1154;
        CAMINFO[6]=0x1252;
        CAMINFO[7]=0x1534;
        CAMINFO[8]=0x1632;
        CAMINFO[9]=0x1914;
        CAMINFO[10]=0x1a12;
        }
```

```
        else if(order==13){
        CAMINFO[0]=10;   /* length of list */
        CAMINFO[1]=0xb84;
        CAMINFO[2]=0xc82;
        CAMINFO[3]=0xf64;
        CAMINFO[4]=0x1062;
        CAMINFO[5]=0x1344;
        CAMINFO[6]=0x1442;
        CAMINFO[7]=0x1724;
        CAMINFO[8]=0x1822;
        CAMINFO[9]=0x1b04;
        CAMINFO[10]=0x1c02;
        }
        }
/* here to decide order of cameras or projectors during calibration */

calset(order)
        int order;
        {
        int min;

        min=40; /* height of lowest calibration light above turntable */
         if(order==0){
         CALCAM[0][0]=3;
         CALCAM[0][1]=min;
         CALCAM[1][0]=7;
         CALCAM[1][1]=min+300;
         CALCAM[2][0]=11;
         CALCAM[2][1]=min+600;
         CALCAM[3][0]=15;
         CALCAM[3][1]=min+900;
         CALCAM[4][0]=19;
         CALCAM[4][1]=min+1200;
         CALCAM[5][0]=23;
         CALCAM[5][1]=min+1500;
         CALCAM[6][0]=27;
         CALCAM[6][1]=1800;
         }
```

```
        else if(order==1){
        CALCAM[0][0]=1;
        CALCAM[0][1]=min;
        CALCAM[1][0]=5;
        CALCAM[1][1]=min+300;
        CALCAM[2][0]=9;
        CALCAM[2][1]=min+600;
        CALCAM[3][0]=13;
        CALCAM[3][1]=min+900;
        CALCAM[4][0]=17;
        CALCAM[4][1]=min+1200;
        CALCAM[5][0]=21;
        CALCAM[5][1]=min+1500;
        CALCAM[6][0]=25;
        CALCAM[6][1]=1800;
        }
      }
/* here to take a vertical slice of measurements */

cograb1(f)
    int f;
      {
      int angl,slice;

      if(f<0)slice=0;
      else slice=f;
      sliceget(0);
      angl=conorm(slice,f);
      return (angl);
      }
/* here to correct using CAL1[] table and normalise */
/* if f=-3 angle=0. */

sliceget(snum)
      int snum;
      {
      if(snum<0)snum=0;
      if (camera >0) camget(camera,snum);
```

```
        else {
        son(4);
        delay(10);
        camset(listflg);
        gcall(slicepoint(snum,0),snum);
        soff(12);
        son(8);
        delay(10);
        camset(listflg+1);
        gcall(slicepoint(snum,1),snum);
        soff(12);
        }
        }
/*
 here to take a vertical slice of measurements from 1 camera
*/

camget(camera,f)
        int camera,f;
        {
        int ff;
        if(f<0)ff=0;
        else ff=f;
        if(camera%2==0)son(8);
        else son(4);                        /* operate lights & shutters. */
        printf("camget(camera)=%d\n",camera);
        ccall(slicepoint(0,0),camera);
        soff(12);
        }
/* here to position pointer to beginning of slice data */

slicepoint(snum,endno)
        int snum,endno;
        {
        int cam;
        unsigned *ap;

        ap=&MEMSP[0];
```

```
        cam=CAMINFO[0];
        if(cam>CAM)cam=cam/2;
        if(snum<0)snum=0;
        if(camera>0)ap=ap+FRAME*snum;
        else  ap=ap+FRAME*cam*snum*2+endno*FRAME*cam;
        return(unsigned)(ap);
        }
/* here to capture wand data */


wand1()
        {
        int flag,portc,count,interval,temp;

        temp=listflg; /* save camera list no. */
        camset(1);  /* select sequential camera list. */
        calset(1);  /* select end, 0 = 0 degrees, 1 = 180 degrees. */
        flag=0;
        count=0;
        HEADER[19]=count;
        while(flag == 0){
    .   while(portc==0){
        c2call();
        portc=port2c & 0xe;
        }
        delay(10);
        c2call();
        portc=port2c & 0xe;
        switch(portc){
        case 4:
        interval = tstep(7);
        timer(clock()+interval);
        anglsum(900);
        break;
        case 8:
        interval = tstep(8);
        timer(clock()+interval);
        anglsum(-900);
        break;
```

```
        case 12:
        flag=1;
        camera=0;
        bell();
        break;
        case 2:
        getwand(count);
        count++;
        HEADER[19]=count;  /* save count of ref points */
        if(count>20)flag=1;
         bell();
          printf("Ready !!!!\n");
        break;
        default:
        break;
        }
        }
        listflg=temp; /* restore camera list flag */
        camset(listflg);
        camera=0;
        }
 /* here to find position of wand */


getwand(count)
        int count;
{
        int o;
          for(o=0;o<=6;o++){
                  initarry();
                  camera=CALCAM[o][0];
                  height=CALCAM[o][1];
        ccall(slicepoint(0,0),camera); /* get slice from 1 camera */
        wandnorm(count);                              /* normalise slice
                                                              data */
                        if(HEADER[count*6+20]>0){
                        printf("ref no.= %d \n",count);
                        printf("x= %f \n",HEADER[count*6+21]);
                printf("height= %f \n",HEADER[count*6+22]);
```

```
                printf("xtan= %f \n",HEADER[count*6+23]);
                printf("ytan= %f \n",HEADER[count*6+24]);
                printf("angle= %f \n",HEADER[count*6+25]);
                        }
                }
                count++;
        }
/* here to print wand data */


wandprnt()
 {
 int i;

 for(i=0;i<HEADER[19];i++){
                        printf("ref no.= %d \n",i);
                        printf("x= %f \n",HEADER[i*6+21]);
                        printf("height= %f \n",HEADER[i*6+22]);
                        printf("xtan= %f \n",HEADER[i*6+23]);
                        printf("ytan= %f \n",HEADER[i*6+24]);
                        printf("angle= %f \n",HEADER[i*6+25]);
                        }
      }
/* here to correct data from the wand using CAL1[] table and normalise
*/


wandnorm(count)
        int count;
        {
        int i,j,k,l,r,xx,yy,nn,aa,xc,yc,camsl,camno,caloff;
        double *cl1,x,y,z,zro,qq;
        short *np;

        xc=(XCOUNT-ZERO)/6;
        yc=(YCOUNT-TOPBLANK)/6;
        caloff=xc/2;            /* offset of calibration frame */
        j=slicepoint(0,0);
        k=slicepoint(1,0);
        qq=0;
```

```
        nn=0;
         HEADER[count*6+20]=nn;
         for(i=j;i<k;i=i+4){
         unpack(i);              /* get raw data */
         r = BYTES[0]-ZERO-caloff;                    /*    uncorrected
radius. */
         l = BYTES[1];                                /* original line no. */
               if(l==YCOUNT){
               camno=BYTES[3];  /* camera list number. */
               printf("%d\n",camno);
               camsl=camno/2 + camno%2;
                              /* get camera number not list no. */
               cl1= &CAL1[CALBLK*(camsl-1)];
                                    /* point to correction block */
 /*     height=HEIGHT;*/
               zro=CAL1[128-camno%2+CALBLK*(camsl-1)];
                     /* from 127 or 128 according to camera */
               }
         else if(l<=(YCOUNT-TOPBLANK) && l>0 && r>0){
               yy=l/yc;                               /* box y value */
               y=l%yc;                                /* y value in box */
               xx=r/xc;                               /* box x value */
               x=r%xc;                                /* x value in box */
               aa=24*xx+4*(5-yy);   /* pointer to box data */
/*
printf("x=%g,y=%g,xs=%g,ys=%g\n",*(cl1+aa),*(cl1+aa+1),*(cl1+aa+2),*
(cl1+aa+3));
*/
               y=*(cl1+aa+1)+*(cl1+aa+3)*y;
                                    /* new y value in real space */
               x=*(cl1+aa)+*(cl1+aa+2)*x-CZERO-zro;
                                    /* new x value in real space */
               qq=qq+x;
               nn++;
               }
         }
         if(nn>0){
         qq=qq/nn; /* x value */
```

```
        printf("count= %d\n",count);
        HEADER[count*6+20]=nn;
        HEADER[count*6+21]=qq;              /* x value in mm. */
        HEADER[count*6+22]=height+y;
                                /* height from base in mm. */
        HEADER[count*6+23]=x/CAMDIST;
                                        /* x direction tangent */
        y=y-YMAX/2;
        HEADER[count*6+24]=y/CAMDIST;
                                        /* y direction tangent */
        HEADER[count*6+25]=anglcount;
        }
}
```

/* newdrive.c */

```
#include "lassdefs"

extern char pt;
extern double anglcount;
extern int po;
extern unsigned char port2c;


/* motor drive selection */


table3()
      {

 printf("0 = Initialize table drive.\n");
 printf("1 = Stop table.\n");
 printf("2 = Set angle counter.\n");
 printf("3 = Read angle counter.\n");
 printf("4 = Not used.\n");
 printf("5 = Run table continuously.\n");
 printf("6 = Drive table clockwise through 1 degrees.\n");
 printf("7 = Drive table anticlockwise through 1 degrees.\n");
 printf("8 = Drive table clockwise through 5 degrees.\n");
```

```
 printf("9 = Drive table anticlockwise through 5 degrees.\n");
 printf("a = Drive table clockwise through 45 degrees.\n");
 printf("b = Drive table anticlockwise through 45 degrees.\n");
 printf("c = Drive table clockwise through 90 degrees.\n");
 printf("d = Drive table anticlockwise through 90 degrees.\n");
 printf("e = Drive table clockwise through 180 degrees.\n");
 printf("f = Drive table anticlockwise through 180 degrees.\n");
 printf("g = Drive table clockwise through 0.1 degrees.\n");
 printf("h = Drive table clockwise through 30 degrees.\n");
 printf("i = Drive table anticlockwise through 30 degrees.\n");
 printf("j = Drive table clockwise through 120 degrees.\n");
 printf("k = Drive table anticlockwise through 120 degrees.\n");
 printf("Type 'space' to return to the main program.\n");
        }

select3()
        {
        int b;

        printf("Angle counter = %4.1f\n",anglcount);
        while ((b=getchar()) != ' '){
        switch (b){
        case '0':
        tstart();
        break;
        case '1':
        tstop();
        break;
        case '2':
        printf("Angle = %4.1f Table angle =      %4.1f\n",anglcount,360-
anglcount);
        printf("Enter new value for Table Angle.");
        scanf("%F",&anglcount);
        anglcount=360-anglcount;
        break;
        case '3':
        printf("Angle = %4.1f Table angle =      %4.1f\n",anglcount,360-
anglcount);
```

```
        break;
        case '4':
        break;
        case '5':
        trun(5);
        break;
        case '6':
        tstep(1);
anglsum(10);
        break;
        case '7':
        tstep(2);
anglsum(-10);
        break;
        case '8':
        tstep(3);
anglsum(50);
        break;
        case '9':
        tstep(4);
anglsum(-50);
        break;
        case 'a':
        tstep(5);
anglsum(450);
        break;
        case 'b':
        tstep(6);
anglsum(-450);
        break;
        case 'c':
        tstep(7);
anglsum(900);
        break;
        case 'd':
        tstep(8);
anglsum(-900);
        break;
```

```
        case 'e':
        tstep(9);
    anglsum(1800);
        break;
        case 'f':
        tstep(10);
    anglsum(-1800);
        break;
        case 'g':
        tstep(11);
    anglsum(1);
        break;
        case 'h':
        tstep(12);
    anglsum(300);
        break;
        case 'i':
        tstep(13);
    anglsum(-300);
        break;
        case 'j':
        tstep(16);
    anglsum(1200);
        break;
        case 'k':
        tstep(17);
    anglsum(-1200);
        break;
        default:
        table3();
        break;
        }
        }
        }
/* here to total angles */

anglsum(step)
        int step;
```

```
        {
       double dstep;

        dstep=step;
        anglcount=dstep/10+anglcount;
        if(anglcount>=360)anglcount=anglcount-360;
        if(anglcount<0)anglcount=anglcount+360;
        printf("Angle = %4.1f Table angle =
       %4.1f\n",anglcount,360-anglcount);
        }
/* call port1b parallel interface card 'off'.*/

pintoff(pp)
        int pp;
       {
       char px;

       px=(char)pp;
       pt=pt & ~px;
       pcall(pt);
       }
/* call port1b parallel interface card 'on'.*/

pinton(pp)
        int pp;
       {
       char px;

       px=(char)pp;
       pt=pt | px;
       pcall(pt);
       }
/* call port1a parallel interface card 'off'.*/

soff(pp)
        int pp;
       {
```

```
        char px;

        px=(char)pp;
        pt=pt & ~px;
        scall(pt);
        }
/* call port1a parallel interface card 'on'.*/


son(pp)
         int pp;
        {
        char px;

        px=(char)pp;
        pt=pt | px;
        scall(pt);
        }
/* table drive */


tstart()
         {
        int write(path,buffer,count);
        char *buffer;
          int open(name,mode);
          char *name,c;
          char *text1,*text2,*text3;
          short mode;
          int i,j;


        text1="(\n";
        text2="U31\n";
        text3="@1004\n";
          name="/t1";
          po = open(name,2);

        for(i=0;i<3;i++){
        c=*text1;
```

```
        text1++;
         write(po,&c,1) ;
        for (j=0;j<10000;j++);
         };
        for(i=0;i<5;i++){
        c=*text2;
        text2++;
         write(po,&c,1) ;
        for (j=0;j<10000;j++);
         };
        for(i=0;i<6;i++){
        c=*text3;
        text3++;
         write(po,&c,1) ;
        for (j=0;j<10000;j++);
         }
          trun(5);
          c2call();
          i=port2c & 1;
          while(i==0){
          c2call();
          i=port2c & 1;
          }
          tstop();
          anglcount=0;
          }


tstop()
          {
          char *text4,c;
          char *name;
          short mode;
          int i,j;
          text4="#\n";
        for(i=0;i<3;i++){
        c=*text4;
        text4++;
         write(po,&c,1) ;
```

```
        for (j=0;j<1000;j++);
         }
         }

trun(ss)
         int ss;
         {
          char *text1,*text2,*text3,*text4,*text5,*text6,c;
          char *name;
          short mode;
          int i,j;
        text1="@508 X+G$\n";
        text2="@508 X-G$\n";
        text3="@300 X+G$\n";
        text4="@300 X-G$\n";
        text5="@1477X+G$\n";
        text6="@465 X-G$\n";
        for(i=0;i<10;i++){
        if(ss==1){
        c=*text1;
        text1++;}
        if(ss==2){
        c=*text2;
        text2++;}
        if(ss==3){
        c=*text3;
        text3++;}
        if(ss==4){
        c=*text4;
        text4++;}
        if(ss==5){
        c=*text5;
        text5++;}
        if(ss==6){
        c=*text6;
        text6++;}
          write(po,&c,1) ;
        for (j=0;j<1000;j++);
```

```
        }
        }

tstep(ss)
        int ss;
        {
         char *text1,*text2,*text3,*text4,*text5,*text6,*text7,c;
         char *text8,*text9,*text10,*text11,*text12;
         char *text13,*text14,*text15;
         char *text16,*text17;
         char *name;
         short mode;
         int i,j,time;
       text1="@1000X  +200$\n";
       text2="@1000X  -200$\n";
       text3="@1000X +1000$\n";
       text4="@1000X -1000$\n";
       text5="@2500X+ 9000$\n";
       text6="@2500X- 9000$\n";
       text7="@2500X+18000$\n";
       text8="@2500X-18000$\n";
       text9="@2500X+36000$\n";
       text10="@2500X-36000$\n";
       text11="@1000X    +20$\n";
       text12="@2500X+ 6000$\n";
       text13="@2500X- 6000$\n";
#if RESOLUTION
       text14="@800X     -480$\n";
       text15="@2500X  -480$\n";
#else
       text14="@800X  -1000$\n";       /* here 5deg./step replace
                                  with above for 2.4 deg./step */
       text15="@2500X -1000$\n";
#endif
       text16="@2500X+24000$\n";
       text17="@2500X-24000$\n";
       for(i=0;i<14;i++){
       if(ss==1){
```

```
time=20;
c=*text1;
text1++;}
if(ss==2){
time=20;
c=*text2;
text2++;}
if(ss==3){
time=100;
c=*text3;
text3++;}
if(ss==4){
time=100;
c=*text4;
text4++;}
if(ss==5){
time=360;
c=*text5;
text5++;}
if(ss==6){
time=360;
c=*text6;
text6++;}
if(ss==7){
time=72;
c=*text7;
text7++;}
if(ss==8){
time=72;
c=*text8;
text8++;}
if(ss==9){
time=1440;
c=*text9;
text9++;}
if(ss==10){
time=1440;
c=*text10;
```

```
        text10++;}
        if(ss==11){
        time=2;
        c=*text11;
        text11++;}
        if(ss==12){
        time=240;
        c=*text12;
        text12++;}
        if(ss==13){
        time=240;
        c=*text13;
        text13++;}
        if(ss==14){
#if RESOLUTION
        time=60;
#else
        time=125;
#endif
        c=*text14;
        text14++;}
        if(ss==15){
#if RESOLUTION
        time=30;
#else
        time=60;
#endif
        c=*text15;
        text15++;}
        if(ss==16){
        time=960;
        c=*text16;
        text16++;}
        if(ss==17){
        time=960;
        c=*text17;
        text17++;}
          write(po,&c,1) ;
```

```
        for (j=0;j<1000;j++);
         }
        return(time);
         }
/* here to control calibration column lights */


calite()
        {
        int x;
        printf("Column number 5 switches off all lights.\n");
        printf("Enter column number (0-4).");
        scanf("%d",&x);
        if(x>=5){
        pintoff(31);                    /* switch off lights */
        }
        switch(x){
        case 0:
        pintoff(31);                    /* switch off lights */
        pinton(1);
        break;
        case 1:
        pintoff(31);                    /* switch off lights */
        pinton(2);
        break;
        case 2:
        pintoff(31);                    /* switch off lights */
        pinton(4);
        break;
        case 3:
        pintoff(31);                    /* switch off lights */
        pinton(8);
        break;
        case 4:
        pintoff(31);                    /* switch off lights */
        pinton(16);
        break;
        }
        }
```

```
/* plot.c */
#include "lassdefs"

        extern short  NDATA[NARRAY];
                                        /* normalised data array */
        extern int mouse_port,mouse_x,mouse_y;
        extern char mouse_buttons;
        extern int BYTES[4];        /* common for passing parms. */
        extern int CAMINFO[2*CAM+1];
        extern double TAPE[STEPS+1][2];
                        /* temp store for tape measurements */
        extern float HEADER[HEADSIZE];/* file header info */
        extern double CAL1[CALBLK*CAM];
                    /* calibration array X,Y,Xslope,Yslope */
        extern int *hrgbase;
        extern char *font_0;
        extern double sintheta,costheta;
                        /*theta is angle for current shadow */
        extern double w_scale,x_scale,y_scale;
        extern int linetype,colour,wndow,zoom;
        extern double rad,offset,kk,smooth;
        extern int camera,height;
        extern int write(path,buffer,count);
        extern char *buffer,pt;
        extern int cflag,avflag,armflag,linesp,steps;
        extern int po;
        extern double xsum,ysum,zsum,xyzcount;
        extern float data;
        extern unsigned count;
        extern int slicearray[STEPS+5][2];
                        /* storage for horizontal slice X,Y */
        extern double slicexy[4*NFRAME][2];
/* here to plot 1 horizontal slice */


splot(hite)
        int hite;
         {
        int i,j,l,nn,angl,anglmax;
```

```
        float nslice;
        int xcent,ycent;
        int xc,yc,r,xtemp,ztemp,x0temp,z0temp;
        int xmax,xmin,zmax,zmin;
        double aa,mm,radius,theta,phi,circ;
        double ctemp,first,old,cosa,rmean;
        double radmax,tape(),measure,radfirst,radprev,area;
        int xx,zz,firstflg;
        short *np;

        xcent=350;  /* center of x axis */
        ycent=286;  /* center of y axis */
        cosa=cos(ANGLE*rad);      /* find cos of ANGLE */
        old=0;
        angl=0;
        circ=0;
        ctemp=0;
        radmax=0;
        anglmax=0;
        firstflg=0;
        area=0;
        theta=angl*ANGLE*rad;
        phi=sin(ANGLE*rad);
/* here to establish average */
        for(angl=0;angl<STEPS;angl++){
         np=NPOINT; /* array point */
         radius=*np;
         nn=*(np+1);
              if(nn>0){
              radius=radius/nn;
               if(radius>radmax){         /* find max radius */
               radmax=radius;
               anglmax=angl;              /* angle of max radius */
               }
              radius=radius*smooth+old*(1-smooth);
              old=radius;
              }
        }
```

```
/* end of average run */
        first=radius;
        rmean=first;
        xx=radius*sin(theta);
        xmax=xx;
        xmin=xx;
        zz=radius*cos(theta);
        zmax=zz;
        zmin=zz;
        xx=xx+xcent;                /* x plotted in y direction */
        zz=zz+ycent;                /* z plotted in x direction */
        slicearray[STEPS][0]=xx;
        slicearray[STEPS][1]=zz;
        zz=zz*w_scale;
        x0temp=xx;
        xtemp=xx;
        z0temp=zz;
        ztemp=zz;
        z0temp=zz;
        wndow=3;
        usewdw(wndow);                  /* chose window */
        colour=7;
        setcol(colour);                 /* set colour */
        lin type=0xffff;
        set_lt(linetype);
        point(x0temp,z0temp);
          for(angl=0;angl<STEPS;angl++){
          np=NPOINT; /* array point */
          radius=*np;
          nn=*(np+1);
                if(nn>0){
                radius=radius/nn;
                  if(firstflg==0){
                  radfirst=radius;
                  radprev=radius;
                  firstflg=1;
                  }
                else area=radius*radprev*phi/2+area;
```

```
        radprev=radius;
        if(radius>(old+15) && armflag==1)radius=old;
                                        /* arm removal */
        radius=radius*smooth+old*(1-smooth);
        circ=circ+sqrt(radius*radius+old*old-
                                2*radius*old*cosa);
        old=radius;
        rmean=rmean+radius;
        theta=angl*ANGLE*rad;
        xx=radius*sin(theta);
        if(xx>xmax)xmax=xx;
        if(xx<xmin)xmin=xx;
        zz=radius*cos(theta);
        if(zz>zmax)zmax=zz;
        if(zz<zmin)zmin=zz;
        xx=xx+xcent;            /* x plotted in y direction */
        zz=zz+ycent;            /* z plotted in x direction */
        slicearray[angl][0]=xx;
        slicearray[angl][1]=zz;
        zz=zz*w_scale;
        vector(xtemp,ztemp,xx,zz);
        xtemp=xx;
        ztemp=zz;
        }
 }
area=radfirst*radprev*phi/2+area;
vector(xtemp,ztemp,x0temp,z0temp);
measure=tape(hite);
circ=circ+sqrt(radius*radius+first*first-
                            2*radius*first*cosa);
printf("circumference=%-5.1f\n",circ);
printf("tape circumference=%-5.1f\n",measure);
printf("width=%-4d,depth=%-4d\n",xmax-xmin,
                                        zmax-zmin);
printf("xmin=%-4d,xmax=%-4d,zmin=%-4d,zmax=
                    %-4d\n",xmin,xmax,zmin,zmax);
printf("area=%1.6f square metres.\n",area/1000000);
printf("height=%d\n",hite*VSTEP);
```

```
        return(area);
        }
/* here to print slice data to a file */

slprint()
        {
        int i;
          for(i=1;i<=STEPS;i++){
          printf("%d %d\n",slicearray[i][0],slicearray[i][1]);
          }
        }
/* here to plot cartesian slice */


xyslplot()
        {
        int i,xx,yy,count,xcent,ycent;

        xcent=350;  /* center of x axis */
        ycent=286;  /* center of y axis */
        setcol(7);
        count=slicexy[0][0];
        if(count>4*NFRAME)count=4*NFRAME;
        clrwdw(wndow);
        for(i=0;i<count;i++){
        xx=slicexy[i][0]+xcent;
        yy=slicexy[i][1]+ycent;
        point(xx,yy);
        }
        }
/* here to calculate tape-measure distance */


double tape(hite)
        int hite;
        {
        int step,angl,refangl,newangl,nn,i;
        int arp1,arp2,arp1max,xx,yy,xs,ys;
        int xcent,ycent,firstangl,oldangl,maxarp1,count,loop;
```

```
        int anglmax,anglmin;
        short *np;
        double ab,ac,ad,ae,af,rad1,rad2,rad3,rad4;
        double radius,circ,first,old,radmax,radmin;
        double temp[STEPS+5][2];

        xcent=350;          /* centre offset in x direction */
        ycent=286;                  /* centre offset in y direction */
        circ=0;                         /* zero distance measure */
        setcol(2);                  /* set colour */
        set_lt(0x1111);             /* set line type */
        rad2=ANGLE*rad; /* unit angle in rads */
        radmax=0;
        radmin=600;
/* here to establish min rad */
         for(angl=0;angl<STEPS;angl++){
         np=NPOINT; /* array point */
         radius=*np;
         nn=*(np+1);
              if(nn>0){
              radius=radius/nn;
                if(radius<radmin && radius>0){
                radmin=radius;
                anglmin=angl;
                }
              }
         }
/* here to eliminate arms - array filled starting with min. rad.*/
        angl=anglmin;
        old=radmin;
        count=0;
         for(i=0;i<STEPS;i++){
         np=NPOINT; /* array point */
         radius=*np;
         nn=*(np+1);
              if(nn>0){
              radius=radius/nn;
              if(radius>(old+15) && armflag==1)radius=old;
```

```
                                        /* arm removal */
            if(radius>radmax){          /* find max radius */
            radmax=radius;
            anglmax=count;  /* position of max radius */
            }
          temp[count][0]=radius;      /* load array */
          temp[count][1]=angl;        /* load array */
          count++;
          old=radius;
          }
        angl++;
        if(angl>=STEPS)angl=angl-STEPS;
       }
/* here to fill array starting with max. rad. */
          TAPE[count][0]=temp[anglmax][0];
                /* make end equal to beginning */
          TAPE[count][1]=temp[anglmax][1];
          for(i=0;i<count;i++){
          TAPE[i][0]=temp[anglmax][0];
          TAPE[i][1]=temp[anglmax][1];
          anglmax++;
          if(anglmax>=count)anglmax=anglmax-count;
          }
/* here to start tape calculation */
      maxarp1=count;
      count=0;
      do{
      count++;
      arp1max=maxarp1;
      maxarp1=0;
      arp1=1;                       /* first array pointer */
          for(loop=2;loop<=arp1max;loop++){
          arp2=loop;
          if(arp2>=STEPS)arp2=arp2-STEPS;
          radius=TAPE[arp2][0];
          angl=TAPE[arp2][1];
          ab=TAPE[arp1-1][0];               /* reference vector */
          refangl=TAPE[arp1-1][1];    /* ref vector angle */
```

```
            ac=TAPE[arp1][0];          /* newest vector */
            newangl=TAPE[arp1][1];  /* new vector angle */
            rad1=(angl-refangl)*rad2;
                                /* convert steps to angle in rads */
            rad3=(angl-newangl)*rad2;
                                /* convert steps to angle in rads */
            rad4=(newangl-refangl)*rad2;
                                /* convert steps to angle in rads */
/*
 ad= length of vector which just sits on line between ab & ac
*/
        ad=radius*ab*sin(rad1)/(ab*sin(rad4)+radius*sin(rad3));
         if(ac<ad){
       if(arp1>1){
       ae=TAPE[arp1-2][0];  /* previous reference vector */
       newangl=refangl;
       refangl=TAPE[arp1-2][1];    /* ref vector angle */
       rad1=(angl-refangl)*rad;
                           /* convert steps to angle in rads */
       rad3=(angl-newangl)*rad2;
                           /* convert steps to angle in rads */
       rad4=(newangl-refangl)*rad2;
                           /* convert steps to angle in rads */

       af=radius*ae*sin(rad1)/(ae*sin(rad4)+radius*sin(rad3));
        if(ab<af){
        arp1--;
       if(arp1<maxarp1)maxarp1=arp1;
       if(arp1<0)arp1=0;
        }
       }
        TAPE[arp1][0]=radius;
        TAPE[arp1][1]=angl;
        }
            else{
            arp1++;
            if(arp1>maxarp1)maxarp1=arp1;
            TAPE[arp1][0]=radius;
```

```
            TAPE[arp1][1]=angl;
            }
            }
        }
        while((arp1max-maxarp1)>0);
            first=TAPE[0][0];  /* starting radius */
            firstangl=TAPE[0][1]; /* starting angle */
            old=first;
            oldangl=firstangl;
            xs=TAPE[0][0]*sin(TAPE[0][1]*rad2)+xcent;
            ys=(TAPE[0][0]*cos(TAPE[0][1]*rad2)+ycent)*w_scale;
              for(i=1;i<=maxarp1;i++){
             radius=TAPE[i][0];
             angl=TAPE[i][1];
             rad1=angl-oldangl;
             if(rad1>STEPS)rad1=rad1-STEPS;
             if(rad1<0)rad1=rad1+STEPS;
             rad1=rad1*rad2;
            circ=circ+sqrt(radius*radius+old*old-
                                2*radius*old*cos(rad1));
             old=radius;
             oldangl=angl;
             xx=TAPE[i][0]*sin(TAPE[i][1]*rad2)+xcent;
             yy=(TAPE[i][0]*cos(TAPE[i][1]*rad2)+ycent)*w_scale;
             vector(xs,ys,xx,yy);
             xs=xx;
             ys=yy;
             }
            return(circ);
        }
/* x,y, plot routine with origin in bottom l.h. corner. */
/* f=0 or f=-1 clears screen. f<0 plots direct (unprojected). */

plot1(angl,f)
          int angl,f;
        {
        int hite;
        hite=height/VSTEP;
```

```
        plot2(angl,hite,angl,f);
        }
/* x,y, plot routine with origin in bottom l.h. corner. */
/* f=0 or f=-1 clears screen. f<0 plots direct (unprojected). */
/* angl,hite address the array.*/
/*Theta is the angle for the projection. */
/* f provides for control like screen clear etc. */

plot2(angl,hite,theta,f)
        int angl,hite,theta,f;
        {
        extern int steps,linesp;
        int h,i,j,l,x,y,htmax;
        double r,s,phi,xs_cale,ys_cale,mm;
        short *np;

        if(angl>=STEPS)angl=angl-STEPS;
        if(theta>=STEPS)theta=theta-STEPS;
      xs_cale=((double)600/(XCOUNT-ZERO));
                                        /* camera x axis 0-416 */
      ys_cale=((double)450*w_scale/(YCOUNT-
                                TOPBLANK)); /* camera y axis */
      colour=7;
      setcol(colour);
      if(f<1){
       clrwdw(wndow);
       grid(4);
       }
            if (cflag==0){
            j=slicepoint(0,0);
            l=slicepoint(1,0);
              for(i=j;i<l;i=i+4*linesp){
             unpack(i);
             h = BYTES[1];
             h = h*ys_cale;
             r =  BYTES[0]-ZERO;/* uncorrected radius. */
             x =r*xs_cale;
             point(x,h);
```

```
                }
               }
                else {
                 phi=theta*ANGLE*rad;
                 if(phi>PI)phi=PI-phi;
                 htmax=hite+YMAX/VSTEP;
                i=0;
                        for(;hite<=htmax;hite++){          /* do rows */
                        np=NPOINT;   /* point to array */
                        r= *np;
                        s = *(np+1);
                        if(s==0)r=0;
                        else r=r/s;
                        if(f>-1)r = r*cos(phi)+XMAX/2+50;
                        x =r*x_scale;
                        y=i*VSTEP*y_scale;
                        point(x,y);
                        i++;
                        }
                }
        }
/* here to plot xyz co-ords. */

cartplot()
        {
        printf("Enter zoom value. ");
        scanf("%d",&zoom);
        xyzplot(0,0);
        }
/* ht is starting height & fb=0 for front & fb=1 for back view. */

xyzplot(ht,fb)
        int ht,fb;
        {
        int i,l,x,y,offset1,offset2,angl,hite,smax;
        short *ap;
        double mm,xx,yy,zz,radius,theta;
```

```
grid(1);
colour=7;
setcol(colour);
set_dm(0);
offset1=150*x_scale;
offset2=350*x_scale;
  mm=ht;                          /* height of bottom line */
  nplot(15,15,mm);
        for(i=1;i<6;i++){     /* plot horizontal grid numbers */
        mm=50*i*zoom+ht;
        l=(50*i+5)*y_scale;
        nplot(15,l,mm);
        }
linetype=0xffff;
set_lt(linetype);
setcol(colour);
smax=(ht+300*zoom)/VSTEP;
if(smax>NFRAME)smax=NFRAME;
  for(angl=0;angl<STEPS;angl++){
        for(hite=0;hite<smax;hite++){
        ap=NPOINT;
        radius=*ap;
        ap++;
        count=*ap;
        if(count>0)radius=radius/count;
        else radius=0;
        theta=angl*ANGLE*rad;
        xx=radius*cos(theta);   /* X value */
        yy=radius*sin(theta); /* Y value */
        zz=hite-ht/VSTEP;                       /* Z value */
        y=zz*y_scale*VSTEP/zoom;
           /* y axis on screen is z axis on co-ord system */
        if(fb==0){
         if(xx>0){
         x=yy*x_scale/zoom+offset1;
         point(x,y);
         }
```

```
            if(yy<0){
             x=xx*x_scale/zoom+offset2;
             point(x,y);
             }
            }
            else {
             if(xx<=0){
             x=offset1-yy*x_scale/zoom;
             po'nt(x,y);
             }
             if(yy>=0){
             x=offset2-xx*x_scale/zoom;
             point(x,y);
             }
            }
        }
       }
       }
/* here to set up screen grid */

grid(number)
 int number;
{
       int a,b,c,d,i,j,l,r,xcent,ycent,caloff;
       double mm,aa;

        if (number==1){
        wndow=2;
        usewdw(wndow);          /* chose window */
        colour=3;
        setcol(7);              /* set colour */
        clrwdw(wndow);
        setcol(colour);         /* set colour */
        linetype=0xffff;
        set_lt(linetype);
        frmwdw(2);
        b=0;
        d=450*y_scale;
```

```
set_lt(0x0101);                 /* set line type */
        for(i=1;i<10;i++){ /* plot vertical grid */
        a=50*i*x_scale;
        c=50*i*x_scale;
        vector(a,b,c,d);
        }
a=0;
c=750;
        for(i=1;i<6;i++){       /* plot horizontal grid */
        b=50*i*y_scale;
        d=50*i*y_scale;
        vector(a,b,c,d);
        }
}
else if(number==2){
wndow=3;
colour=3;
usewdw(wndow);                  /* chose window */
setcol(7);                      /* set colour */
clrwdw(wndow);
linetype=0xffff;
setcol(colour);                 /* set colour */
frmwdw(wndow);
xcent=350;              /* centre offset in x direction */
ycent=286;              /* centre offset in y direction */
r=50;
        for(i=1;i<=6;i++){   /* plot circles */
                for(j=0;j<20*PI;j++){
                aa=(double)j/10;
                a=i*r*sin(aa)+xcent;
                b=(i*r*cos(aa)+ycent)*w_scale;
                point(a,b);
                }
        mm=(double)50*i;    /* plot numbers */
        l=(ycent-10-(50*i))*w_scale;
        nplot(xcent-30,l,mm);
        }
}
```

```
else if(number==3){
colour=3;
wndow=2;
usewdw(wndow);          /* chose window */
setcol(7);              /* set colour */
clrwdw(wndow);
setcol(colour);         /* set colour */
linetype=0xffff;
set_lt(linetype);
frmwdw(2);
b=0;
d=450*y_scale;
linetype=(0x0101);      /* set line type */
set_lt(linetype);
      for(i=1;i<10;i++){  /* plot vertical grid */
      a=50*i*x_scale;
      c=50*i*x_scale;
      vector(a,b,c,d);
      }
a=0;
c=750;
      for(i=1;i<6;i++){         /* plot horizontal grid */
      b=50*i*y_scale;
      d=50*i*y_scale;
      vector(a,b,c,d);
      }
}
else if(number==4){
colour=3;
wndow=5;
usewdw(wndow);          /* chose window */
setcol(colour);         /* set colour */
setcol(7);              /* set colour */
clrwdw(wndow);
setcol(colour);         /* set colour */
linetype=0xffff;
set_lt(linetype);
frmwdw(wndow);
```

```
b=0;
d=450*w_scale;
linetype=(0x0101);                    /* set line type */
set_lt(linetype);
caloff=600/12;      /* move calibration window 1/2 pitch */
      for(i=0;i<6;i++){              /* plot vertical grid */
      a=600*i/6+caloff;
      c=600*i/6+caloff;
      vector(a,b,c,d);
      }
a=0;
c=600;
      for(i=1;i<6;i++){              /* plot horizontal grid */
      b=450*i*w_scale/6;
      d=450*i*w_scale/6;
      vector(a,b,c,d);
      }
}
else if(number==5){
wndow=1;
usewdw(1);                    /* chose window */
colour=3;
setcol(colour);               /* set colour */
setcol(7);                    /* set colour */
clrwdw(wndow);
setcol(colour);               /* set colour */
linetype=0xffff;
set_lt(linetype);
frmwdw(1);
mesg(1);
mesg(2);
mesg(3);
mesg(4);
}
else if(number==6){
wndow=1;
usewdw(1);                    /* chose window */
colour=3;
```

```
        setcol(7);                      /* set colour */
        clrwdw(wndow);
        setcol(colour);                 /* set colour */
        linetype=0xffff;
        set_lt(linetype);
        frmwdw(1);
        mesg(6);
        mesg(7);
        mesg(3);
        mesg(4);
        }
        linetype=0xffff;
        set_lt(linetype);
}
/* here to print messages on the graphics screen */

mesg(number)
      int number;
      {
      char  *mm,*m1,*m2,*m3,*m4,*m5,*m6,*m7;
      int x1,x2,y1,y2,y3;

      set_dm(0);
      x1=50;
      x2=375;
      y1=5*y_scale;
      y2=25*y_scale;
      y3=45*y_scale;
      m1="Rotate clockwise.      ";
      m2="Rotate anticlockwise. ";
      m3="Move higher.                ";
      m4="Move lower.            ";
      m5="Quit.                       ";
      m6="Zoom larger            ";
      m7="Zoom smaller                ";
       if(number==1){
       chrcur(x1,y3);
        mm=&m1[0];
```

```
        }
        else if(number==2){
        chrcur(x1,y2);
        mm=&m2[0];
        }
        else if(number==3){
        chrcur(x2,y3);
        mm=&m3[0];
        }
        else if(number==4){
        chrcur(x2,y2);
        mm=&m4[0];
        }
        else if(number==5){
        chrcur(x2,y1);
        mm=&m5[0];
        }
        else if(number==6){
        chrcur(x1,y3);
        mm=&m6[0];
        }
        else if(number==7){
        chrcur(x1,y2);
        mm=&m7[0];
        }
        while(*mm !='\0'){
        dischr(*mm);
        mm++;
        }
        dischr(32);
        dischr(127);
    }
/* here to plot numbers on the graphics screen */

nplot(x,y,mm)
    int x,y;
    double mm;
    {
```

```
        int sw,nn;
        int i,j;
        char k;

        mm=mm*10;
        nn=0;
        setcol(7);
        set_lt(0xffff);
        set_dm(0);
        chrcur(x,y);
        for(i=1000000;i>1;i=i/10){
                                /* here to convert binary to ASCII */
        k=0;
        while(mm >=0 ){
        k++;
        mm=mm-i;
        }
        mm=mm+i;
        if(k>1)nn=1;
        k=k+47;
        if(nn>0){
        dischr(k);
        }
        }
        dischr(46);
        k=mm+47+1;
        dischr(k);
        setcol(colour);
        set_lt(linetype);
        }
/* here to display font */

fontest()
        {
        int x,y,i,j,k,cn;
        char chr;

 usewdw(2);
```

```
 set_lt(0xffff);                    /* set line type */
 clrall();
 i=100;
 j=500;
 do{
       printf(" Enter char no. ");
       scanf("%d",&cn);
        if(cn==1){
              for(k=30;k<120;k++){
              chr=k;
              i=i+16;
               if(i>512){
               j=j-30;
               i=100;
               }
              chrcur(i,j);
              dischr(chr);
              }
       }
       else{
       chr=cn;
       i=i+16;
              if(i>512){
              j=j-30;
              i=100;
              }
       chrcur(i,j);
       dischr(chr);
       }
 }
 while(cn!=0);
 }
/* here to plot array front view only */

xyplot()
       {
       int i,angl,f,camno,camsl,hite;
```

```
        printf("Enter starting slice.");
        scanf("%d",&angl);
        hite=height/VSTEP;
        grid(1);
        colour=4;
        setcol(colour);
        f=0;
        for(i=0;i<=STEPS/2;i++){
        plot2(angl,hite,i,f);
        f=1;
        angl++;
        if(angl>STEPS)angl=angl-STEPS;
        }
        }
/* here to plot array front view only without screen redo */

xyplot2(angl)
        int angl;
        {
        int i,l,f,hite;
        double mm;

        hite=height/VSTEP;
        wndow=2;
        usewdw(wndow);
        colour=4;
        setcol(colour);
                set_lt(0xffff);
                set_dm(0);      /* complement */
                setcol(4);
          mm=height;                    /* convert height to double */
          nplot(15,15,mm);
                for(i=1;i<6;i++){               /* plot horizontal grid */
                mm=50*i+height;
                l=(50*i+5)*y_scale;
                nplot(15,l,mm);
                }
 f=1;
```

```
        for(i=0;i<=STEPS/2;i++){
        plot2(angl,hite,i,f);
        angl++;
        if(angl>STEPS)angl=angl-STEPS;
        }
        }
/* here to define windows */

window(number)
        int number;
        {
        int xs,ys,xe,ye,sw;
        float a11,a12,a21,a22;

        if(number==1){
         xs=10;
         ys=10;
         xe=xs+750;
          ye=ys+65*y_scale;
        defwdw(xs,ys,xe,ye,number);   /* define window */
        usewdw(number);              /* chose window */
        a11=0.8;
        a12=0;
        a21=0;
        a22=0.8;
        setcol(2);              /* set colour */
        set_lt(0xffff);                 /* set line type */
        def_mc(a11,a12,a21,a22);
        sw=1;
        sw_mc(sw);
        usefnt(font_0);
        set_dz(1);                      /* set zoom */
        set_dm(0);                      /* set draw mode */
         }
        else if(number==2){
         xs=10;
         ys=200;
         xe=xs+750;
```

```
 ye=ys+300*y_scale;
defwdw(xs,ys,xe,ye,number);  /* define window */
usewdw(number);              /* chose window */
a11=0.8;
a12=0;
a21=0;
a22=0.8;
setcol(2);              /* set colour */
set_lt(0xffff);          /* set line type */
def_mc(a11,a12,a21,a22);
sw=1;
sw_mc(sw);
usefnt(font_0);
set_dz(1);                   /* set zoom */
set_dm(0);                   /* set draw mode */
 }
else if(number==3){
 xs=50;
 ys=0;
 xe=xs+700;
 ye=ys+573*w_scale;
defwdw(xs,ys,xe,ye,number);  /* define window */
usewdw(number);              /* chose window */
a11=0.8;
a12=0;
a21=0;
a22=0.8;
setcol(2);              /* set colour */
def_mc(a11,a12,a21,a22);
sw=1;
sw_mc(sw);
usefnt(font_0);
set_dz(1);                   /* set zoom */
set_dm(0);                   /* set draw mode */
 }
else if(number==4){
 xs=50;
 ys=0;
```

```
 xe=xs+700;
 ye=ys+573*y_scale;
defwdw(xs,ys,xe,ye,number);  /* define window */
usewdw(number);              /* chose window */
a11=0.8;
a12=0;
a21=0;
a22=0.8;
setcol(1);              /* set colour */
def_mc(a11,a12,a21,a22);
sw=1;
sw_mc(sw);
usefnt(font_0);
set_dz(1);                  /* set zoom */
set_dm(0);                  /* set draw mode */
 }
else if(number==5){
 xs=100;
 ys=100;
 xe=xs+600;
 ye=ys+450*w_scale;
defwdw(xs,ys,xe,ye,number);  /* define window */
usewdw(number);              /* chose window */
a11=0.8;
a12=0;
a21=0;
a22=0.8;
setcol(1);              /* set colour */
def_mc(a11,a12,a21,a22);
sw=1;
sw_mc(sw);
usefnt(font_0);
set_dz(1);                  /* set zoom */
set_dm(0);                  /* set draw mode */
 }
else if(number==6){
 xs=10;
 ys=100;
```

```
 xe=xs+750;
 ye=ys+450*w_scale;
defwdw(xs,ys,xe,ye,number);  /* define window */
usewdw(number);          /* chose window */
a11=0.8;
a12=0;
a21=0;
a22=0.8;
setcol(1);              /* set colour */
def_mc(a11,a12,a21,a22);
sw=1;
sw_mc(sw);
usefnt(font_0);
set_dz(1);                  /* set zoom */
set_dm(0);                  /* set draw mode */
 }
}
```

/* newfile.c */

```
#include "lassdefs"


      extern short NDATA[NARRAY];       /* normalised data array
*/
      extern int BYTES[4];        /* common for passing parms. */
      extern int CAMINFO[4];
      extern double TAPE[STEPS+1][2];       /* temp store for tape
                                            measurements */
      extern float HEADER[HEADSIZE];/* file header info */
      extern double CAL1[CALBLK*CAM];  /* calibration
                              array X,Y,Xslope,Yslope */
      extern int height,camera;
      extern double rad;
      extern int slicearray[STEPS+5][2];

/* here to save calibration in /h0/newdata/cal.nc */
```

```
savcal()
        {
        FILE *fopen(name,action),*file_ptr;
        register char *name,*action;
        char *string2,*string3;
        int i,cc,camsl;
        double *cl1;

        string2 = "/h0/newdata/cal.nc";
        string3 = "w";

        name = string2;
        action = string3;
        file_ptr=fopen(name,action);
        printf("%6d\n",file_ptr);
        cl1 = &CAL1[0];      /* point to final calibration array. */
                fwrite(cl1,8,CALBLK*CAM,file_ptr);
                fclose(file_ptr);
                }
/* here to load       calibration in /h0/newdata/cal.nc */


lodcal()
        {
        FILE *fopen(name,action),*file_ptr;
        register char *name,*action;
        char string1[50],*string2,*string3;
        int i,cc,camsl;
        double *cl1;

        string2 = "/h0/newdata/cal.nc";
        string3 = "r";

        name = string2;
        action = string3;
        file_ptr=fopen(name,action);
        printf("%6d\n",file_ptr);
        cl1 = &CAL1[0];      /* point to final calibration array. */
                fread(cl1,8,CALBLK*CAM,file_ptr);
```

```
            fclose(file_ptr);
            }
/* here to save data to a new file */

newsavcal()
      {
      FILE *fopen(name,action),*file_ptr;
      register char *name,*action;
      char string1[500],*string2;
      extern int steps,linesp;
      int i,cc;
      short *cyl;
      float *hed;

      string2 = "w";

      name = &string1[0];
      printf("Enter file name.\n");
      scanf("%s",name);
            cc=NARRAY; /* length of array. */
            printf("file size = %d\n",cc);
            HEADER[0]=NARRAY;
            HEADER[1]=linesp;
            HEADER[2]=CAMINFO[0];
            HEADER[3]=camera;
      action = string2;
      file_ptr=fopen(name,action);
      printf("%6d\n",file_ptr);
      hed=&HEADER[0];         /* point to header array. */
            fwrite(hed,4,HEADSIZE,file_ptr);
      cyl=&NDATA[0];          /* point to data array. */
            fwrite(cyl,2,cc,file_ptr);
            fclose(file_ptr);
            }
/* here to save data to a new text file */

textsav()
      {
```

```
        FILE *fopen(name,action),*file_ptr;
        register char *name,*action;
        char string1[500],*string2;
        int angl,hite,top,bottom,tophite,count;
        double r,angle,height,zero;
        int page,linecnt;
        short *cyl;

        string2 = "w";

        name = &string1[0];
        printf("Enter file name.\n");
        scanf("%s",name);
        printf("Enter bottom height (mm).");
        scanf("%d",&bottom);
        printf("Enter top height (mm).");
        scanf("%d",&tophite);
        page=1;
        linecnt=0;
        zero=0.0;
        bottom=bottom/VSTEP;
        tophite=tophite/VSTEP;
        action = string2;
        file_ptr=fopen(name,action);
        printf("%6d\n",file_ptr);
                if(linecnt%183==0){
                if(linecnt>0)fprintf(file_ptr,"\n\n\n\n");
                fprintf(file_ptr,"%d\n\n\n\n\n",page);
                page++;
                fprintf(file_ptr,"Radius  Angle  Height Radius
        Angle  Height  Radius Angle  Height\n\n");
        linecnt++;
                }
*/
        if(count>0){
fprintf(file_ptr,"%10.2f %6.2f %-8.0f",r/count,angle,height);
        }
else fprintf(file_ptr,"%10.2f %6.2f %-8.0f",zero,angle,height);
```

```
        linecnt++;
        if(linecnt%3==0)fprintf(file_ptr,"\n");
                }
                }
                fclose(file_ptr);
                }
/* here to save slice data to a new text file */


slicesav()
        {
        FILE *fopen(name,action),*file_ptr;
        register char *name,*action;
        char string1[500],*string2;
        int i,angl,hite,top,bottom,tophite,count,spacing;
        double xx,yy,height;

        string2 = "w";

        name = &string1[0];
        printf("Enter file name.\n");
        scanf("%s",name);
        printf("Enter bottom height (mm).");
        scanf("%d",&bottom);
        printf("Enter top height (mm).");
        scanf("%d",&tophite);
        printf("Enter slice spacing.(e.g., 3=every 3rd slice.)");
        scanf("%d",&spacing);
        bottom=bottom/VSTEP;
        tophite=tophite/VSTEP;
        action = string2;
        file_ptr=fopen(name,action);
        printf("%6d\n",file_ptr);
fprintf(file_ptr,"%d%d\n",STEPS,(tophite - bottom)/spacing+1);
         for(hite=bottom;hite<=tophite;hite=hite+spacing){
         grid(2);
         splot(hite);
        fprintf(file_ptr,"%d\n",hite*VSTEP);
                for(i=1;i<=STEPS;i++){
```

```
fprintf(file_ptr,"%d %d\n",slicearray[i][0],slicearray[i][1]);
                }
        }
        fclose(file_ptr);
        }
/* here to calculate volume */


volume()
        {
        int angl,hite,bottom,tophite,area,temp;
        double volume;

        printf("Enter bottom height (mm).");
        scanf("%d",&bottom);
        printf("Enter top height (mm).");
        scanf("%d",&tophite);
        bottom=bottom/VSTEP;
        tophite=tophite/VSTEP;
        area=0;
         grid(2);
         for(hite=bottom;hite<=tophite;hite++){
         area=area+splot(hite);
                }
        volume=area*VSTEP;
printf("volume=%f cubic metres.\n",volume/1000000000);
        }
/* here to load data file */


newlodcal()
                {
        FILE *fopen(name,actin),*file_ptr;
        register char *name,*action;
        extern int steps,linesp;
        char string1[50],*string2;
        int cc;
        short *cyl;
        float *hed;
```

```
        string2 = "r";

        name = &string1[0];
        printf("Enter file name.\n");
        scanf("%s",name);
        action = string2;
        file_ptr=fopen(name,action);
        printf("%6d\n",file_ptr);
        hed=&HEADER[0];                 /* point to data array. */
                fread(hed,4,HEADSIZE,file_ptr);
                cc=HEADER[0];
                linesp=HEADER[1];
                CAMINFO[0]=HEADER[2];
                camera=HEADER[3];
                height=HEADER[4];
                printf("cc=%6d\n",cc);
                steps=linesp*cc/FRAME;
        cyl=&NDATA[0];          /* point to data array. */
        fread(cyl,2,NARRAY,file_ptr);
                fclose(file_ptr);
        CAL1[120]=cc;
                }
/* here to list data in normalised array */


listdata()
        {
        int angl,hite,top,bottom,tophite,count;
        double r,angle,height;
        int linecnt,page;
        short *cyl;

        printf("Enter bottom height (mm).");
        scanf("%d",&bottom);
        printf("Enter top height (mm).");
        scanf("%d",&tophite);
        linecnt=0;
        page=1;
        bottom=bottom/VSTEP;
```

```
        tophite=tophite/VSTEP;
        for(angl=0;angl<STEPS;angl++){
        for(hite=bottom;hite<=tophite;hite++){
        cyl=NPOINT;          /* point to data array. */
        r=*cyl;
        cyl++;
        count=*cyl;
        cyl++;
        angle=angl*ANGLE;
        height=hite*VSTEP;
        if(linecnt%183==0){
        if(linecnt>0)printf("\n\n\n\n");
        printf("                    %d\n\n\n\n\n\n",page);
        page++;
        printf("      Radius Angle      Height              Radius
Angle       Height              Radius Angle      Height\n\n");
        }
        if(count>0){
        printf("%10.2f %6.2f %-8.0f",r/count,angle,height);
        linecnt++;
        if(linecnt%3==0)printf("\n");
        }
              }
              }
              }
/* here to plot on the printer. */


plotprint()
      {
      int angl,hite,top,bottom,tophite,count;
      int xx,scale,offset,xtab[STEPS/2];
      int gap,i,j,k,temp,stangle;
      short *cyl;
      double r,theta;

      offset=464; /* offset to centre of page 72=1 inch */
      scale=2;       /* 1mm=scale/72 inches */
      printf("Enter bottom height (mm).");
```

```
        scanf("%d",&bottom);
        printf("Enter top height (mm).");
        scanf("%d",&tophite);
        printf("Enter starting angle (degrees).");
        scanf("%d",&stangle);
        stangle=stangle/ANGLE;
        bottom=bottom/VSTEP;
        tophite=tophite/VSTEP;
        PRPLOT;
        for(hite=tophite;hite>=bottom;hite--){
                for(k=0;k<STEPS/2;k++){
                angl=stangle+k;
                if(angl>=STEPS)angl=angl-STEPS;
                cyl=NPOINT;          /* point to data array. */
                r=*cyl;
                cyl++;
                count=*cyl;
                cyl++;
                if(count>0){
                r=r/count;
                theta=k*ANGLE*rad-PI/2;
                xx=r*sin(theta)*scale+offset;
                xtab[k]=xx;
                }
/* here to Shell sort xtab[]. */
                for(gap=STEPS/4;gap>0;gap=gap/2){
                        for(i=gap;i<STEPS/2;i++){
                for(j=i-gap;j>=0 && xtab[j]>xtab[j+gap];j=j-gap){
                        temp=xtab[j];
                        xtab[j]=xtab[j+gap];
                        xtab[j+gap]=temp;
                        }
                        }
                }
                }
/* here to print xtab[i] */
                for(i=0;i<STEPS/2;i++){
                temp=xtab[i];
```

```
                if(temp >0 && temp<(2*offset)){
                printf("\033F%4d",temp); /* tabs to temp dot
                places from margin */
                printf("\033∀       18");
                                /* puts ASCII 8 on print wires */
        }
        }
        printf("\n");
    }
    printf("\033c"); /* resets printer */
    }
```

/* xyzmouse.c */

```
#ij clude "lassdefs"

    extern short NDATA[NARRAY];
                                    /* normalised data array */
    extern int mouse_port,mouse_x,mouse_y;
    extern char  mouse_buttons;
    extern double w_scale,x_scale,y_scale,rad;
    extern int linetype,colour,wndow,zoom;
    extern int height,pltflg;
    extern double slicexy[4*NFRAME][2];
/* here to point with the mouse in cartesian co-ords. */

mousexyz()
    {
    int i,x,y,z,xx,yy,zz,yl,flag,fb,xmin,xmax,ymin,ymax;
    int angl,hite,dcnt,xp,yp,slicecnt;
    short *np;
    double r,s,theta,angle,xxx,yyy,zzz,qqq,rrr,sss;
    double rthetah[3][3];
    double xyz[3][3];
    double sintheta,costheta,sina,cosa,sinb,cosb,sinc,cosc;
    double temp1,temp2,denom;
    fb=0;
    x=150*x_scale;
```

```
        y=150*y_scale;
        z=350*x_scale;
        xx=x;
        yy=y;
        zz=z;
        yl=65*y_scale;
        flag=0;
        xmin=0;
        xmax=500*x_scale;
        ymin=0;
        ymax=300*y_scale;
        dcnt=0;
        angl=STEPS/2;
        slicecnt=1;
                setcol(4);
                set_lt(0xffff);
                set_dm(1);      /* complement */
                vector(xmin,y,xmax,y);   /* write start position */
                vector(x,ymin,x,ymax);
                vector(z,ymin,z,ymax);
    .     do{
           read_mouse();
if((mouse_x !=0) | | (mouse_y !=0) | | (mouse_buttons !=0)){
                 if(flag<100){
                 setcol(4);
                 set_lt(0xffff);
                  set_dm(1);     /* complement */
                  vector(xmin,y,xmax,y);   /* rub out old lines */
                  vector(x,ymin,x,ymax);
                  vector(z,ymin,z,ymax);
                x=xx;
                y=yy;
                z=zz;
                 }
                x=x+mouse_x;
                y=y+mouse_y;
                  if(y<height && flag==0){
                              /* here to change to lower window */
```

```
                usewdw(1);
               set_lt(0xffff);
               set_dm(1);      /* complement */
               setcol(4);
                flag=1;
                z=-1;
                y=yl;
                }
                else if(y>yl && flag==1){
                              /* here to change to upper window */
                usewdw(wndow);
               set_lt(0xffff);
               set_dm(1);      /* complement */
                flag=0;
                y=height;
                z=350*x_scale;
                }
/* here to move up */
                if((mouse_buttons & 2)>0 && y>93 && y<158 &&
                                                x>375 && flag==1){
                flag=100;
                 height=height+150*zoom;
                y=height;
                xyzplot(height,fb,pltflg);
                }
/* here to move down */
                else if((mouse_buttons & 2)>0 && y>43 && y<93
                                              && x>375 && flag==1){
                flag=100;
                 height=height-(150*zoom);
                y=height;
                xyzplot(height,fb,pltflg);
                }
/* here to zoom down */
                if((mouse_buttons & 2)>0 && y>43 && y<93 &&
                                                x<375 && flag==1){
                usewdw(wndow);
                set_lt(0xffff);
```

```
                set_dm(1);      /* complement */
                flag=100;
                y=0;
                 zoom=zoom+zoom/2;
                 if(zoom/2<1)zoom=zoom+1;
                xyzplot(height,fb,pltflg);
                }
/* here to zoom up */
                else if((mouse_buttons & 2)>0 && y>93 && y<158
                                                && x<375 && flag==1){
                usewdw(wndow);
                set_lt(0xffff);
                 set_dm(1);     /* complemeent */
                flag=100;
                y=0;
                 zoom=zoom-zoom/2;
                 if(zoom<1)zoom=1;
                xyzplot(height,fb,pltflg);
                }
/* here to flip */
                else if((mouse_buttons & 2)>0 && flag==0){
                if(fb!=0)fb=0;
                else fb=1;
                xyzplot(height,fb,pltflg);
                }
/* here to point to r,theta,h */
                xx=x;
                yy=y;
                zz=z;
                if(flag==0){
                angl=x/3;
                      if(angl>STEPS)angl=angl-STEPS;
                      else if(angl<0)angl=STEPS-angl;
                hite=y/VSTEP; /* therefore y=actual heigh */
                np=NPOINT;
                r=*np;
                s=*(np+1);
                      if(s==0)r=0;
```

```
                else r=r/s;
            angle=angl*ANGLE;
            theta=angle*rad;
                if(fb==0){
                x=(r*sin(theta)/zoom+150)*x_scale;
                z=(r*cos(theta)/zoom+350)*x_scale;
                }
                else {
                x=(150-r*sin(theta)/zoom)*x_scale;
                z=(350-r*cos(theta)/zoom)*x_scale;
                }
            y=(y-height)*y_scale/zoom;
            }
/* here to print x,y */
            if((mouse_buttons & 4)>0){
            setcol(7);
            set_dm(0);
             rthetah[dcnt][0]=r;
             rthetah[dcnt][1]=theta;
             rthetah[dcnt][2]=hite*VSTEP;
             xyz[dcnt][0]=r*cos(theta);
             xyz[dcnt][1]=r*sin(theta);
             xyz[dcnt][2]=hite*VSTEP;
            printf("r=%g theta=%g
                        height=%d\n",r,angle,hite*VSTEP);
            printf("cnt=%d x=%g y=%g z=%g\n",dcnt,
                        xyz[dcnt][0],xyz[dcnt][1],xyz[dcnt][2]);
             set_lt(0);
             chrcur(700,700);
             dischr(127);
             set_lt(0xffff);
             chrcur(700,700);
             dischr(dcnt+49);
             dcnt++;
             if(dcnt>2) {
                for(i=0;i<3;i++){
                xyz[1][i]=xyz[1][i]-xyz[0][i];
                xyz[2][i]=xyz[2][i]-xyz[0][i];
```

```
        }
        for(i=0;i<3;i++){
 printf("x=%g y=%g z=%g\n",
                        xyz[i][0],xyz[i][1],xyz[i][2]);
        }
 denom=sqrt(xyz[1][1]*xyz[1][1]+xyz[1][2]*xyz[1][2]);
  printf("denom=%g\n",denom);
 sina=-xyz[1][2]/denom;
 cosa=xyz[1][1]/denom;
 temp1=(xyz[1][1]*cosa-xyz[1][2]*sina);
 temp2=xyz[1][0];
  denom=sqrt(temp2*temp2+temp1*temp1);
  printf("temp1=%g temp2=%g denom=%g\n",
                                temp1,temp2,denom);
 sinb=-xyz[1][0]/denom;
 cosb=(xyz[1][1]*cosa-xyz[1][2]*sina)/denom;
 temp1=(xyz[2][1]*sina+xyz[2][2]*cosa);
 temp2=xyz[2][0]*cosb+sinb*(xyz[2][1]*
                        cosa-xyz[2][2]*sina);
  denom=sqrt(temp2*temp2+temp1*temp1);
  printf("temp1=%g temp2=%g denom=%g\n",
                        temp1,temp2,denom);
 sinc=-(xyz[2][1]*sina+xyz[2][2]*cosa)/denom;
 cosc=(xyz[2][0]*cosb+sinb*(xyz[2][1]*cosa-
                        xyz[2][2]*sina))/denom;
printf("sa=%g ca=%g sb=%g cb=%g sc=%g
        cc=%g\n",sina,cosa,sinb,cosb,sinc,cosc);
        xxx=xyz[1][0]; /* test only */
        yyy=xyz[1][1];
        zzz=xyz[1][2];
        qqq=(xxx*cosb+(yyy*cosa-zzz*sina)*
                sinb)*cosc-(yyy*sina+zzz*cosa)*sinc;
        rrr=(yyy*cosa-zzz*sina)*cosb-xxx*sinb;
        sss=(xxx*cosb+(yyy*cosa-zzz*sina)*
                sinb)*sinc+(yyy*sina+zzz*cosa)*cosc;
printf("qqq=%g rrr=%g sss=%g\n",qqq,rrr,sss);
        xxx=xyz[2][0]; /* test only */
        yyy=xyz[2][1];
```

```
                    zzz=xyz[2][2];
                    qqq=(xxx*cosb+(yyy*cosa-zzz*sina)*
                          sinb)*cosc-(yyy*sina+zzz*cosa)*sinc;
                    rrr=(yyy*cosa-zzz*sina)*cosb-xxx*sinb;
                    sss=(xxx*cosb+(yyy*cosa-zzz*sina)*sinb)*
                          sinc+(yyy*sina+zzz*cosa)*cosc;
printf("qqq=%g rrr=%g sss=%g\n",qqq,rrr,sss);/* end of test */
                    for(angl=0;angl<STEPS;angl++){
                    theta=angl*ANGLE*rad;
                    sintheta=sin(theta);
                    costheta=cos(theta);
                      for(hite=0;hite<NFRAME;hite++){
                     np=NPOINT;
                     r=*np;
                     s=*(np+1);
                    if(s==0)r=0;
                    else r=r/s;
                          if(r!=0){
                          xxx=r*costheta-xyz[0][0];
                          yyy=r*sintheta-xyz[0][1];
                          zzz=hite*VSTEP-xyz[0][2];
                          sss=(xxx*cosb+(yyy*cosa-zzz*sina)*
                          sinb)*sinc+(yyy*sina+zzz*cosa)*cosc;
                            if(sss>-VSTEP/2 && sss<VSTEP/2){
                            qqq=(xxx*cosb+(yyy*cosa-zzz*sina)*
                               sinb)*cosc-(yyy*sina-zzz*cosa)*sinc;
                         rrr=(yyy*cosa-zzz*sina)*cosb-xxx*sinb;
                                if(qqq !=0 | | rrr !=0 ){
                                slicexy[slicecnt][0]=qqq;
                                slicexy[slicecnt][1]=rrr;
                                slicecnt++;
                                xp=(qqq/zoom+250)*x_scale;
                                yp=(rrr/zoom+150)*y_scale;
                                }
                           }
                          }
                     }
                    }
```

```
                dcnt=0;
                slicexy[0][0]=slicecnt;
                xyslplot();
for(read_mouse();(mouse_buttons & 2)==0;read_mouse());
                xyzplot(height,fb);
                slicecnt=1;
                }
              }
/* here to return to menu */
                if((mouse_buttons & 1)>0 ){
                flag=2;
                }
                setcol(4);
                set_lt(0xffff);
                set_dm(1);     /* complement */
              vector(xmin,y,xmax,y);   /* write new lines */
              vector(x,ymin,x,ymax);
              vector(z,ymin,z,ymax);
              if(flag>99)flag=flag-100;
              }
    }
      while(flag !=2);
       setcol(1);
       tsleep(5);
       usewdw(wndow);
       setcol(colour);
       set_lt(linetype);
      }
/* here to point with the mouse. */


mousepnt()
      {
      int i,x,y,xx,yy,yl,flag,fb,xmin,xmax,ymin,ymax;
      int angl,hite,dcnt,xp,yp,slicecnt,phi;
      short *np;
      double r,s,angle,theta,xxx,yyy,qqq,rrr,sss;
      double rthetah[3][3];
      double xyz[3][3];
```

```
        double sintheta,costheta,sina,cosa,sinb,cosb,sinc,cosc;
        double temp1,temp2,denom;
        fb=0;
        x=150*x_scale;
        y=150*y_scale;
        xx=x;
        yy=y;
        yl=65*y_scale;
        flag=0;
        xmin=0;
        xmax=500*x_scale;
        ymin=0;
        ymax=300*y_scale;
        dcnt=0;
        phi=0;
        slicecnt=1;
                setcol(4);
                set_lt(0xffff);
                set_dm(1);      /* complement */
                vector(xmin,y,xmax,y);   /* write start position */
                vector(x,ymin,x,ymax);
         do{
         read_mouse();
if((mouse_x !=0) || (mouse_y !=0) || (mouse_buttons !=0)){
                 if(flag<100){
                 setcol(4);
                 set_lt(0xffff);

                 set_dm(1);     /* complement */
                 vector(xmin,y,xmax,y);   /* rub out old lines */
                 vector(x,ymin,x,ymax);
                x=xx;
                y=yy;
                 }
                x=x+mouse_x;
                y=y+mouse_y;
                 if(y<height && flag==0){
                                /* here to change to lower window */
```

```
                usewdw(1);
                flag=1;
                y=yl;
                }
                else if(y>yl && flag==1){
                                /* here to change to upper window */
                usewdw(wndow);
                flag=0;
                y=height;
                x=250*x_scale;
                }
/* here to move up */
                if((mouse_buttons & 2)>0 && y>93 && y<158 &&
                                                x>375 && flag==1){
                grid(1);
                flag=100;
                y=0;
                height=height+150;
                xyplot2(phi);
                }
/* here to move down */
                else if((mouse_buttons & 2)>0 && y>43 && y<93
                                        && x>375 && flag==1){
                grid(1);
                flag=100;
                y=0;
                height=height-150;
                xyplot2(phi);
                }
/* here to rotate anticlockwise */
                if((mouse_buttons & 2)>0 && y>43 && y<93 &&
                                        x<375 && flag==1){
                grid(1);
                flag=0;
                y=0;
                phi=phi-STEPS/8;
                if(phi<0)phi=STEPS-phi;
                xyplot2(phi);
```

```
                }
/* here to rotate clockwise */
                else if((mouse_buttons & 2)>0 && y>93 && y<158
                                        && x<375 && flag==1){
                grid(1);
                flag=0;
                y=0;
                phi=phi+STEPS/8;
                if(phi>STEPS)phi=phi-STEPS;
                xyplot2(phi);
                }
/* here to plot slice */
                else if((mouse_buttons & 2)>0 && flag==0){
                clrall();
                grid(2);
                splot(hite);
                mouse_buttons=0;
                while((mouse_buttons & 2)==0)read_mouse();
                clrall();
                grid(1);
                grid(5);
                xyplot2(phi);
                }
/* here to point to r,theta,h */
                xx=x;
                yy=y;
                        if(flag==0){
                        angl=x/3;
                        if(angl>STEPS)angl=angl-STEPS;
                        else if(angl<0)angl=STEPS-angl;
                        hite=y/VSTEP; /* therefore y=actual height */
                        np=NPOINT;
                        r=*np;
                        s=*(np+1);
                        if(s==0)r=0;
                        else r=r/s;
                        angle=angl*ANGLE;
                        theta=angle*rad;
```

```
                    if(fb==0){
                     x=(r*cos(theta)+250)*x_scale;
                    }
                    else {
                     x=(250-r*cos(theta))*x_scale;
                    }
                   y=(y-height)*y_scale;
                }
/* here to return to menu */
                 if((mouse_buttons & 1)>0 ){
                 flag=2;
                 }
                 setcol(4);
                 set_lt(0xffff);
                 set_dm(1);     /* complement */
               vector(xmin,y,xmax,y);   /* write new lines */
               vector(x,ymin,x,ymax);
               if(flag>99)flag=flag-100;
               }
        }
    while(flag !=2);
      setcol(1);
      tsleep(5);
      usewdw(wndow);
      setcol(colour);
      set_lt(linetype);
     }
```

MACHINE CODE PROGRAMS

/*newapalset.a */

```
            NAM newapalset.a
            ttl trap call prog.
            use          /dd/defs/oskdefs.d
```

```
            psect

trapno      equ 5
intname     dc.b        "qtrapo:",0



gset:
            movem.l     d4/a0/a1,-(a7)
            move.w      #trapno,d0          trap number to
                                            assign
            move.l      #0,d1               no optional
                        memory override
            lea         ((intname).l,pc),a0 get address of
                                            int. prog.
            os9         F$TLink             link to trap
                                               handler
            bcs.s       wrong               check if ok
            movem.l     (a7)+,d4/a0/a1
            moveq       #0,d1
            rts


gcall:
            movem.l     a2/a5,-(a7)
                         movea.l    d0,a5   get array
                                               address
                        lea
((CAMINFO).l,a6),a2  get camera                        list
address
            tcall     trapno,0              trap to prog
            bcs.s       wrong1
            movem.l     (a7)+,a2/a5
            rts


ccall
            movem.l     a2/a5,-(a7)
```

```
                movea.l   d0,a5                  get array
address
                lea       ((CAMINFO).l,a6),a2    get
camera list address .
                tcall     trapno,1               trap to prog
                bcs.s     wrong1
                movem.l   (a7)+,a2/a5
                rts

scall:
                tcall     trapno,2               trap to prog
                bcs.s     wrong1
                rts

pcall:
                tcall     trapno,3               trap to prog
                bcs.s     wrong1
                rts

c2call:
                lea       ((port2c).l,a6),a2     get address of
                                                 data      space
                tcall     trapno,4               trap to prog
                bcs.s     wrong1
                rts

wrong:
                movem.l   (a7)+,d4/a0/a1

wrong1:
                os9       F$Exit

unpack:
                movem.l   a4/a5,-(a7)
                movea.l   d0,a5      address of byte to unpack
```

```
        lea         ((BYTES).l,a6),a4       get address of
unpack list
        move.l      (a5),d1                 get data
        rol.l       #8,d1                   move next bits
                                            to lsb
        rol.l       #2,d1                   move next bits
                                            to lsb
        move.l      d1,d0
        andi.l  #$3ff,d0                    mask to 10 bits
        move.l      d0,(a4)+                save radius
        rol.l       #8,d1                   move next bits
                                            to lsb
        rol.l       #1,d1                   move next bits
                                            to lsb
        move.l      d1,d0
        andi.l      #$1ff,d0                mask to 9 bits
        move.l      d0,(a4)+                save line count
        rol.l       #8,d1                   move next bits
                                            to lsb
        move.l      d1,d0
        andi.l      #$ff,d0                 mask to 8 bits
        move.l      d0,(a4)+                save slice count
        rol.l       #5,d1                   move next bits
                                            to lsb
        move.l      d1,d0
        andi.l      #$1f,d0                 mask to 5 bits
        move.l      d0,(a4)                 save camera
                                            info
        movem.l     (a7)+,a4/a5
        rts


packcart:
        movem.l     d5/a4/a5,-(a7)
        movea.l     d0,a5       address of byte to pack
        lea         ((BYTES).l,a6),a4       get address of
unpack list
        move.l      (a4)+,d0                get x
        andi.l      #$3ff,d0                mask to 10 bits
```

```
        move.l    d0,d5                    put in d5
        rol.l     #6,d5      make room for next bits
        rol.l     #6,d5      make room for next bits
        move      (a4)+,d0                 get y
        andi.l    #$fff,d0                 mask to 12 bits
        or.l      d0,d5                    insert y into d5
        rol.l     #5,d5      make room for next bits
        rol.l     #5,d5      make room for next bits
        move.l    (a4)+,d0                 get z
        andi.l    #$3ff,d0                 mask to 10 bits
        or.l      d0,d5                    insert z into d5
        move.l    d5,(a5)
        movem.l   (a7)+,d5/a4/a5
        rts

unpkcart:
        movem.l   a4/a5,-(a7)
        movea.l   d0,a5      address of byte to unpack
        lea       ((BYTES).l,a6),a4        get address of
        unpack list
        bfexts    (a5){0:10},d0            unpack x
        move.l    d0,(a4)+                 put inlist
        bfexts    (a5){10:12},d0           unpack y
        move.l    d0,(a4)+                 put in list
        bfexts    (a5){22:10},d0           unpack z
        move.l    d0,(a4)+                 put in list
        movem.l   (a7)+,a4/a5
        rts

packpola:
        movem.l   d5/a4/a5,-(a7)
        movea.l   d0,a5    address of byte to pack
        lea       ((BYTES).l,a6),a4        get address of
        unpack list
        move.l    (a4)+,d0                 get r
        andi.l    #$7ff,d0                 mask to 11 bits
        move.l    d0,d5                    put r in d5
        rol.l     #5,d5       make room for next bits
```

```
        rol.l       #5,d5       make room for next bits
        move.l      (a4)+,d0                get theta
        andi.l      #$3ff,d0                mask to 10 bits
        or.l        d0,d5                   put theta in d5
        rol.l       #6,d5       make room for next bits
        rol.l       #5,d5       make room for next bits
        move.l      (a4)+,d0                get h
        andi.l      #$7ff,d0                mask to 11 bits
        or.l        d0,d5                   put h in d5
        move.l      d5,(a5)
        movem.l     (a7)+,d5/a4/a5
        rts

unpkpola:
        movem.l     a4/a5,-(a7)
        movea.l     d0,a5       address of byte to unpack
        lea         ((BYTES).l,a6),a4       get address of
        unpack list
        bfextu      (a5){0:11},d0           unpack r
        move.l      d0,(a4)+                put in list
        bfextu      (a5){11:10},d0          unpack theta
        move.l      d0,(a4)+                put in list
        bfextu      (a5){21:11},d0          unpack h
        move.l      d0,(a4)+                put in list
        movem.l     (a7)+,a4/a5
        rts
        vsect

buffer
        ds.b 64
        ends

#define PORT_INIT


/* ELTEC-mouse-100 program */
#include <stdio.h>
```

```
#include <sgstat.h>
#include <modes.h>

#define PORT        "/t1"      /* mouse port */
#define UPDATE      S_IREAD+S_IWRITE    /* access mode */
#define TIMEOUT     1000              /* for mouse_request () */
#define BAUD_RATE   0x0e              /* baudrate = 9600 */
#define PARITY      0X00             /* 1 stop bit */
#define DATA_FORMAT "T"                /* three byte packed binary */
#define REPORT_MODE "D"                /* select prompt mode */
#define PROMPT      "P"               /* prompt */
#define BREAK       0x01              /* break condition */

struct sgbuf       new_mode;         /* buffer for new device mode */
struct sgbuf       old_mode;         /* buffer for old device mode */

struct mouse_data{                    /* this structure depends on the /*
                          /* used DATA_FORMAT  */
   char buttons;                      /* here : */
   char x;                           /* 'Three Byte Packed Binary Format' */
   char y;
   char dummy;
};

int mouse_port;
int mouse_x;
int mouse_y;
char mouse_buttons;

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



init_mouse()
/*
* initialization of the mouse port
*
* after opening the apprppriate port, this routine will first read
* the old status of the port and save it in a buffer, then it will
```

```
* set the right device parameters.
*/

{
  int   n;
  char  dummy[256];

   mouse_port = open (PORT,UPDATE);      /* open mouse path for
update */

#ifdef PORT_INIT
   getstat (0, mouse_port, &old_mode);  /* save old mode */
   getstat (0, mouse_port, &new_mode);  /* get old mode to change */

  new_mode.sg_baud          =BAUD_RATE;
  new_mode.sg_parity    = PARITY;
  new_mode.sg_nulls         = 0;
  new_mode.sg_echo          = 0;
  new_mode.sg_pause     = 0;
  new_mode.sg_xon           = 0;
  new_mode.sg_xoff          = 0;
  new_mode.sg_kbich         = 0;
  new_mode.sg_eorch         = 0;
  new_mode.sg_eofch     = 0;
  new_mode.sg_kbach     = 0;
  new_mode.sg_dlnch         = 0;
  new_mode.sg_rlnch         = 0;
  new_mode.sg_psch          = 0;
  new_mode.sg_bspch         = 0;
  new_mode.sg_bsech         = 0;

  setstat  (0, mouse_port, &new_mode);   /* set new mode */
#endif

  write   (mouse_port, DATA_FORMAT, 1);  /* data format */
  write   (mouse_port, REPORT_MODE, 1);  /* report mode */

  while   ((n = _gs_rdy (mouse_port)) > 0) /* flush input buffer */
```

```
        read (mouse_port, dummy, n);
}
/*********************************************/


mouse_request()

{
  int n,cnt;
  char dum_buff[4];

  do
  {
    write (mouse_port, PROMPT,1);
   n=TIMEOUT;

    while (((cnt=_gs_rdy (mouse_port))<3) && (n != 0))
       n--;

    if(n==0)printf("timeout %d\n",cnt);
   if((n==0) && (cnt>0)){
    cnt=_gs_rdy(mouse_port);
    read(mouse_port,dum_buff,cnt);
   }
  }
  while (n==0);
}
/*********************************************/


read_mouse()
/*
* interpretation of the mouse data
* this routine reads the mouse data and interprets this data
*/

{
   struct mouse_data input;
```

```
  do{

    mouse_request();

    read(mouse_port, &input, 3);

    mouse_buttons = (char) input.buttons;
    mouse_x       = (char) input.x;
    mouse_y       = (char) input.y;
  }
   while ((mouse_buttons == 0) && (mouse_x == 0) &&    (mouse_y
== 0));

   if(mouse_buttons !=0){
    do{
      mouse_request();
    read (mouse_port, &input, 3);
    }
    while (input.buttons != 0);
  }
}
/*********************************************/



finito_mouse()
/*
* restore old device parameters
*/

{
#ifdef PORT_INIT
   setstat (0, mouse_port, &old_mode);
#endif
   close (mouse_port);
}
/*********************************************/
```

APPENDIX 6

Lass program listings.

The following are all the source files which when compiled together will produce the Lass program.

/* lassdefs */

```
#include <stdio.h>
#include <math.h>
#define RESOLUTION  1 /* high=1 low=0 */
#if RESOLUTION
#define ANGLE 2.4 /* angular increment */
#define STEPS 150 /* 360/ANGLE steps per rev */
#define VSTEP 5   /* vertical step height */
#define NARRAY 126000 /* NFRAME*STEPS*2 size of normalised data array */
#define SLSPACE 30 /* STEPS/VSTEP */
#define NFRAME 420 /* no. of vertical intervals in 2.1 metres 2100/VSTEP */
#define PRPLOT printf("\033n\033T20")  /* sets line feed scale on printer */
#define DENS 2 /* density function for patches */
#define ARMTOL 15 /* limit for arm removal */
#else
#define ANGLE 5.0 /* angular increment */
#define STEPS 72 /* 360/ANGLE steps per rev */
#define VSTEP 10 /* vertical step height */
#define NARRAY 30240 /* NFRAME*STEPS*2 size of normalised data array */
#define SLSPACE 15 /* STEPS/VSTEP */
#define NFRAME 210 /* no. of vertical intervals in 2.1 metres 2100/VSTEP */
```

```
#define PRPLOT printf("\033n\033T40")  /* sets line feed scale on
printer */
#define DENS 1 /* density function for patches */
#define ARMTOL 20 /* limit for arm removal */
#endif
#define ARRAY FRAME*CAM*2*STEPS /* memory required for raw
data */
#define CAM 14      /* no. of cameras */
#define CAMDIST 2900 /* camera to centre distance */
#define FRAME 312   /* no of lines per frame */
#define TOPBLANK 25 /* no of hidden lines at top of screen */
#define STEPREV 72000 /* no of steps per rev */
#define ARDIM (FRAME-20)*2
#define ZERO 75 /*57*/
#define NPTS 16 /* number of points in curve fit polygon */
#define XMAX 500 /* max measuring range in X direction. */
#define YMAX 400 /* max measuring range in Y direction. */
#define CALBLK 140 /* size of calibration block per camera */
#define THETA1 CAL1[121+CALBLK*(camsl-1)] /* angle of leading
shadow */
#define THETA2 CAL1[122+CALBLK*(camsl-1)] /* angle of trailing
shadow */
#define SINE1  CAL1[123+CALBLK*(camsl-1)] /* sine of THETA1 */
#define SINE2  CAL1[124+CALBLK*(camsl-1)] /* sine of THETA2 */
#define COS1   CAL1[125+CALBLK*(camsl-1)] /* cos of THETA1 */
#define COS2   CAL1[126+CALBLK*(camsl-1)] /* cos of THETA2 */
#define ZERO1  CAL1[127+CALBLK*(camsl-1)] /* zero of shadow 1 */
#define ZERO2  CAL1[128+CALBLK*(camsl-1)] /* zero of shadow 2 */
#define HEIGHT CAL1[129+CALBLK*(camsl-1)] /* height of lowest
calibration light */
#define KK CAL1[130+CALBLK*(camsl-1)]     /* camera distance */
#define NPOINT (angl*NFRAME+hite)*2 /* point to array */
#define XCOUNT 590 /* scale range in X direction. */
#define YCOUNT 312 /* scale range in Y direction. */
#define CZERO 180   /* center zero offset */
#define PI 3.141592654
#define RAD PI/180
#define HEADSIZE 140 /* no of elements in file header */
```

```
#define SCALE 50 /* window to plot scale factor */
#define PLUSX 10 /* window scale */
#define MINUSX -10 /* window scale */
#define PLUSY 7.5 /* window scale */
#define MINUSY -7.5 /* window scale */
#define PLUSZ 10 /* window scale */
#define MINUSZ -10 /* window scale */
#define PATCHMAX 1600 /* size of patches array */
#define MAXSLICES 100 /* size of slices array */
#define BLANK 0
#define TRANSX 1
#define TRANSY 2
#define TRANSZ 3
#define ROTX 4
#define ROTY 5
#define ROTZ 6
#define ZOOM 7
#define RESET 8
#define XFILE 9
#define EXITTRANS 10
#define CURVE 11
#define XYZERO 12
#define LINES 13
#define IRIS 14
#define IRIS2 15
#define SKIN 16
#define VOLUME 17
#define X1SET 19
#define X2SET 20
#define Y1SET 21
#define Y2SET 22
#define FB 23
#define SETZERO 24
#define SSAVE 25
#define PPLOT 26
#define ARMS 27
#define MARKS 28
#define OBLIQUE 29
```

```
#define SET1 30
#define SET2 31
#define LINESTEP 32
#define DPX0 34
#define DPY0 35
#define DP1 36
#define DP2 37
#define DP3 38
#define DP4 39
#define DP5 40
#define DP6 41
#define DP7 42
#define DP8 43
#define DP9 44
#define DP10 45
#define DP11 46
#define DP12 47
#define DP13 48
#define DP14 49
#define DPX15 50
#define DPY15 51
#define UP 52
#define UP5 53
#define FITP 54
#define DOWN 55
#define SAVE 56
#define PSAVE 57
#define PLOAD 58
#define RECALL 59
#define DELTAX 60
#define DELTAY 61
#define PPLOT2 62
#define PMPRNT 63
#define DPZERO 64
#define CFIT 65
#define SPLINE 66
#define PCLEAR 67
#define DAT1 68
```

```
#define DAT2 69
#define SMES 70
#define PBPLOT 71
#define SET3 72
#define SET4 73
#define TGL 74
#define INP 75
#define PTCH 76
#define LPP 77
#define LPSAVE 78
#define LPLOAD 79
#define LCLEAR 80
#define ROTPX 81
#define ROTPY 82
#define ROTPZ 83
#define ROTMX 84
#define ROTMY 85
#define ROTMZ 86
#define TRANSPX 87
#define TRANSPY 88
#define TRANSPZ 89
#define TRANSMX 90
#define TRANSMY 91
#define TRANSMZ 92
#define ZOOMP 93
#define ZOOMM 94
#define SPANG 95
#define MIRROR 96
#define GETSL 97
#define CNTUP 98
#define CNTDN 99
#define KEYS1 100
#define KEYS2 101
#define ZFLG 102
#define REDO 103
#define ELIPSE 104
#define MLOAD 105
#define SETU 106
```

```
#define REGEN 107
#define SAVCNT 108
#define GETCNT 109
#define SHOCNT 110
#define TAPEFLG 111
#define TCIRC 112
#define REV 113
```

/* lass1.c */

```
#include "device.h"
#include "lassdefs"
#include <gl.h>
#include <gl/gl.h>
#include <gl/device.h>
```

```
        short NDATA[NARRAY]; /* normalised data array */
        float HEADER[HEADSIZE]; /* file header info */
        float WAND[20][3];
        float DPOLY[NPTS+1][2];
        float bslice[NFRAME][NPTS][2];
        float measblock[NPTS];
        long barwin, cubewin, slicewin, vertwin, keywin;
        long datawin, nurbswin, textwin;
        float transx,transy,transz,deltax,deltay;
        float zoom,sliceht,set1,set2,linesteps,lastheight;
        float datum1,datum2,pmax,pmin;
        int rotx,roty,rotz,count,spangcnt;
        int steps,linesp,dcount,scount,dcnttemp;
        int maxhite, minhite, maxangl, minangl,stangle;
        int lineflg,polyflg,clearflg,fbflg,cartdataflg;
        int patchflg,rotyflg,armflg,markflg;
        int splineflg,kbdflg,mflg,zflg,tapeflg;
        int revflg;
        short xcurs, ycurs;
        char *name;
```

```
        char string[64];
        float irismat[4][4];
        float datablock[20];
        short blackvec[3] = { 0, 0, 0};
        short redvec[3] = { 255, 0, 0};
        short greenvec[3] = { 73, 206, 130};
        short bluevec[3] = { 0, 0, 255};
        short ltbluevec[3] = { 0, 142, 255};
        short yellowvec[3] = { 255, 255, 0};
        short pinkvec[3] = { 255, 121, 130};
        short purplevec[3] = { 210, 146, 241};
        short greyvec[3] = { 150, 150, 150};
        short whitevec[3] = {255,255,255};
        float  MINIHEAD[20];
        float patches[PATCHMAX][3][4][4];
        float slices[MAXSLICES][NPTS+1][3];
        float interpsl[4*MAXSLICES][NPTS+1][3];
        float uvalue;
        int patchcount,bspcnt;

        extern int obdatacnt;
        extern float trans[4][4];


main()
{
   Device val;
   int command, i,j,m, wscale, exitflg, mouseflg, hite;
   int flg,inttemp;
   float transparam;
    float minval,maxval;
   float zoomtemp,temp;


        transx = 0; transy = MINUSY; transz = 0;
        deltax =0; deltay =0;
        zoom=1;
        rotx=0; roty=0; rotz=0;
```

```
        wscale=SCALE;
        stangle=0;
        minangl=stangle/ANGLE-STEPS/4;
        maxangl=minangl+STEPS/2;
        minhite= (MINUSY-transy)*wscale/VSTEP;
        maxhite= (PLUSY-transy)*wscale/VSTEP;
        pmin=NFRAME;
        pmax=0;
        linesteps=1.0;
        lineflg=1; /* 0=dots 1=lines */
        mflg=1;   /* selects mirror image */
        patchflg=0; /* selects sequence for drawit() */
        markflg=0;
        rotyflg=0;
        armflg=0;
        kbdflg=0;
        polyflg=0;
        lastheight=0;
        clearflg=0;
        splineflg=0;/* 0=rawdata 1=splinedata */
        zflg=0;
        cartdataflg=0;
        patchcount=0;
        dcnttemp=0;
        tapeflg=1; /* 0=tape 1=no tape */
        revflg=0; /* 0=normal fit 1=reverse fit */
        bspcnt=1;
        initarry();
        pclear();
        initcubewin();
        initbarwin();
        initslicewin();
        initvertwin();
        initdatawin();
        initkeywin();
        inittextwin();
        printdata(datablock);
    drawit();
```

```
drawit2();
 drawbar(minval,maxval);

qdevice(MIDDLEMOUSE);
qdevice(LEFTMOUSE);
qdevice(ESCKEY);
qdevice(KEYBD);
qdevice(RETKEY);
qdevice(GHOSTX);
qdevice(GHOSTY);
qdevice(MOUSEX);
qdevice(MOUSEY);
 qdevice(WINQUIT);

exitflg=1;
while (exitflg==1) {
    if(patchflg==1 || patchflg>=3){
  switch (qread(&val)) {
              case GHOSTX:
                   attachcursor(GHOSTX,GHOSTY);
                break;
              case GHOSTY:
                   attachcursor(GHOSTX,GHOSTY);
                break;
    case MIDDLEMOUSE:
          slice(1,0,tapeflg);
          qreset();
          break;
    case LEFTMOUSE:
          pickx();
    if(kbdflg==3){
                   command = rdkey3a();
          sginap(20);
                   command = rdkey3a();
        switch (command) {
            case 0:
                 break;
            case DAT1:
```

```
        minval= 0;
        maxval= obdatacnt*1.5;
        drawbar(minval,maxval);
        attachcursor(MOUSEX,MOUSEY);
        qreset();
        break;
    case DAT2:
        minval= 0;
        maxval= obdatacnt*1.5;
        drawbar(minval,maxval);
        attachcursor(MOUSEX,MOUSEY);
        qreset();
        break;
    case SETU:
        minval= 0;
        maxval= 3;
        drawbar(minval,maxval);
        attachcursor(MOUSEX,MOUSEY);
        qreset();
        break;
    case IRIS2:
        patchflg=0;
        mouseflg = 0;
        splineflg=0;
        polyflg=0;
        kbdflg=0;
        lineflg=1;
        drawit();
        drawit2();
        qreset();
        break;
    case TGL:
        dcount=0;
        sprintf(string,"  %d",dcount);
        winset(keywin);
        cmov2(0,9.75);
        charstr(string);
        winset(cubewin);
```

```
        qreset();
        break;
    case REDO:
        redo();
        patchflg=0;
        mouseflg = 0;
        splineflg=0;
        polyflg=0;
        kbdflg=0;
        lineflg=1;
        keyboard();
        drawit();
        qreset();
        break;
    case SAVE:
        slicesav();
        drawit();
        keybd3a();
        qreset();
        break;
    case TCIRC:
        datum1 =  360.1;
        measblock[6]=datum1;
        datum2 = 359.9;
        measblock[7]=datum2;
        for(i=0;i<dcount;i++){
        getslice(i);
        bezplot(0);
        lintape(datum1,datum2,0);
        slices[i][NPTS][1]=measblock[1];
        stslice(slices[i][NPTS][0],i);
        }
        slicesav();
        drawit();
        keybd3a();
        qreset();
        break;
    case RECALL:
```

```
        dcount=0;
        patchcount=0;
        slicelod();
        bslcptch(dcount);
        dcnttemp=dcount;
        drawit();
        keybd3a();
        qreset();
        break;
    case SAVCNT:
        dcnttemp=dcount;
        qreset();
        break;
    case GETCNT:
        dcount=dcnttemp;
        sprintf(string,"  %d",dcount);
        winset(keywin);
        cmov2(0,9.75);
        charstr(string);
        winset(cubewin);
        qreset();
        break;
    case ZFLG:/* changes slice file format */
        zflg=zflg+1;
        if(zflg>1)zflg=0;
        sprintf(string,"  %d",zflg);
        winset(keywin);
        cmov2(0,9.75);
        charstr(string);
        winset(cubewin);
        qreset();
        break;
    case DELTAX:
    case DELTAY:
        drawbar(-500.0, 500.0);
        attachcursor(MOUSEX,MOUSEY);
        patchflg=1;
        qreset();
```

```
        break;
    case PCLEAR:
        pclear();
        qreset();
        break;
    case SETZERO:
        setzero();
        qreset();
        break;
    case DPZERO:
        for(i=0;i<NPTS;i++){
        DPOLY[i][0]=0;
        DPOLY[i][1]=0;
        }
        drawit();
        drawit2();
        qreset();
        break;
    case XYZERO:
        deltax=0;
        deltay=0;
        roty=0;
        patchflg=1;
        drawit();
        qreset();
        break;
    case MIRROR:
        if(mflg==0)mflg=1;
        else mflg=0;
        hite=sliceht*SCALE/VSTEP;
        fitpoly(hite);
        patchflg=1;
        drawit();
        qreset();
        break;
    case ROTY:
        drawbar(-180.0, 180.0);
        attachcursor(MOUSEX,MOUSEY);
```

```
            patchflg=1;
            qreset();
            break;
        case GETSL:
            datum1 =  360.1;
            measblock[6]=datum1;
            datum2 = 359.9;
            measblock[7]=datum2;
            patchflg=4;
            drawit();
            qreset();
            break;
        case SPLINE:
            datum1 =  360.1;
            measblock[6]=datum1;
            datum2 = 359.9;
            measblock[7]=datum2;
            if(splineflg==0){
            patchflg=4;
            splineflg=1;
            }
            else {
            splineflg=0;
            patchflg=1;
            }
            qreset();
            break;
        case VOLUME:
            vol();
            patchflg=1;
            drawit();
            qreset();
            break;
        case OBLIQUE:
            patchflg=1;
            if(cartdataflg==0){
            fillcart(1);
            cartdataflg=1;
```

```
        }
        obltrig();
        kbdflg=3;
        drawit();
        keybd3a();
        qreset();
        break;
    case SMES:
        bezplot(1);
        patchflg=3;
        kbdflg=3;
        drawit();
        keybd3a();
        qreset();
        break;
   case PTCH:
        bslcptch(dcount);
        drawit();
        qreset();
        break;
   case SHOCNT:
        sprintf(string,"  %d",dcount);
        winset(keywin);
        cmov2(0,9.75);
        charstr(string);
        winset(cubewin);
        qreset();
        break;
  case CNTUP:
       dcount=dcount+1;
       sprintf(string,"  %d",dcount);
       winset(keywin);
       cmov2(0,9.75);
       charstr(string);
       winset(cubewin);
       qreset();
       break;
 case CNTDN:
```

```
            dcount=dcount-1;
            sprintf(string,"   %d",dcount);
            winset(keywin);
            cmov2(0,9.75);
            charstr(string);
            winset(cubewin);
            qreset();
            break;
        case LINES:
            lineflg=1;
            patchflg=0;
            clearflg=0;
            splineflg=0;
            kbdflg=0;
            keyboard();
            drawit();
            qreset();
            break;
        case MLOAD:
            masterlod();
            keybd3a();
            drawit();
            qreset();
            break;
        case REGEN:
            shapelod();
            keybd3a();
            shapeint();
            resltxt();
            drawit();
            qreset();
            break;
        case TAPEFLG:
            if(tapeflg>0)tapeflg=0;
            else tapeflg=1;
            drawit();
            qreset();
            break;
```

```
                default:
                break;
            } /* end of switch */
        } /*end if kbdflg = 3 */


else if(kbdflg==4){
        if(getvaluator(MOUSEX)>XMAXSCREEN/10)comm
and=pickx();
else {
command = rdkey3b();
sginap(20);
command = rdkey3b();
    }
      switch (command) {
          case FITP:
       hite=sliceht*SCALE/VSTEP;
                fitpoly(hite);
                patchflg=1;
                drawit();
                qreset();
                break;
           case UP:
/* sliceht = true height/wscale */
        temp=(float)VSTEP/SCALE;
        sliceht=sliceht + temp;
        if(sliceht>pmax)pmax=sliceht;
        if(sliceht<pmin)pmin=sliceht;
                patchflg=1;
                drawit();
                qreset();
                break;
           case UP5: /* UPDY !!! */
/* sliceht = true height/wscale */
        temp=linesteps/SCALE;
        sliceht=sliceht + temp;
        if(sliceht>pmax)pmax=sliceht;
        if(sliceht<pmin)pmin=sliceht;
```

```
            hite=sliceht*SCALE/VSTEP;
                    fitpoly(hite);
                    patchflg=1;
                    drawit();
                    qreset();
                    break;
                case DOWN:
        /* sliceht = true height/wscale */
            temp=(float)VSTEP/SCALE;
            sliceht=sliceht - temp;
            if(sliceht>pmax)pmax=sliceht;
            if(sliceht<pmin)pmin=sliceht;
                    patchflg=1;
                    drawit();
                    qreset();
                    break;
                case TGL:
                    dcount=0;
                    sprintf(string,"   %d",dcount);
                    winset(keywin);
                    cmov2(0,9.75);
                    charstr(string);
                    winset(cubewin);
                    qreset();
                    break;
                case INP:
                    lastheight=sliceht*wscale;
                    pmeas(measblock);
                    stslice(lastheight,dcount);
                    dcount++;
                    if(dcount>49)dcount=0;
                    sprintf(string,"   %d",dcount);
                    winset(keywin);
                    cmov2(0,9.75);
                    charstr(string);
                    winset(cubewin);
                    qreset();
                    break;
```

```
case CNTUP:
    dcount=dcount+1;
    sprintf(string,"  %d",dcount);
    winset(keywin);
    cmov2(0,9.75);
    charstr(string);
    winset(cubewin);
    qreset();
    break;
case CNTDN:
    dcount=dcount-1;
    sprintf(string,"  %d",dcount);
    winset(keywin);
    cmov2(0,9.75);
    charstr(string);
    winset(cubewin);
    qreset();
    break;
case SPANG:
    spangcnt=spangcnt+1;
    if(spangcnt>5)spangcnt=0;
    sprintf(string,"  %d",spangcnt);
    winset(keywin);
    cmov2(0,9.75);
    charstr(string);
    winset(cubewin);
    hite=sliceht*SCALE/VSTEP;
    fitpoly(hite);
    patchflg=1;
    drawit();
    qreset();
    break;
    case ELIPSE:
    elipse();
    patchflg=6;
    drawit();
    qreset();
    break;
```

```
case ROTY:
        drawbar(-180.0, 180.0);
        attachcursor(MOUSEX,MOUSEY);
        qreset();
        break;
case DELTAX:
case DELTAY:
        drawbar(-500.0, 500.0);
        attachcursor(MOUSEX,MOUSEY);
        patchflg=1;
        qreset();
        break;
case GETSL:
        patchflg=4;
        drawit();
        qreset();
        break;
case IRIS2:
        patchflg=0;
        mouseflg = 0;
        splineflg=0;
        polyflg=0;
        kbdflg=0;
        lineflg=1;
        drawit();
        drawit2();
        drawbar(minval,maxval);
        qreset();
        break;
case PTCH:
        bslcptch(dcount);
        drawit();
        qreset();
        break;
case KEYS1:
        kbdflg=3;
        qreset();
        break;
```

```
            case REV:
                if(revflg==0)revflg=1;
                else revflg=0;
                qreset();
                break;
            default:
                break;
                } /* end of switch */
                } /* end if kbdflg = 4 */
        } /* end of switch */
while (qtest() == 0)  {
if(kbdflg==3){
        switch (command) {
          case DAT1:
                readbar(&transparam);
                datum1 =  transparam;
                measblock[6]=datum1;
                pmeas(measblock);
                drawit();
                break;
           case DAT2:
                readbar(&transparam);
                datum2 = transparam;
                measblock[7]=datum2;
                pmeas(measblock);
                drawit();
                break;
          case DELTAX:
                readbar(&transparam);
                deltax=transparam;
                datablock[13]=deltax;
                printdata(datablock);
                drawit();
                break;
          case DELTAY:
                readbar(&transparam);
                deltay=transparam;
                datablock[14]=deltay;
```

```
            printdata(datablock);
            drawit();
            break;
        case SETU:
            datum1 =  360.1;
            measblock[6]=datum1;
            datum2 = 359.9;
            measblock[7]=datum2;
            readbar(&transparam);
            uvalue=transparam;
            measblock[8]=uvalue;
            crslice(uvalue,dcount);
            patchflg=3;
            bezplot(1);
            lintape(datum1,datum2,0);
            pmeas(measblock);
            kbdflg=3;
            drawit();
            keybd3a();
            break;
        case ROTY:
            readbar(&transparam);
            roty= transparam*10;
            datablock[1]= transparam;
            printdata(datablock);
            drawit();
            break;
        case KEYS2:
            kbdflg=4;
            break;
        default:
            break;
    } /* end of switch */
    } /*end if kbdflg = 3 */


    if(kbdflg==4){
    switch (command) {
```

```
case DPX0:
        spoint(0,1);
        break;
case DPY0:
        spoint(0,2);
        break;
case DP1:
        spoint(1,3);
        break;
case DP2:
        spoint(2,3);
        break;
case DP3:
        spoint(3,3);
        break;
case DP4:
        spoint(4,3);
        drawit();
        break;
case DP5:
        spoint(5,3);
        break;
case DP6:
        spoint(6,3);
        break;
case DP7:
        spoint(7,3);
        break;
case DP8:
        spoint(8,3);
        break;
case DP9:
        spoint(9,3);
        break;
case DP10:
        spoint(10,3);
        break;
case DP11:
```

```
        spoint(11,3);
        break;
    case DP12:
        spoint(12,3);
        break;
    case DP13:
        spoint(13,3);
        break;
    case DP14:
        spoint(14,3);
        break;
    case DPX15:
        spoint(15,1);
        break;
    case DPY15:
        spoint(15,2);
        break;
    case ROTY:
        readbar(&transparam);
        roty= transparam*10;
        datablock[1]= transparam;
        printdata(datablock);
        slice(1,0,tapeflg);
        drawit();
        break;
    case DELTAX:
        readbar(&transparam);
        deltax=transparam;
        datablock[13]=deltax;
        printdata(datablock);
        drawit();
        break;
    case DELTAY:
        readbar(&transparam);
        deltay=transparam;
        datablock[14]=deltay;
        printdata(datablock);
        drawit();
```

```
                    break;
                    default:
                    break;
            } /* end of switch */
            } /* end if kbdflg = 4 */
        } /* end of while qtest */
    } /* end of if patchflg */

    else {

switch (qread(&val)) {
            case GHOSTX:
                attachcursor(GHOSTX,GHOSTY);
                break;
            case GHOSTY:
                attachcursor(GHOSTX,GHOSTY);
                break;
            case MIDDLEMOUSE:
                attachcursor(MOUSEX,MOUSEY);
                slice(0,0,tapeflg);
                drawit();
                drawit2();
          break;
            case LEFTMOUSE:
                command = readkey();
                sginap(20);
                switch (command = readkey()) {
            case 0:
                continue;
             case TRANSY:
                drawbar(0.0, -2100.0);
                attachcursor(MOUSEX,MOUSEY);
                qreset();
                break;
             case TRANSX:
             case TRANSZ:
                drawbar(-500.0, 500.0);
                attachcursor(MOUSEX,MOUSEY);
```

```
        qreset();
        break;
    case ZOOM:
        drawbar(0.0, 4.0);
        attachcursor(MOUSEX,MOUSEY);
        qreset();
        break;
    case ROTX:
    case ROTY:
    case ROTZ:
        drawbar(-180.0, 180.0);
        attachcursor(MOUSEX,MOUSEY);
        qreset();
        break;
    case RESET:
        transx=0; transy= MINUSY; transz=0;
        rotx=0; roty=0; rotz=0;
        zoom=1;
        setzero();
        rotyflg=0;
        minangl=stangle/ANGLE-STEPS/4;
        maxangl=minangl+STEPS/2;
        patchflg=0;
        kbdflg=0;
        slice(0,0,tapeflg);
        keyboard();
        drawit2();
        drawit();
        drawbar(minval,maxval);
        attachcursor(MOUSEX,MOUSEY);
        qreset();
        break;
    case XFILE:
        initarry();
        loadfile();
        command = 99;
        drawit();
        drawit2();
```

```
        attachcursor(MOUSEX,MOUSEY);
        keyboard();
        drawit2();
        drawit();
        qreset();
        break;
    case EXITTRANS:
        attachcursor(MOUSEX,MOUSEY);
        greset();
        gexit();
        exit(0);
        break;
    case CURVE:
        patchflg=1;
        polyflg=0;
        keybd3a();
        kbdflg=3;
        pmax=sliceht;
        pmin=sliceht;
        drawit();
        qreset();
        break;
    case LINES:
        lineflg=1;
        patchflg=0;
        clearflg=0;
        splineflg=0;
        kbdflg=0;
        keyboard();
        drawit();
        qreset();
        break;
    case X1SET:
    case X2SET:
        rotyflg=1;
        minval= (stangle*10-roty)/10/ANGLE;
        maxval= minval+STEPS;
        drawbar(minval,maxval);
```

```
            qreset();
            break;
        case FB:
            fbflg++;
            if(fbflg>1)fbflg=0;
            stangle=fbflg*180;
            minangl=minangl+stangle/ANGLE;
            maxangl=maxangl+stangle/ANGLE;
            patchflg=0;
            drawit();
            drawit2();
            qreset();
            break;
        case Y1SET:
        case Y2SET:;
            minval=0;
            maxval=NFRAME*VSTEP;
            drawbar(minval,maxval);
            qreset();
            break;
        case ARMS:
            armflg=armflg+1;
            if(armflg >3)armflg=0;
            slice(0,0,tapeflg);
            drawit();
            drawit2();
            qreset();
            break;
        case LINESTEP:
            nslice();
            linesteps = (set2-set1)/(linesteps);
            drawit();
            keyboard();
            qreset();
            break;
        case SETZERO:
            setzero();
            qreset();
```

```
        break;
    case MARKS:
        wandmark(); /* get markers */
        markflg = markflg+1;
        if(markflg > 3)markflg=0;
        drawit();
        drawit2();
        qreset();
        break;
    case LPP:
        patchflg=2;
        drawit();
        qreset();
        break;
    case LPSAVE:
        lpatchsav();
        drawit();
        keyboard();
        qreset();
        break;
    case SSAVE:
        splinesav();
        drawit();
        keyboard();
        qreset();
        break;
    case LPLOAD:
        patchcount=0;
        lpatchlod();
        keyboard();
        drawit();
        drawit2();
        qreset();
        break;
    case LCLEAR:
        patchcount=0;
        qreset();
        break;
```

```
                case SKIN:
                    patchflg=4;
                    initskin();
                    nplot();
                    winset(barwin);
                    winpop();
                    winset(keywin);
                    winpop();
                    kbdflg=0;
                    keyboard();
                    rotx=0;
                    roty=0;
                    rotz=0;
                    cartdataflg=0;
                    printdata(datablock);
                    drawit();
                    drawit2();
                    qreset();
                    break;
                case SET1:
                    set1=sliceht*wscale;
                    cartdataflg=0;
                    pmeas(measblock);
                    qreset();
                    break;
                case SET2:
                    set2=sliceht*wscale;
                    cartdataflg=0;
                    pmeas(measblock);
                    qreset();
                    break;
                default:
                    break;
            } /* end of switch command = readkey */
            } /* end of switch qread &val */
    while (qtest() == 0)  /* while no buttons pressed */ {
      if (readbar(&transparam)) {
          switch (command) {
```

```
        case TRANSX:
            transx=transparam/zoom/SCALE;
            datablock[3]= transparam*zoom;
            printdata(datablock);
            drawit();
            drawit2();
            break;
        case TRANSY:
            transy=transparam/zoom/SCALE;
            datablock[4]= transparam*zoom;
            printdata(datablock);
            minhite=(MINUSY-transy)*wscale
                    /VSTEP-zoomtemp;
            maxhite= (PLUSY-transy)*wscale
                    /VSTEP+zoomtemp;
            if(minhite<0)minhite=0;
            if(minhite>NFRAME)minhite=NFRAME;
            if(maxhite<0)maxhite=0;
    if(maxhite>NFRAME)maxhite=NFRAME;
            drawit();
            drawit2();
            break;
        case TRANSZ:
            transz=transparam/zoom/SCALE;
            datablock[5]= transparam*zoom;
            printdata(datablock);
            drawit();
            drawit2();
            break;
        case ZOOM:
            zoom=transparam;
            if(zoom<0.2)zoom=0.2;
            if(zoom>10)zoom=10;
            datablock[12]=zoom;
            printdata(datablock);
            zoomtemp= 15/zoom*wscale/VSTEP;
            minhite= (MINUSY-transy)*wscale
                    /VSTEP-zoomtemp;
```

```
            maxhite= (PLUSY-transy)*wscale
                    /VSTEP+zoomtemp;
            if(minhite<0)minhite=0;
            if(minhite>NFRAME)minhite=NFRAME;
            if(maxhite<0)maxhite=0;
        if(maxhite>NFRAME)maxhite=NFRAME;
            drawit();
            drawit2();
            break;
        case ROTX:
            rotx=transparam*10;
            datablock[0]= transparam;
            printdata(datablock);
            drawit();
            drawit2();
            break;
        case ROTY:
            roty= transparam*10;
            datablock[1]= transparam;
            printdata(datablock);
            if(rotyflg==0){
            minangl= (stangle*10-roty)/10
                    /ANGLE-STEPS/4;
            maxangl=minangl+STEPS/2;
            }
            drawit();
            drawit2();
            break;
        case ROTZ:
            rotz=transparam*10;
            datablock[2]= transparam;
            printdata(datablock);
            drawit();
            drawit2();
            break;
        case X1SET:
            minangl=stangle/ANGLE-STEPS/4
                    +transparam*10;
```

```
                    drawit();
                    drawit2();
                    break;
                case X2SET:
                    maxangl=transparam*10;
                    drawit();
                    drawit2();
                    break;
                case Y1SET:
                    minhite=transparam/VSTEP;
                    drawit();
                    drawit2();
                    break;
                case Y2SET:
                    maxhite=transparam/VSTEP;
                    drawit();
                    drawit2();
                    break;
                default:
                    break;
            } /* end of switch */
        } /* end of if */
    } /* end of while */
  } /* end of switch */
 } /* end of else */
} /* end of main */

drawit()
{
    int i;
    float rotangle;


    winset(cubewin);

    pushmatrix();

    if(patchflg == 0)rotax(1);
```

```
else if(patchflg == 2)rotax(1);
else if(patchflg == 3)rotax(1);
else translate(transx,transy,transz);

if(clearflg==0){
c3s(whitevec);
clear();
}
c3s(bluevec);

switch(patchflg){
case 0:

        ldrw();
        break;
case 1:
        translate(transx,-transy,transz);
        slicedrw(sliceht);
        dpolyplot(NPTS);
        bezplot(0);
        if(tapeflg==0)lintape(datum1,datum2,1);
        drwangref();
        break;
case 2:
        ldrw();
        for(i=0;i<patchcount;i=i+1)ptchplt(i);
        break;
case 3:
        plotobdata(datum1,datum2);
        break;
case 4:
        translate(transx,-transy,transz);
        getslice(dcount);
        sliceht=slices[dcount][NPTS][0]/SCALE;
        slicedrw(sliceht);
        dpolyplot(NPTS);
        bezplot(0);
        if(tapeflg==0)lintape(datum1,datum2,1);
```

```
        drwangref();
        pmeas(measblock);
        break;
case 5:
        slice(1,0,tapeflg);
        break;
case 6:
        translate(transx,-transy,transz);
        sliceht=slices[dcount][NPTS][0]/SCALE;
        slicedrw(sliceht);
        dpolyplot(NPTS);
        bezplot(0);
        if(tapeflg==0)lintape(datum1,datum2,1);
        drwangref();
        pmeas(measblock);
        break;

default:
        break;
}
heightref(sliceht);
if(markflg !=0)wandplot(0);
popmatrix();
drawaxes();
swapbuffers();
winset(slicewin);
pushmatrix();
c3s(whitevec);
clear();
c3s(bluevec);
rotangle=(float)roty*PI/1800;
translate(transx*cos(rotangle)+transz*sin(rotangle),-
transx*sin(rotangle)+transz*cos(rotangle),0);
slicedrw(sliceht);
popmatrix();
sliceref();
swapbuffers();
winset(cubewin);
```

```
}

drawit2()
{
        if(patchflg == 0){
        winset(vertwin);
        pushmatrix();
        scale(zoom,zoom,zoom);
        translate(0,transy,0);
        vertdrw();
        c3s(redvec);
        move2(-5,sliceht);
        draw2(5,sliceht);
        popmatrix();
        swapbuffers();
        }
}
/* here to recall data */

recall()
{
        int i,hite;

                hite=sliceht*SCALE/VSTEP;
                for(i=0;i<NPTS;i++){
                DPOLY[i][0]=bslice[hite][i][0];
                DPOLY[i][1]=bslice[hite][i][1];
                }
}

initcubewin()
{
   short *nofly;

   foreground();
   attachcursor(MOUSEX,MOUSEY);
   prefposition(0, XMAXSCREEN, 0, YMAXSCREEN-30);
```

```
    cubewin = winopen("Lass");
        RGBmode();
        drawmode(CURSORDRAW);
        mapcolor(1,0,0,0);
        drawmode(NORMALDRAW);
        curstype(CCROSS);
        defcursor(1,nofly);
        setcursor(1);
        ortho(MINUSX,PLUSX,MINUSY,PLUSY,
                    MINUSZ,PLUSZ);
        doublebuffer();
        gconfig();
        c3s(whitevec);
        clear();
        c3s(bluevec);
        swapbuffers();
}

initbarwin()
{
  prefposition(10, 1270, 10, 100);
  noborder();
    barwin = winopen("trans");
        RGBmode();
        doublebuffer();
        gconfig();
        wintitle("");
        drawbar(-10.0, 10.0);
        c3s(whitevec);
        clear();
        c3s(bluevec);
        swapbuffers();
}

initvertwin()
{
    prefposition(XMAXSCREEN*3/4, XMAXSCREEN,
        120,(YMAXSCREEN-70)*3/4);
```

```
    vertwin = winopen("vert. slice");
        RGBmode();
        shademodel(FLAT);
        ortho2(-5,5,-7.5,7.5);
        doublebuffer();
        gconfig();
        c3s(whitevec);
        clear();
        c3s(bluevec);
        c3s(blackvec);
}

initslicewin()
{
   prefposition(XMAXSCREEN*3/4, XMAXSCREEN,
        (YMAXSCREEN-30)*3/4, YMAXSCREEN-30);
   slicewin = winopen("hor. slice");
        ortho2(-5,5,-3.75,3.75);
/*      ortho(MINUSX,PLUSX,MINUSY,PLUSY,
                    MINUSZ,PLUSZ);*/
        RGBmode();
        doublebuffer();
        gconfig();
        c3s(whitevec);
        clear();
        c3s(bluevec);
}

initdatawin()
{
   prefposition(0, XMAXSCREEN/3, (YMAXSCREEN-30)
                        *3/4,YMAXSCREEN-30);
   datawin = winopen("data");
        RGBmode();
        shademodel(FLAT);
        ortho2(0,15,0,10);
        gconfig();
        c3s(whitevec);
```

```
        clear();
        c3s(blackvec);
}

initkeywin()
{
   prefposition(5, XMAXSCREEN/10, 0,(YMAXSCREEN-
            70)*3/4);
   keywin = winopen("controls");
        RGBmode();
        shademodel(FLAT);
        ortho2(0,2,0,10);
        gconfig();
        c3s(whitevec);
        clear();
        c3s(blackvec);
        keyboard();
        winset(vertwin);
        c3s(whitevec);
        clear();
        c3s(bluevec);
        vertdrw();
}

inittextwin()
{
   prefposition(0, XMAXSCREEN/3, YMAXSCREEN
                        /10,2*YMAXSCREEN/10);
   textwin = winopen("text");
        RGBmode();
        shademodel(FLAT);
        ortho2(0,15,0,10);
        foreground();
        gconfig();
        winpush();
}
/* here to get line into s, return length */
```

```
int getline(s,cnt,lim)
        char s[];
        int cnt,lim;
{
        Device val;
        long dev;
        int c,flg,i;

        flg=1; i=cnt;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
        case KEYBD:
        if(val=='\b'){
        i--;
        s[i]= '\0';
        c3s(whitevec);
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        s[i++]=val;
        }
        if(val=='\n')flg=0;
        putline(s,1.0,5.0);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
```

```
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
  return(i);
}
/* here to write a line to a window */

putline(s,x,y)
        char s[];
        Coord x,y;
{
        cmov2(x,y);
        charstr(s);
}
/* here to move spline points */
spoint(nn,oo)
        int nn,oo;
{
        float xcen,ycen,xmult,ymult;

        xcen=XMAXSCREEN/2;
        ycen=(YMAXSCREEN-30)/2;

        xmult=xcen/(PLUSX*SCALE*1.64);
        ymult=ycen/(PLUSY*SCALE*1.74);

        if(oo==1)DPOLY[nn][0]=(getvaluator(MOUSEX)-
                                xcen)*xmult;
        if(oo==2)DPOLY[nn][1]=(getvaluator(MOUSEY)-
                                ycen)*ymult;
        if(oo==3){
        DPOLY[nn][0]=(getvaluator(MOUSEX)-xcen)*xmult;
        DPOLY[nn][1]=(getvaluator(MOUSEY)-ycen)*ymult;
                }
        patchflg=1;
        drawit();
}
```

```
/* here to find spline points */

fpoint(nn)
        int nn;
{
        int oo;
        float xcen,ycen,xmult,ymult,tol;

        tol=10;
        xcen=XMAXSCREEN/2;
        ycen=(YMAXSCREEN-30)/2;

        xmult=xcen/(PLUSX*SCALE*1.64);
        ymult=ycen/(PLUSY*SCALE*1.74);

        if((getvaluator(MOUSEX)-        xcen)*xmult>DPOLY[nn][0]-tol
&&
        (getvaluator(MOUSEX)-  xcen)*xmult<DPOLY[nn][0]+tol &&
        (getvaluator(MOUSEY)-ycen)*ymult>DPOLY[nn][1]-
        tol && (getvaluator(MOUSEY)-
        ycen)*ymult<DPOLY[nn][1]+tol)
        oo=1;
        else oo=0;
        return(oo);
}
```

/* lass2.c */

```
#include "device.h"
#include "lassdefs"
#include <gl.h>
#include <math.h>


        extern short NDATA[NARRAY];
        extern int rotx,roty,rotz;
        extern int steps,linesp;
        extern float zoom,pmax,pmin;
```

```
        extern float sliceht,linesteps,lastheight;
        extern long barwin,cubewin,slicewin;
        extern float vertwin,datawin,textwin;
        extern float transx,transy,transz;
        extern int patchflg,patchcount,dcount;
        extern int minangl,maxangl;
        float barmin, barmax, bardelta;
        extern short blackvec[3];
        extern short redvec[3];
        extern short greenvec[3];
        extern short bluevec[3];
        extern short whitevec[3];
        extern float datablock[20];
        extern float HEADER[HEADSIZE];
        extern float WAND[20][3];
        extern float DPOLY[NPTS+1][2];
        extern float set1,set2,measblock[NPTS];
        extern float bslice[NFRAME][NPTS][2];
        extern char string[20];


drawbar(minval, maxval)
float minval, maxval;
{
  register i;
  int xmouse;
  char str[20];

  winset(barwin);
  c3s(whitevec);
  clear();
  c3s(blackvec);
  barmin = minval;
  barmax = maxval;
  bardelta=(maxval-minval)/1000;
  cursoff();
  recti(150, 20, 1150, 40);
  xmouse = getvaluator(MOUSEX);
  for (i = 0; i < 5; i++) {
```

```
        move2i(150 + i*250, 40);
        draw2i(150 + i*250, 50);
        cmov2i(153 + i*250, 44);
        sprintf(str, "%6.2f", minval + i*(maxval - minval)/4.0);
        charstr(str);
    }
    c3s(redvec);
     if(xmouse>160 && xmouse<1160)rectfi(xmouse-
                        5,20,xmouse-15,40);
    curson();
    swapbuffers();
}

/* The readbar routine returns 1 if the value stored in retval
        is valid,and zero otherwise.*/


readbar(rtval)
float *rtval;
{
    int xmouse, ymouse;

    ymouse = getvaluator(MOUSEY);
    if (10 <= ymouse && ymouse <= 80) {
        xmouse = getvaluator(MOUSEX);
        if (100 <= xmouse && xmouse <= 1200) {
            *rtval = barmin + bardelta * (xmouse-160);
            drawbar(barmin,barmax);
            return 1;
        }
    }
    return 0;
}


drawref()
{
        c3s(blackvec);
        pushmatrix();
        move(-0.2, 0, 0);
```

```
        draw(0.2, 0, 0);
        move(0, -0.2, 0);
        draw(0, 0.2, 0);
        move(0, 0, -0.2);
        draw(0, 0, 0.2);
        popmatrix();
}

drawaxes()
{
        c3s(redvec);
        pushmatrix();
        translate(-6,0,0);
        rotate((int)rotx, 'x');
        rotate((int)roty, 'y');
        rotate((int)rotz, 'z');
        movei(0, 0, 0);
        drawi(0, 0, 1);
        movei(0, 1, 0);
        drawi(0, 0, 0);
        drawi(1, 0, 0);
        cmovi(0, 0, 1);
        charstr("z");
        cmovi(0, 1, 0);
        charstr("y");
        cmovi(1, 0, 0);
        charstr("x");
        popmatrix();
}

printdata(block)
        float block[];
{
        char num[50];

        winset(datawin);
        c3s(whitevec);
        clear();
```

```
        c3s(blackvec);
        cmov2(1,9);
        sprintf(num,"Zero              Present");
        charstr(num);
        cmov2(1,8.8);
        sprintf(num,"____              _______");
        charstr(num);
        cmov2(6,8);
        sprintf(num," rotx=%7.0f  dif=%6.0f",block[0],block[6]-
                          block[0]);
        charstr(num);
        cmov2(6,7.5);
        sprintf(num," roty=%7.0f  dif=%6.0f",block[1],block[7]-
                    block[1]);
        charstr(num);
        cmov2(6,7);
        sprintf(num," rotz=%7.0f  dif=%6.0f",block[2],block[8]-
                    block[2]);
        charstr(num);
        cmov2(6,6);
        sprintf(num,"transx=%7.1f         dif=%6.0f",block[3],block[9]-
block[3]);
        charstr(num);
        cmov2(6,5.5);
        sprintf(num,"transy=%7.1f         dif=%6.0f",block[4],block[10]-
block[4]);
        charstr(num);
        cmov2(6,5);
        sprintf(num,"transz=%7.1f         dif=%6.0f",block[5],block[11]-
block[5]);
        charstr(num);
        cmov2(1,8);
        sprintf(num,"rotx  =%5.0f",block[6]);
        charstr(num);
        cmov2(1,7.5);
        sprintf(num,"roty  =%5.0f",block[7]);
        charstr(num);
        cmov2(1,7);
```

```
        sprintf(num,"rotz  =%5.0f",block[8]);
        charstr(num);
        cmov2(1,6);
        sprintf(num,"     x=%5.1f",block[9]);
        charstr(num);
        cmov2(1,5.5);
        sprintf(num,"     y=%5.1f",block[10]);
        charstr(num);
        cmov2(1,5);
        sprintf(num,"     z=%5.1f",block[11]);
        charstr(num);
        cmov2(1,4);
        sprintf(num,"Deltax=%5.1f
        dif=%6.0f",block[13],block[13]-block[15]);
        charstr(num);
        cmov2(1,3.5);
        sprintf(num,"Deltay=%5.1f
        dif=%6.0f",block[14],block[14]-block[16]);
        charstr(num);
        cmov2(1,2.5);
        sprintf(num,"Zoom  =%5.1f",block[12]);
        charstr(num);
        winset(cubewin);
}
/* here to print measurement data */

pmeas(block)
        float block[];
{
        char num[50];
        float lincnt,linsp;

        lincnt=7.0;
        linsp=0.475;
        winset(vertwin);
        pushmatrix();
        c3s(whitevec);
        clear();
```

```
c3s(blackvec);
cmov2(-5.0,lincnt);
sprintf(num," P( 0,0) = %4.2f",DPOLY[0][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P( 0,1) = %4.2f",DPOLY[0][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P( 1,0) = %4.2f",DPOLY[1][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P( 1,1) = %4.2f",DPOLY[1][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P( 2,0) = %4.2f",DPOLY[2][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P( 2,1) = %4.2f",DPOLY[2][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P( 3,0) = %4.2f",DPOLY[3][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P( 3,1) = %4.2f",DPOLY[3][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P( 4,0) = %4.2f",DPOLY[4][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P( 4,1) = %4.2f",DPOLY[4][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P( 5,0) = %4.2f",DPOLY[5][0]);
```

```
charstr(num);
cmov2(0,lincnt);
sprintf(num," P( 5,1) = %4.2f",DPOLY[5][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P( 6,0) = %4.2f",DPOLY[6][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P( 6,1) = %4.2f",DPOLY[6][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P( 7,0) = %4.2f",DPOLY[7][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P( 7,1) = %4.2f",DPOLY[7][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P( 8,0) = %4.2f",DPOLY[8][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P( 8,1) = %4.2f",DPOLY[8][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P( 9,0) = %4.2f",DPOLY[9][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P( 9,1) = %4.2f",DPOLY[9][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P(10,0) = %4.2f",DPOLY[10][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P(10,1) = %4.2f",DPOLY[10][1]);
```

```
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P(11,0) = %4.2f",DPOLY[11][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P(11,1) = %4.2f",DPOLY[11][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P(12,0) = %4.2f",DPOLY[12][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P(12,1) = %4.2f",DPOLY[12][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P(13,0) = %4.2f",DPOLY[13][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P(13,1) = %4.2f",DPOLY[13][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P(14,0) = %4.2f",DPOLY[14][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P(14,1) = %4.2f",DPOLY[14][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," P(15,0) = %4.2f",DPOLY[15][0]);
charstr(num);
cmov2(0,lincnt);
sprintf(num," P(15,1) = %4.2f",DPOLY[15][1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
```

```
sprintf(num," Set 1  = %4.2f mm.",set1);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," Set 2  = %4.2f mm.",set2);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," Volume = %4.2f cc.",block[4]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," Area   = %4.1f sq.cm.",block[5]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," Height = %4.2f            mm.",DPOLY[NPTS][0]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," distance black - red = %4.2f",block[0]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," tape  distance (mm)  = %4.2f",block[1]);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," tape  distance (in)  =
%4.2f",block[1]/25.4);
charstr(num);
lincnt=lincnt-linsp;
cmov2(-5.0,lincnt);
sprintf(num," arc  distance (mm)   = %4.1f",block[2]);
charstr(num);
lincnt=lincnt-linsp,
cmov2(-5.0,lincnt);
sprintf(num," arc  area  (sq.cm)   = %5.1f",block[3]);
```

```
        charstr(num);
        lincnt=lincnt-linsp;
        cmov2(-5.0,lincnt);
        sprintf(num," last height        = %5.1f",lastheight);
        charstr(num);
        lincnt=lincnt-linsp;
        cmov2(-5.0,lincnt);
        sprintf(num," last count         = %4d",dcount);
        charstr(num);
        lincnt=lincnt-linsp;
        cmov2(-5.0,lincnt);
        sprintf(num," datum1   = %5.1f",block[6]);
        charstr(num);
        lincnt=lincnt-linsp;
        cmov2(-5.0,lincnt);
        sprintf(num," datum2   = %5.1f",block[7]);
        charstr(num);
        lincnt=lincnt-linsp;
        cmov2(-5.0,lincnt);
        sprintf(num," uvalue   = %5.2f",block[8]);
        charstr(num);
        popmatrix();
        swapbuffers();
        winset(cubewin);
}
/* here to calculate wand markers */

wandmark()
        {
        int i,z,angl,hite,hitetemp,error,errormin;
        float x,y,xslope,yslope,angle,angltemp;
        float nangle,radtemp,wandrad,truhite;
        short rads,n;


        for(i=0;i<HEADER[19];i++){
        x=HEADER[i*6+21];
        y=HEADER[i*6+22];
```

```
        xslope=IIEADER[i*6+23];
        yslope=HEADER[i*6+24];
        angle=IIEADER[i*6+25];
        errormin=1000;

         for(z=0;z<600;z++){
         truhite=y+z*yslope;
         hite=truhite/VSTEP;
         nangle=270-(180*fatan((x-z*xslope)/z)/PI+angle);
         if(nangle>360)nangle=nangle-360;
         angl=nangle/ANGLE;
         if(angl<0)angl=angl+STEPS;
         if(angl>STEPS)angl=angl-STEPS;
         rads=NDATA[NPOINT];
         n=NDATA[NPOINT+1];
          if(n>0){
          rads=rads/n;
           wandrad=x-z*xslope;
           wandrad=fhypot(wandrad,z);
           error=wandrad-rads;
            if(error<0)error=error*-1;
              if(error<errormin){
              errormin=error;
              angltemp=nangle;
               hitetemp=truhite;
              radtemp=wandrad;
              }
            }
          }
        WAND[i][0]=radtemp;
        WAND[i][1]=angltemp;
        WAND[i][2]=hitetemp;
        }
    }
/* here to plot wand markers */

wandplot(phi)
        int phi;
```

```
        {
        int i,hite,angl,maxmin;
        float rads,angle,xx,yy,zz,wscale;

        wscale=SCALE;
        maxmin=STEPS + minangl;
        for(i=0;i<HEADER[19];i++){
         rads=WAND[i][0];
         angle=WAND[i][1];
         hite=WAND[i][2];
          angle=angle-phi*ANGLE;
         if(angle>360)angle=angle-360;
          if(angle<0)angle=angle+360;
          angl=STEPS - angle/ANGLE;
          if(angl > maxmin  || angl < maxangl ||
                        patchflg > 4){
          xx= -rads*sin(angle*RAD)/wscale;
          yy=hite/wscale;
          zz=rads*cos(angle*RAD)/wscale;
          bigcube(xx,yy,zz);
          }
    }
 }

bigcube(xx,yy,zz)
        float xx,yy,zz;
{
        float vert[3],inc;

        c3s(blackvec);


        inc = 0.05;
        bgnpolygon();
        vert[0]=xx-inc;
        vert[1]=yy+inc;
        vert[2]=zz;
        v3f(vert);
```

```
vert[0]=xx+inc;
vert[1]=yy+inc;
vert[2]=zz;
v3f(vert);
vert[0]=xx+inc;
vert[1]=yy-inc;
vert[2]=zz;
v3f(vert);
vert[0]=xx-inc;
vert[1]=yy-inc;
vert[2]=zz;
v3f(vert);
endpolygon();
bgnpolygon();
vert[0]=xx-inc;
vert[1]=yy;
vert[2]=zz+inc;
v3f(vert);
vert[0]=xx+inc;
vert[1]=yy;
vert[2]=zz+inc;
v3f(vert);
vert[0]=xx+inc;
vert[1]=yy;
vert[2]=zz-inc;
v3f(vert);
vert[0]=xx-inc;
vert[1]=yy;
vert[2]=zz-inc;
v3f(vert);
endpolygon();
bgnpolygon();
vert[0]=xx;
vert[1]=yy-inc;
vert[2]=zz+inc;
v3f(vert);
vert[0]=xx;
vert[1]=yy+inc;
```

```
        vert[2]=zz+inc;
        v3f(vert);
        vert[0]=xx;
        vert[1]=yy+inc;
        vert[2]=zz-inc;
        v3f(vert);
        vert[0]=xx;
        vert[1]=yy-inc;
        vert[2]=zz-inc;
        v3f(vert);
        endpolygon();
}
/* here to clear bslice[][][] */

pclear()
{
        int i,j;

        for(i=0;i<NFRAME;i++){
          for(j=0;j<NPTS;j++){
         bslice[i][j][0]=0;
         bslice[i][j][1]=0;
         }
        }
        clrpatches();
        patchcount = 0;
}
/* here to enter from keyboard the number of slices between
                                                    limmits */

nslice()
{
        char *string1,string2[50],spacing[50];
        int i;

        string1 = "Enter number of slices ";
        winset(textwin);
        winpop();
```

```
    c3s(whitevec);
    clear();
    c3s(blackvec);
    for(i=0;i<50;i++){
    string2[i]='\0';
    }
    i=0;
    while(string1[i] !='\0'){
    string2[i]=string1[i];
    i++;
    }
  putline(string2,1.0,5.0);
  getline(string2,i,50);
  sscanf(&string2[i],"%s",spacing);
  linesteps=atof(spacing);
  winpush();
}
```

/* lass3.c */

```
#include "device.h"
#include "lassdefs"
#include <gl.h>
#include <math.h>
#define MAXINFILE 20
#define MAXSHAPEFILE 100
#define UMAX 3.0 /* maximum u value allowed */
#define DEF 10 /* binary digit resolution */


    extern short NDATA[NARRAY];
    extern float HEADER[HEADSIZE];
    extern float datablock[20];
    extern cardinal[4][4];
    extern float transx, transy, transz,deltax,deltay;
    extern long barwin, cubewin, slicewin;
    extern long vertwin,datawin, textwin;
    extern int rotx,roty,rotz;
```

```
extern int lineflg,armflg,splineflg,mflg,zflg,patchflg;
extern int steps,linesp,dcount,scount;
extern float zoom,sliceht,pmax,pmin;
extern float set1,set2,datum1,datum2;
extern float bslice[NFRAME][NPTS][2];
extern char *name;
extern int maxhite,minhite,maxangl,minangl;
extern int cartanglmax,stangle;
extern short blackvec[3];
extern short redvec[3];
extern short greenvec[3];
extern short bluevec[3];
extern short whitevec[3];
extern float xxx,yyy,zzz;
extern Coord cartesian[NARRAY][3];
extern float  MINIHEAD[20];
extern float patches[PATCHMAX][3][4][4];
extern float slices[MAXSLICES][NPTS+1][3];
extern float interpsl[4*MAXSLICES][NPTS+1][3];
extern float spangles[6][NPTS];
extern float apoly[3];
extern float cart[NFRAME*VSTEP][STEPS+10][3];
extern float DPOLY[NPTS+1][2];
extern int patchcount,spangcnt,cartrowcnt;
extern          float measblock[NPTS];
long vert1[] = { 0, 5};
long vert2[] = { 0,-5};
long vert3[] = { 5, 0};
long vert4[] = {-5, 0};
float infile[MAXINFILE][3];
float shapefile[MAXSHAPEFILE][2];
int  incount,shapecnt,mcount;



/* here to load data file */
```

```
loadfile()
   {
  FILE *fopen(),*file_ptr;
  char *string1,string2[50],*string3,string4[50];
  int cc,i,count;
  short *cyl;
  float *hed;


  string1 = "Enter file name ";
  string3 = "r";
       for(i=0;i<50;i++){
       string2[i]='\0';
       string4[i]='\0';
       }
       i=0;
       while(string1[i] !='\0'){
       string2[i]=string1[i];
       i++;
       }
  name = &string4[0];
       winset(textwin);
       winpop();
       c3s(whitevec);
       clear();
       c3s(blackvec);
  putline(string2,1.0,5.0);
  count = getline(string2,i,50);
  sscanf(&string2[i],"%s",name);
  winpush();
  initarry();
  file_ptr=fopen(name,string3);
  hed=HEADER;          /* point to data array. */
    fread(hed,4,HEADSIZE,file_ptr);
   cc=HEADER[0];
   linesp=HEADER[1];
    steps=linesp*cc/FRAME;
  cyl=NDATA;          /* point to data array. */
```

```
 fread(cyl,2,NARRAY,file_ptr);
   fclose(file_ptr);
  }
/* here to save large patch data  */


lpatchsav()
 {
 FILE *fopen(),*file_ptr;
  char *string1,string2[50],*string3,string4[50];
  int bytecount,count,i;

  string1 = "Enter file name  ";
  string3 = "w";

       for(i=0;i<50;i++){
       string2[i]='\0';
       string4[i]='\0';
       }
       i=0;
       while(string1[i] !='\0'){
       string2[i]=string1[i];
       i++;
       }
       string2[i]='\0';
       name = string4;
       winset(textwin);
       winpop();
       c3s(whitevec);
       clear();
       c3s(blackvec);
  putline(string2,1.0,5.0);
  count = getline(string2,i,50);
  sscanf(&string2[i],"%s",name);
  winpush();

  file_ptr=fopen(name,string3);
  printf("%d\n",file_ptr);
 MINIHEAD[0] = patchcount;
```

```
 fwrite(MINIHEAD,sizeof(*MINIHEAD),20,file_ptr);
 bytecount = fwrite(patches,sizeof(*patches),
                                PATCHMAX,file_ptr);
       fclose(file_ptr);
}
/* here to load large patch data */


lpatchlod()
{
  FILE *fopen(),*file_ptr;
   char *string1,string2[50],*string3,string4[50];
   int bytecount,count,i;

   clrpatches();

   string1 = "Enter file name ";
   string3 = "r";

        for(i=0;i<50;i++){
        string2[i]='\0';
        string4[i]='\0';
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }

   name = &string4[0];
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
    putline(string2,1.0,5.0);
    count = getline(string2,i,50);
    sscanf(&string2[i],"%s",name);
    winpush();
```

```
 file_ptr=fopen(name,string3);

 fread(MINIHEAD,sizeof(*MINIHEAD),20,file_ptr);
 patchcount =(int) MINIHEAD[0];
 bytecount = fread(patches,sizeof(*patches),
                              PATCHMAX,file_ptr);
      fclose(file_ptr);
      set1=patches[0][1][0][0];
      set2=patches[patchcount-1][1][0][0];
}
/* here to save  patches slice data  */


slicesav()
 {
 FILE *fopen(),*file_ptr;
 char *string1,string2[50],*string3,string4[50];
  int count,i,j;

  string1 = "Enter file name ";
  string3 = "w";

      for(i=0;i<50;i++){
      string2[i]='\0';
      string4[i]='\0';
      }
      i=0;
      while(string1[i] !='\0'){
      string2[i]=string1[i];
      i++;
      }

  name = &string4[0];
      winset(textwin);
      winpop();
      c3s(whitevec);
      clear();
      c3s(blackvec);
   putline(string2,1.0,5.0);
```

```
 count = getline(string2,i,50);
 sscanf(&string2[i],"%s",name);
 winpush();

 file_ptr=fopen(name,string3);
    fprintf(file_ptr,"%d",dcount);
    fprintf(file_ptr," %d\n",spangcnt);
      for(i=0;i<dcount;i++){
        for(j=0;j<(NPTS+1-zflg);j++){
        if(zflg==0)fprintf(file_ptr,"%8.1f%8.1f",
                          slices[i][j][0],slices[i][j][1]);
        else fprintf(file_ptr,"%8.1f%8.1f%8.1f",
            slices[i][j][0],slices[i][j][1],slices[i][j][2]);
       }
      fprintf(file_ptr,"\n");
       }
      fclose(file_ptr);
}
/* here to load  patches slice data (36 line text file)*/

slicelod()
{
  FILE *fopen(),*file_ptr;
  char *string1,string2[50],*string3,string4[50],buffer[20];
   int count,i,j,k,m;
  float xx,yy,temp;

   string1 = "Enter file name ";
  string3 = "r";

      for(i=0;i<50;i++){
      string2[i]='\0';
      string4[i]='\0';
      }
      i=0;
      while(string1[i] !='\0'){
      string2[i]=string1[i];
      i++;
```

```
    }

name = &string4[0];
    winset(textwin);
    winpop();
    c3s(whitevec);
    clear();
    c3s(blackvec);
putline(string2,1.0,5.0);
count = getline(string2,i,50);
sscanf(&string2[i],"%s",name);
winpush();
file_ptr=fopen(name,string3);
    fscanf(file_ptr,"%d",&dcount);
    fscanf(file_ptr,"%d",&spangcnt);
      for(i=0;i<dcount;i++){
        for(j=0;j<=NPTS-zflg;j++){
         for(k=0;k<2+zflg;k++){
    fscanf(file_ptr,"%f",&xx);
    slices[i][j][k]=xx;
       } /* end of for k */
      } /* end of for j */
    if(zflg==0){
    for(m=0;m<dcount;m++){
      for(j=0;j<NPTS;j++){
     slices[m][j][2]=slices[m][NPTS][0];
     } /* end of for j */
    } /* end of for m */
    } /* end of if zflg */
    else {
    for(m=0;m<dcount;m++){
    temp=0;
      for(j=0;j<NPTS;j++){
     temp=temp+slices[m][j][2];
     } /* end of for j */
     slices[m][NPTS][0]=temp/(NPTS-1);
    } /* end of for m */
    } /* end of else */
```

```
        } /* end of for i */
        fclose(file_ptr);
        set1=slices[0][NPTS][0]+1;
        set2=slices[dcount-1][NPTS][0];
        if(spangcnt>4)mflg=0;
}
/* here to load master shape data */


masterlod()
{
  FILE *fopen(),*file_ptr;
  char *string1,string2[50],*string3,string4[50],buffer[20];
   int count,i,j,k,m;
  float xx,yy,temp;

    string1 = "Enter file name ";
   string3 = "r";

        for(i=0;i<50;i++){
        string2[i]='\0';
        string4[i]='\0';
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }

   name = &string4[0];
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
    putline(string2,1.0,5.0);
    count = getline(string2,i,50);
    sscanf(&string2[i],"%s",name);
    winpush();
```

```
 file_ptr=fopen(name,string3);
      fscanf(file_ptr,"%d",&mcount);
      fscanf(file_ptr,"%d",&spangcnt);
        for(i=0;i<mcount;i++){
          for(j=0;j<=NPTS;j++){
            for(k=0;k<2;k++){
      fscanf(file_ptr,"%f",&xx);
      interpsl[i][j][k]=xx;
         } /* end of for k */
        } /* end of for j */
      } /* end of for i */
      fclose(file_ptr);
}
/* here to load shape data */


shapelod()
{
  FILE *fopen(),*file_ptr;
  char *string1,string2[50],*string3,string4[50],buffer[20];
   int count,i,j,k,m;
  float xx,yy,temp;

   string1 = "Enter file name ";
  string3 = "r";

      for(i=0;i<50;i++){
      string2[i]='\0';
      string4[i]='\0';
      }
      i=0;
      while(string1[i] !='\0'){
      string2[i]=string1[i];
      i++;
      }

  name = &string4[0];
      winset(textwin);
      winpop();
```

```
      c3s(whitevec);
      clear();
      c3s(blackvec);
 putline(string2,1.0,5.0);
 count = getline(string2,i,50);
 sscanf(&string2[i],"%s",name);
 winpush();
incount=0;
 file_ptr=fopen(name,string3);
      fscanf(file_ptr,"%d",&scount);
        for(i=0;i<scount;i++){
          for(k=0;k<3;k++){
             fscanf(file_ptr,"%f",&xx);
             infile[i][k]=xx;
        } /* end of for k */
     if(infile[i][0]!=0)incount++;
      } /* end of for i */
      fclose(file_ptr);
      findu();
}
/* here to convert circumference to u value */

findu()
{
       int i,j,k;
       float aa,bb,cc,xx;


       datum1 =  360.1;
       measblock[6]=datum1;
       datum2 = 359.9;
       measblock[7]=datum2;

       for(j=0;j<scount;j++){
       aa=0.0;
       bb=UMAX;
       cc=(bb-aa)/2;
       k=(int)infile[j][0];
```

```
        for(i=0;i<=DEF;i++){
        crslice(cc,k);
        bezplot(1);
        lintape(datum1,datum2,1);
        if(measblock[1]<infile[j][2])aa=cc;
        else bb=cc;
        cc=aa+(bb-aa)/2;
      }
      infile[j][2]=cc;
      measblock[8]=cc;
      }
}
/* here to create slice data from master file */

crslice(point,lincnt)
      float point;
      int lincnt;
{
      int u,v,p,m,n,o;
      float frangle[4],temp[4],q;

      p=(int)point;
      q=point-p;
      n=(int)mcount/4;

      frangle[3]=1.0;
      frangle[2]=q;
      frangle[1]=frangle[2]*frangle[2];
      frangle[0]=frangle[1]*frangle[2];

      colmat(frangle,cardinal,temp);

      for(v=0;v<=NPTS+1;v++){
      DPOLY[v][0]=0;
      DPOLY[v][1]=0;
        for(u=0;u<4;u++){
        if(p==0)o= -1;
        else if(p==2)o=1;
```

```
        else o=0;
        if(u+o<0){
        DPOLY[v][0]=DPOLY[v][0]+temp[u]*(2*interpsl[n+
                        lincnt][v][0]-interpsl[lincnt][v][0]);
        DPOLY[v][1]=DPOLY[v][1]+temp[u]*(2*interpsl[n+
                        lincnt][v][1]-interpsl[lincnt][v][1]);
      } /* end of if u */
        else if(u+o>3){
        DPOLY[v][0]=DPOLY[v][0]+temp[u]*(2*interpsl[3*n+
                lincnt][v][0]-interpsl[2*n+lincnt][v][0]);
        DPOLY[v][1]=DPOLY[v][1]+temp[u]*(2*interpsl[3*n
                +lincnt][v][1]-interpsl[2*n+lincnt][v][1]);
      } /* end of else if u */
        else{
          m=(u+o)*n+lincnt;
        DPOLY[v][0]=DPOLY[v][0]+temp[u]*interpsl[m][v][0];
        DPOLY[v][1]=DPOLY[v][1]+temp[u]*interpsl[m][v][1];
        } /* end of else */
       } /* end of for u */
      } /* end of for v */
}
/* here to linearly interpolate shape file */

shapeint()
{
      int i,j,maxfile,incnt,linesp;
      float intht,intu;



      shapecnt=2;
      incnt=0;
      for(i=0;i<2;i++)shapefile[0][i]=infile[0][i+1];/* copy 1st
                                                        line */
      for(i=0;i<2;i++)shapefile[1][i]=infile[0][i+1];/* and
                                                        again */
      for(i=0;i<2;i++)shapefile[2][i]=infile[0][i+1];/* and
                                                        again */
```

```
        maxfile=infile[incount][0];/* read last linefor max file
                                                    length */
        for(i=3;i<=maxfile;i++){ /*loop for every line of the
                                                    o/p file */

    /* interpolate between input file lines */
        linesp=infile[incnt+1][0]-infile[incnt][0];
        if(infile[incnt][0]==0)linesp=linesp-2;
        if(linesp<0 || linesp>10)linesp=0;
        intht=(infile[incnt+1][1]-infile[incnt][1])/linesp;
        intu =(infile[incnt+1][2]-infile[incnt][2])/linesp;
          for(j=0;j<linesp;j++){
        shapefile[shapecnt][0]=infile[incnt][1]+j*intht;
        shapefile[shapecnt][1]=infile[incnt][2]+j*intu;
          shapecnt++;
             if(shapecnt>MAXSHAPEFILE)shapecnt=MAXSHAPEFILE;
          } /* end of for j */
         incnt++;
        } /* end of for i */

  shapefile[shapecnt][0]=infile[scount-1][1];
  shapefile[shapecnt][1]=infile[scount-1][2];

        dcount=shapecnt;

}
/* here to cubic spline interpolate shape file */

xshapeint()
{
        int i,j,k,l,m,maxfile,incnt,linesp;
        float frangle[4],temp[4],q;



        shapecnt=2;
        incnt=0;
```

```
    for(i=0;i<2;i++)shapefile[0][i]=infile[0][i+1];/* copy 1st
                                                        line */
    for(i=0;i<2;i++)shapefile[1][i]=infile[0][i+1];/* and
                                                        again */
    for(i=0;i<2;i++)shapefile[2][i]=infile[0][i+1];/* and
                                                        again */
    maxfile=infile[incount][0];/* read last linefor max file
                                                    length */
    for(i=3;i<=maxfile;i++){ /*loop for every line of the
                                                    o/p file */

   /* interpolate between input file lines */
      linesp=infile[incnt+1][0]-infile[incnt][0];
      if(infile[incnt][0]==0)linesp=linesp-2;
      if(linesp<0 || linesp>10)linesp=0;
        for(j=0;j<linesp;j++){
            q=(float)j/linesp;
            frangle[3]=1.0;
            frangle[2]=q;
            frangle[1]=frangle[2]*frangle[2];
            frangle[0]=frangle[1]*frangle[2];
            colmat(frangle,cardinal,temp);
            shapefile[shapecnt][0]=0.0;
            shapefile[shapecnt][1]=0.0;
              for(k=0;k<4;k++){
              m=incnt+k-1;
if(m<0){
shapefile[shapecnt][0]=shapefile[shapecnt][0]+
                    temp[k]*(2*infile[0][1]-infile[1][1]);
shapefile[shapecnt][1]=shapefile[shapecnt][1]+
                    temp[k]*(2*infile[0][2]-infile[1][2]);
      }
 else if(m==scount){
            shapefile[shapecnt][0]=shapefile[shapecnt][0]-
            temp[k]*(2*infile[scount][1]-infile[scount-1][1]);
            shapefile[shapecnt][1]=shapefile[shapecnt][1]-
            temp[k]*(2*infile[scount][2]-infile[scount-1][2]);
      }
```

```
 else {
                shapefile[shapecnt][0]=shapefile[shapecnt][0]+
                                        temp[k]*infile[m][1];
                shapefile[shapecnt][1]=shapefile[shapecnt][1]+
                                        temp[k]*infile[m][2];
     }
                 } /* end of for k */
                shapecnt++;
                if(shapecnt>MAXSHAPEFILE)shapecnt=MAXSHAPE
                                                        FILE;
          } /* end of for j */
          incnt++;
        } /* end of for i */

  shapefile[shapecnt][0]=infile[scount-1][1];
  shapefile[shapecnt][1]=infile[scount-1][2];

        dcount=shapecnt;
}
/* here to re-create slice  file */


resltxt()
{
        int i;
        float height;


        crslice(shapefile[0][1],0);
        stslice(shapefile[0][0],0);
        crslice(shapefile[0][1],0);
        stslice(shapefile[0][0],1);
        crslice(shapefile[0][1],0);
        stslice(shapefile[0][0],2);

        for(i=3;i<shapecnt;i++){
        crslice(shapefile[i][1],i);
        stslice(shapefile[i][0],i);
        } /* end of for i */
```

```
        crslice(shapefile[shapecnt][1],shapecnt);
        stslice(shapefile[shapecnt][0],shapecnt);
        crslice(shapefile[shapecnt][1],shapecnt);
        stslice(shapefile[shapecnt][0],shapecnt+1);
        shapecnt=shapecnt+1;
        dcount=shapecnt;
        set1=infile[0][1];
        set2=infile[incount][1];
        bslcptch(shapecnt);/* make patches */
}
/* here to write slice spline data to a file */

splinesav()
 {
        FILE *fopen(),*file_ptr;
        char *string1,*string5,string2[50],*string3;
        char string4[50],spacing[50];
        int bytecount,count,i,space;

        string1 = "Enter file name ";
        string5 = "Enter slice spacing ";
        string3 = "w";

        for(i=0;i<50;i++){
        string2[i]='\0';
        string4[i]='\0';
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
```

```
  putline(string2,1.0,5.0);
  count = getline(string2,i,50);
  sscanf(&string2[i],"%s",string4);
        for(i=0;i<50;i++){
        string2[i]='\0';
        spacing[i]='\0';
        }
        c3s(whitevec);
        clear();
        c3s(blackvec);
        i=0;
        while(string5[i] !='\0'){
        string2[i]=string5[i];
        i++;
        }
   putline(string2,1.0,5.0);
   count = getline(string2,i,50);
   sscanf(&string2[i],"%s",spacing);
   space=atoi(spacing);
   winpush();
        dcount=0;
        for(i=(int)set1;i<=(int)set2;i=i+space){
        getpoly((float)i);
        stslice((float)i,dcount);
        dcount++;
        if(dcount>100)break;
        }
pslice();
  file_ptr=fopen(string4,string3);
  MINIHEAD[1] = (float)dcount;
  MINIHEAD[2] = (float)mflg;
  fwrite(MINIHEAD,sizeof(*MINIHEAD),20,file_ptr);
  bytecount = fwrite(slices,sizeof(*slices),dcount,file_ptr);
        fclose(file_ptr);
}

/* here to initialise NDATA[] */
```

```
initarry()
        {
        int i;
        for(i=0;i<=NARRAY;i++)NDATA[i]=0;
        }
/* here to give line drawing */
ldrw()
{
        register int i,j;
        int angl,hite;
        float theta,radius,wscale;
        float vert[3];

        wscale=SCALE;
        p ıshmatrix();
                for(i=minangl;i<maxangl;i++){
                  angl=STEPS-i;
                while(angl>=STEPS)angl=angl-STEPS;
                while(angl<0)angl=STEPS+angl;
                if(lineflg == 1)bgnline();
                        for(j=minhite;j<maxhite;j++){
                        hite=j;
                        if(hite>NFRAME)hite=NFRAME;
                        if(hite<0)hite=0;
                        radius=radget(angl,hite,0);
                        if(radius>0){
                                theta=i*ANGLE*RAD;/* ANGLE IN
                                                     RADIANS */
                                vert[0]=radius*sin(theta)/wscale;
                                vert[1]=hite*VSTEP/wscale;
                                vert[2]=radius*cos(theta)/wscale;
                                if(lineflg !=1)bgnpoint();
                                v3f(vert);
                                 if(lineflg !=1){
                                 drawcube();
                                 endpoint();
                                 }
                                }
```

```
                }
            if(lineflg == 1){
            endline();
            }
        }
        popmatrix();
        refaxes();
}

refaxes()
{
        pushmatrix();
        translate(-transx,-transy,-transz);
        rotate((int)-rotx, 'x');
        rotate((int)-roty, 'y');
        rotate((int)-rotz, 'z');
        c3s(blackvec);
        move(0, 0, 0);
        rdri(0, 0, 1);
        move(0, 0, 0);
        rdri(0, 10, 0);
        move(0, 0, 0);
        rdri(0,-10, 0);
        move(0, 0, 0);
        rdri(1, 0, 0);
        cmov(0.1, 0.1, 1);
        charstr("z");
        cmov(0.1, 1, 0.1);
        charstr("y");
        cmov(1, 0.1, 0.1);
        charstr("x");
        popmatrix();
 }
 /* here to draw angle reference grid */

 drwangref()
 {
        int i,flg;
```

```
        float xx,yy,theta,radius;
        char nn[2];

        pushmatrix();
        translate(-transx,0,-transz);

        radius=4.5;

        for(i=0;i<NPTS;i++){
        theta=spangles[spangcnt][i];
        if(theta>=0){
        theta=theta*RAD;
        flg = 0;
        }
        else {
        theta= -theta*RAD;
        flg = 1;
        }
        xx= -radius*sin(theta);
        yy= -(radius*cos(theta));
        c3s(greenvec);
        move(0.0,0.0,0.0);
        rdr(xx,yy,0.0);
        if(flg==0)c3s(blackvec);
        else c3s(redvec);
        sprintf(nn,"%d",i);
        cmov2(xx-0.2,yy);
        charstr(nn);
        }
        popmatrix();
}

drawcube()
{
        /* draw the outline of the cube */

        rdr(0.01, -0.01, -0.01);
        rdr(0.01, 0.01, -0.01);
```

```
        rdr(-0.01, 0.01, -0.01);
        rdr(-0.01, -0.01, -0.01);
        rdr(-0.01, -0.01, 0.01);
        rdr(0.01, -0.01, 0.01);
        rdr(0.01, 0.01, 0.01);
        rdr(-0.01, 0.01, 0.01);
        rdr(-0.01, -0.01, 0.01);
        rmv(-0.01, 0.01, -0.01);
        rdr(-0.01, 0.01, 0.01);
        rmv(0.01, 0.01, -0.01);
        rdr(0.01, 0.01, 0.01);
        rmv(0.01, -0.01, 0.01);
        rdr(0.01, -0.01, -0.01);
        closeobj();
}

slice(win,scl,tap)
        int win,scl,tap;
{
        float mouse,wscale;

        wscale=SCALE;
        while(getbutton(MIDDLEMOUSE)){
        mouse = getvaluator(MOUSEY);
        mouse=((mouse*(PLUSY-(MINUSY))/(YMAXSCREEN-
                30)+MINUSY))/zoom-transy;
        if(win==0)winset(slicewin);
        else winset(cubewin);
        c3s(whitevec);
        clear();
        c3s(bluevec);
        sliceht=mouse;
        slicedrw(sliceht);
        if(splineflg!=0){
        getpoly(sliceht*wscale);
        bezplot(scl);
        if(tap==0)lintape(datum1,datum2,1);
        dpolyplot(NPTS);
```

```
        sliceref();
        }
        swapbuffers();
        printdata(datablock);
        }
}
/* here to give slice drawing */

slicedrw(height)
float height;
{
        register int i,hite;
        int angl,ht;
      float theta,angle,radius,wscale,circ,xx,yy;
        float vert[3];
        float vertemp[3];
        char num[10];

        c3s(bluevec);
        wscale=SCALE;
        circ = 0;
        vert[2]=0;
        vertemp[2] = vert[2];
                hite=height*wscale/VSTEP;
                angle=RAD*roty/10;
                bgnclosedline();
                for(i=1;i<=STEPS;i++){
                  angl=STEPS-i;
                radius=radget(angl,hite,i-1);
                if(radius >0){
                        theta=i*ANGLE*RAD;/* ANGLE IN
                                             RADIANS */
                        xx= radius*sin(theta);
                        yy= -(radius*cos(theta));
                        vert[0] = (xx*cos(angle)-    yy*sin(angle))+deltax;
                        vert[1] = (xx*sin(angle)+yy*cos(angle))+deltay;
      circ = circ + fhypot((vert[0]-vertemp[0]),(vert[1]-vertemp[1]));
                 vertemp[0]=vert[0];
```

```
                vertemp[1]=vert[1];
                vert[0]=vert[0]/wscale;
                vert[1]=vert[1]/wscale;
                v3f(vert);
                                } /* end of if radius */
            } /* end of for i */
                endclosedline();
        c3s(blackvec);
        obref(datum1);
        c3s(redvec);
        obref(datum2);

                sprintf(num,"%.0f",height*wscale);
                cmov2(-0.25,0.0);
                charstr(num);

}

sliceref()
{
        c3s(redvec);
        bgnline();
        v2i(vert1);
        v2i(vert2);
        endline();
        bgnline();
        v2i(vert3);
        v2i(vert4);
        endline();
}
heightref(temp)
        float temp;
{
        float vert[3];

        c3s(redvec);
        bgnline();
        vert[0]= PLUSX;
```

```
        vert[1]=temp;
        vert[2]=0;
        v3f(vert);
        vert[0]= MINUSX;
        v3f(vert);
        endline();
        bgnline();
        vert[0]=0;
        vert[2]= PLUSZ;
        v3f(vert);
        vert[2]= MINUSZ;
        v3f(vert);
        endline();
}
/* here to give vertical slice drawing */

vertdrw()
{
        int i, hite;
        int angl,count,r;
      float radius,wscale;
        float vert[2];

        c3s(whitevec);
        clear();
        c3s(bluevec);

    wscale=SCALE;
                for(i=0;i<2;i++){
                angl=i*STEPS/2+STEPS-(stangle-roty/10)/ANGLE;
                if(angl>=STEPS)angl=angl-STEPS;
                if(angl<0)angl=angl+STEPS;
                bgnline();
                        for(hite=minhite;hite<maxhite;hite++){
                        r=NDATA[NPOINT];
                        count=NDATA[NPOINT+1];
                                if(count>0){
                                radius=r/count;
```

```
                        vert[0]=radius/wscale;
                        if(i==1)vert[0]= -vert[0];
                        vert[1]=hite*VSTEP/wscale;
                        v2f(vert);
                        }
                }
            endline();
        }
}
/* here to get radius */

int radget(angl,hite,flg)
        int angl,hite,flg;
{
        int r,count,radius,old;


        if(flg==0)old=10000;
        r=NDATA[NPOINT];
        count=NDATA[NPOINT+1];
        if(count>0){
        radius=r/count;
         if(radius >= (old+ARMTOL) && armflg > 0 && ((angl>25
               && angl<50) || (angl>100 && angl<125))){
         radius=old;
         }
        else {
           old=radius;
           }
        }
        else radius=0;
        return(radius);
}
 /* here to re-create an input file from patch data */


 redo()
 {
        int i,j,hite,angl,vstep,resltn,radius,step;
```

```
float angle,xx,yy;
float phi,pi2,inc;

vstep=5;
pi2=2*PI;
inc=0.041887902;
resltn=750;
hite=(int)set1/VSTEP;
initarry();
for(i=(int)set1+1;i<(int)set2;i=i+vstep){
getpoly((float)i);
step=0;
  for(j=0;j<resltn;j++){
 cardret(j,resltn);
 xx=apoly[0];
 yy=apoly[1];
 if(xx>0)phi=fatan(yy/xx);
 else if(xx<0)phi=PI+fatan(yy/xx);
 else if(xx==0 && yy>0)phi= PI/2;
 else if(xx==0 && yy<0)phi= 3*PI/2;
 else if(xx==0 && yy==0)phi=0;
 phi=3*PI/2-phi;
 if(phi<0)phi=pi2+phi;
 if(phi>pi2)phi=phi-pi2;
  if(phi>=step*inc){
  angl=step;
  step++;
   if(angl<0)angl=STEPS+angl;
   if(angl>=STEPS)angl=angl-STEPS;
  radius=(int)fhypot(xx,yy);
   NDATA[NPOINT]=radius;
   NDATA[NPOINT+1]=1;
if(mflg ==1){
   angl=STEPS-angl-1;
   if(angl<0)angl=STEPS+angl;
   if(angl>=STEPS)angl=angl-STEPS;
   NDATA[NPOINT]=radius;
   NDATA[NPOINT+1]=1;
```

```
            } /* end of if mflg */
           } /* end of if fabs */
          } /* end of for j */
           hite++;
         } /* end of for i */
}
/* here to print slice array */

pslice()
{
        int i,j;

        for(i=0;i<dcount;i++){
          for(j=0;j<=NPTS;j++){
           printf("%d %7.2f %7.2f  ",j,slices[i][j][0],slices[i][j][1]);
          if(j==3 || j==7 || j==11 || j==15)printf("\n");
          }
        printf("\n");
        }
}
 /* here to print shapefile */
 shapeprnt(maxfile)
        int maxfile;
 {
        int i;

        for(i=0;i<=maxfile;i++){
        printf("%d %f %f\n",i,shapefile[i][0],shapefile[i][1]);
        }
 }
 /* here to print infile */
 inprnt()
 {
        int i;


        for(i=0;i<=incount;i++){
        printf("%d %f %f %f\n",i,infile[i][0],infile[i][1],infile[i][2]);
```

```
        }
}
```

/* lass4.c */

```
#include "device.h"
#include "lassdefs"
#include <gl.h>
#include <math.h>

#define BEZIER   1
#define BSPLINE  2
#define CARDINAL 3


float trans[4][4] = {
        { 1,  0,  0, 0 },
        { 0,  1,  0, 0 },
        { 0,  0,  1, 0 },
        { 0,  0,  0, 1 },
        };
Matrix cardinal = {
        {-0.5, 1.5,-1.5, 0.5},
        { 1.0,-2.5, 2.0,-0.5},
        {-0.5, 0.0, 0.5, 0.0},
        { 0.0, 1.0, 0.0, 0.0},
        };
Matrix bspline = {
        {-0.1667, 0.5000,-0.5000, 0.1667 },
        { 0.5000,-1.0000, 0.5000, 0.0    },
        {-0.5000, 0.0   , 0.5000, 0.0    },
        { 0.1667, 0.6667, 0.1667, 0.0    },
        };

Matrix bezcon = {
        { 0.0    , 0.0    , 0.0    , 1.0    },
        { 0.333333,-0.888889, 2.666667,-1.111111},
        {-1.111111, 2.666667,-0.888889, 0.333333},
```

```
        { 1.0    , 0.0    , 0.0    , 0.0    },
        };

Matrix wscalematrix[4][4] = {
        {1.0/SCALE, 0.0, 0.0, 0.0 },
        {0.0, 1.0/SCALE, 0.0, 0.0 },
        {0.0, 0.0, 1.0/SCALE, 0.0 },
        {0.0, 0.0, 0.0, 1.0/SCALE },
        };
Coord reflect[4][4] = {
        { -1.0, 0.0, 0.0, 0.0 },
        {  0.0,-1.0, 0.0, 0.0 },
        {  0.0, 0.0,-1.0, 0.0 },
        {  0.0, 0.0, 0.0,-1.0 },
        };

float spangles[7][NPTS] = {
        { 0.0,  4.0,  15.0, 25.0, 40.0,  55.0,  70.0,  90.0,  115.0, 120.0,
                                  130.0, 140.0, 150.0, 160.0, 170.0, 180.0 },
        { 0.0,  4.0,  15.0, 25.0, 40.0,  55.0, -70.0, -90.0, -115.0, 120.0,
                                  130.0, 140.0, 150.0, 160.0, 170.0, 180.0 },
        { 0.0,  4.0,  15.0, 25.0, 40.0,  55.0,  70.0,  90.0,  104.0, 120.0,
                                  130.0, 140.0, 150.0, 160.0, 170.0, 180.0 },
        { 0.0,  4.0,  15.0, 25.0, 40.0,  55.0, -65.0, -90.0,  110.0, 120.0,
                                  130.0, 140.0, 150.0, 160.0, 170.0, 180.0 },
        { 0.0, 10.0,  20.0, 30.0, 40.0,  50.0,  60.0,  70.0,   80.0, 120.0,
                                  130.0, 140.0, 150.0, 160.0, 170.0, 180.0 },
        { 0.0, 24.0,  48.0, 72.0, 96.0, 120.0, 144.0, 168.0,  192.0, 216.0,
                                  240.0, 264.0, 288.0, 312.0, 336.0, 360.0 },
        { 0.0, 24.0,  48.0, 72.0, 96.0, 120.0, 144.0, 168.0,  192.0,
                -216.0,-240.0,-264.0,-288.0,-312.0,-336.0, 360.0 },
        };

        float temp[4][4];
        float ptemp[4][4];
        float measblock[10];
        float patchx[4][4];
        float patchy[4][4];
```

```
        float patchz[4][4];
        int obdatacnt;
        extern float obdata[STEPS*NFRAME][2];
        extern float irismat[4][4];
        extern short NDATA[NARRAY];
        extern float DPOLY[NPTS+1][2];
        extern float bslice[NFRAME][NPTS][2];
        extern float zoom,pmax,pmin;
        extern float transx, transy, transz,deltax,deltay;
        extern int rotx,roty,rotz,patchflg,spangcnt,mflg,revflg;
        extern short blackvec[3];
        extern short redvec[3];
        extern short greenvec[3];
        extern short bluevec[3];
        extern short whitevec[3];
        float slicedata[STEPS][2];
        char num[50];

printemp()
{
        int i, j;

        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
          printf("%f  ",temp[i][j]);
          }
        printf("\n");
        }
        printf("\n");
}

/* here to plot Cardinal curve */

bezplot(flg)
        int flg;
{
        int s,t,u,npoints,temp1;
        float poly[3],distnce[2],frangle[4],temp[4],xx,yy,wscale;
```

```
float theta,phi,lasttheta,hyp,lasthyp;
float area1,area2,pi2,artan();

wscale=SCALE;
pushmatrix();

c3s(redvec);
poly[2]=0;
pi2=2*PI;

obdatacnt=0;
npoints=50;
distnce[0]=0;
distnce[1]=0;
area1=0;
area2=0;
for(s= -1;s<(NPTS-2);s++){
bgnline();
for(t=0;t<=npoints;t++){
frangle[3]=1.0;
frangle[2]=(float)t/npoints;
frangle[1]=frangle[2]*frangle[2];
frangle[0]=frangle[1]*frangle[2];

colmat(frangle,cardinal,temp);

poly[0]=0;
poly[1]=0;

  for(u=0;u<4;u++){
   if(s<0 && u==0){
   poly[0]=poly[0]-temp[0]*DPOLY[1][0];
   poly[1]=poly[1]+temp[0]*DPOLY[1][1];
   }
    else if (s!=(NPTS-3)){
   poly[0]=poly[0]+temp[u]*DPOLY[u+s][0];
   poly[1]=poly[1]+temp[u]*DPOLY[u+s][1];
   }
```

```
 if(s==(NPTS-3)){
  if(u==3){
  poly[0]=poly[0]-temp[3]*DPOLY[NPTS-2][0];
  poly[1]=poly[1]+temp[3]*DPOLY[NPTS-2][1];
  }
  else{
  poly[0]=poly[0]+temp[u]*DPOLY[u+s][0];
  poly[1]=poly[1]+temp[u]*DPOLY[u+s][1];
  }
 }
}

hyp=fhypot(poly[0],poly[1]);
theta=artan(poly[1],poly[0]);
phi= theta-lasttheta;
if(phi<0)phi= -phi;
if(t>0 && phi<PI/10){
distnce[0]=distnce[0]+fhypot((poly[0]-xx),(poly[1]-yy));
area1=hyp*lasthyp*sin(phi)/2+area1;
}

xx=poly[0];
yy=poly[1];
lasthyp=hyp;
lasttheta=theta;

obdata[obdatacnt][0]=poly[0];
obdata[obdatacnt][1]=poly[1];
obdatacnt++;

if(flg==0){
poly[0]=poly[0]/wscale;
poly[1]=poly[1]/wscale;
}
v3f(poly);
}
endline();
}
```

```
        distnce[1]=0;
        area2=0;
        if(mflg==1){ /* plot mirror image */
        distnce[1]=distnce[0];
        area2=area1;
        temp1=obdatacnt-2;
        bgnline();
        for(s=temp1;s>=0;s--){
        obdata[obdatacnt][0]= -obdata[s][0];
        obdata[obdatacnt][1]= obdata[s][1];
        obdatacnt++;
        poly[0]= -obdata[s][0];
        poly[1]= obdata[s][1];
        if(flg==0){
        poly[0]=poly[0]/wscale;
        poly[1]=poly[1]/wscale;
        }
        v3f(poly);
        }
        endline();
    }
        popmatrix();
        measblock[2]=(distnce[0]+distnce[1]);
        measblock[3]= (area1+area2)/100;
        pmeas(measblock);
}

zeroaxes()
        {
        transx = 0; transy = 0; transz = 0;
        rotx = 0; roty = 0; rotz = 0;
        rotax(1);
        }
/* here to plot defining polygon */

dpolyplot(npts)
        int npts;
{
```

```
        int i,xscale,yscale;

        xscale=SCALE;
        yscale=SCALE;

        pushmatrix();
        for(i=0;i<npts;i++)
                crossplot(DPOLY[i][0]/xscale,DPOLY[i][1]/yscale,i);
        popmatrix();
}
/* here to fit defining polygon to a slice */
fitpoly(hite)
        int hite;
{
        int i,ii,j,k,flg;
        float theta,radius,xx,yy,pi2,tan1,tan2,temp1,temp2;
        float xx1,xx2,yy1,yy2,angle,radstep;
        float spang,artan(),stpangle;


        pi2=2*PI;
        stpangle=ANGLE*RAD;
        angle=(float)roty;
        angle= -RAD*angle/10;
        if(angle>pi2)angle=angle-pi2;
        if(angle<0)angle=pi2+angle;
        ii=0;
        for(i=0;i<STEPS;i++){
                /* here to convert to cartesian co-ords */
        theta= i*ANGLE*RAD;
        if(revflg==0){
          radius=radget(i,hite,0);
          xx= radius*sin(theta);
          yy= -(radius*cos(theta));
          slicedata[ii][0] = (xx*cos(angle)-yy*sin(angle))-deltax;
          slicedata[ii][1] = (xx*sin(angle)+yy*cos(angle))+deltay;
          }
        else {
```

```
 radius=radget((STEPS-i),hite,0);
 xx= radius*sin(theta);
 yy= -(radius*cos(theta));
 slicedata[ii][0] = (xx*cos(pi2-angle)-yy*sin(pi2-
                angle))+deltax;
 slicedata[ii][1] = (xx*sin(pi2-angle)+yy*cos(pi2-
                angle))+deltay;
 }
if(radius<=2)ii--;
else ii++;
} /* end of for i */

j=0;k=0;
while(k<2){
  for(i=0;i<ii-1;i++){
 xx1=slicedata[i][0];
 xx2=slicedata[i+1][0];
 yy1=slicedata[i][1];
 yy2=slicedata[i+1][1];
 tan1=artan(yy1,xx1);
 tan2=artan(yy2,xx2);
 if(tan1==0)tan1=temp1;
 if(tan2==0)tan2=temp2;
 temp1=tan1;
 temp2=tan2;
 spang=spangles[spangcnt][j];
 if(spang>=0)flg=0;
 else {
        flg=1;
        spang= -spang;
    }
 spang=spang*RAD-PI/2;
 if(spang<0)spang=pi2+spang;
 if(spang>pi2)spang=spang-pi2;
  xx=(xx1+xx2)/2;
  yy=(yy1+yy2)/2;
  radius=fhypot(xx,yy);
   if(tan2>=spang && tan1<=spang
```

```
          && fabs(tan2-tan1)<PI){
        if(flg==0){
         DPOLY[j][0]= -radius*cos(spang);
         DPOLY[j][1]= radius*sin(spang);
        } /* end of if flg */
        j++;
       } /* end of if tan2 */
      else if(yy1<0 && yy2>0 && xx1>0 && j>4
      && fabs(tan2-tan1)>3*PI/2){
         if(flg==0){
         DPOLY[j][0]= -radius*cos(spang);
         DPOLY[j][1]= radius*sin(spang);
         } /* end of if flg */
        j++;
       } /* end of if yy */
         if(j>=NPTS)break;
      } /* end of for i */
      k++;
      if(k>=2 && j<(NPTS-1)){
      j++;
      k=0;
      }
      if(j>=NPTS)break;
      } /* end of while k */
}
/* here to fit an elipse to a slice */

elipse()
{
       int i;
       float theta,sinsq,cossq,aa,bb,radius,pi2;

       pi2=2*PI;
       aa=DPOLY[0][1];
       aa=aa*aa;
       bb=DPOLY[7][0];
       bb=bb*bb;
       for(i=1;i<7;i++){
```

```
        theta=(spangles[spangcnt][i]);
        if(theta<0)theta= -theta;
        theta=theta*PI/180;
        if(theta>2*PI)theta=theta-pi2;
        if(theta<0)theta=theta+pi2;
        sinsq= sin(theta)*sin(theta);
        cossq= cos(theta)*cos(theta);
        radius= 1/(sqrt((cossq/aa)+(sinsq/bb)));
        DPOLY[i][1]= -radius*cos(theta);
        DPOLY[i][0]= -radius*sin(theta);
      } /* end of for i */
      pmeas(measblock);
}
/* here to plot 2-D crosses */

crossplot(xx,yy,nnn)
      float xx,yy;
      int  nnn;
{
      float vert[3],inc;
      char nn[2];

      c3s(blackvec);
      vert[2]=0;

      inc = 0.02;
      if(patchflg==1 || patchflg==4)inc=0.15;
      bgnline();
      vert[0]=xx-inc;
      vert[1]=yy;
      v3f(vert);
      vert[0]=xx+inc;
      vert[1]=yy;
      v3f(vert);
      endline();
      bgnline();
      vert[0]=xx;
      vert[1]=yy-inc;
```

```
        v3f(vert);
        vert[0]=xx;
        vert[1]=yy+inc;
        v3f(vert);
        endline();
        }
/* matrix multiply of 4 X 4 array , A * B result in R*/

matmult(mata,matb,matr)
        float mata[4][4],matb[4][4],matr[4][4];
{
        int i,j,k;
        float temp[4][4];

        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
         temp[i][j]=0;
           for(k=0;k<4;k++){
          temp[i][j]=temp[i][j]+mata[i][k]*matb[k][j];
           }
         }
        }
        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
         matr[i][j]=temp[i][j];
         }
        }
 }
 /* matrix transposition of 4 X 4 array , result in R*/

 transmat(mata,matr)
        float mata[4][4],matr[4][4];
 {
        int i,j;
        float temp[4][4];

        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
```

```
        temp[i][j]=mata[j][i];
        }
       }

      for(i=0;i<4;i++){
       for(j=0;j<4;j++){
       matr[i][j]=temp[i][j];
       }
      }
}
/*matrix multiply 4 X 4 array times 4 X 1 array, A * B result in R */

matcol(mata,matb,matr)
      float mata[4][4],matb[4],matr[4];
{
      int i,j;
      float temp[4];

      for(i=0;i<4;i++){
      temp[i]=0;
        for(j=0;j<4;j++){
       temp[i]=temp[i]+mata[i][j]*matb[j];
       }
      }
      for(i=0;i<4;i++)matr[i]=temp[i];
}
/*matrix multiply 4 X 1 array times 4 X 4 array, A * B result in R */

colmat(mata,matb,matr)
      float mata[4],matb[4][4],matr[4];
{
      int i,j;
      float temp[4];

      for(i=0;i<4;i++){
      temp[i]=0;
        for(j=0;j<4;j++){
       temp[i]=temp[i]+mata[j]*matb[j][i];
```

```
        }
       }
       for(i=0;i<4;i++)matr[i]=temp[i];
}

/* here to plot large bspline patches */

xpatchplot2()
{
       Coord tempx[4][4];
       Coord tempy[4][4];
       Coord tempz[4][4];

       defbasis(BSPLINE,bspline);
       patchcurves(18,18);
       patchprecision(20,20);
       patchbasis(BSPLINE,BSPLINE);
       c3s(redvec);
       matmult(wscalematrix,patchx,tempx);
       matmult(wscalematrix,patchy,tempy);
       matmult(wscalematrix,patchz,tempz);
       patch(tempx,tempz,tempy);
       if(mflg==1){
       matmult(reflect,tempy,tempy);
       patch(tempx,tempz,tempy);
 }
}
/* here to print patch x,y,z matrices */

pmatprnt()
{
       int i;

       printf("patchx\n");
       for(i=0;i<4;i++)
       printf("%4.2f %4.2f %4.2f %4.2f\n",
              patchx[i][0],patchx[i][1],patchx[i][2],patchx[i][3]);
       printf("patchy\n");
```

```
        for(i=0;i<4;i++)
        printf("%4.2f %4.2f %4.2f %4.2f\n",
                patchy[i][0],patchy[i][1],patchy[i][2],patchy[i][3]);
        printf("patchz\n");
        for(i=0;i<4;i++)
        printf("%4.2f %4.2f %4.2f %4.2f\n",
                patchz[i][0],patchz[i][1],patchz[i][2],patchz[i][3]);
}
/* here to print bslice array */

bsprint()
{
        int hite,i;
        int hitemin,hitemax,wscale;


        wscale=SCALE;
        hitemin=pmin*wscale/VSTEP;
        hitemax=pmax*wscale/VSTEP;

        for(hite=hitemin;hite<=hitemax;hite++){
          for(i=0;i<7;i++){
          printf("%3.1f %3.1f ",bslice[hite][i][0],bslice[hite][i][1]);
          }
        printf("\n");
        }
        printf("\n");
}

/* subroutine to print array for test only */

printarr(array)
        float array[4][4];
{
        int i,dim;

        dim=4;
        for(i=0;i<dim;i++){
```

```
        printf("%.6g, %.6g, %.6g, %.6g\n",
                array[i][0],array[i][1],array[i][2],array[i][3]);
        }
    printf("\n");
}
/* here to shell sort oblique data according to angle */

shell(v,n) /* sort v[0].....v[n-1] into increasing order */
    int  n;
    float v[];

    {
    int gap, i, j, k;
    float  temp[2],tan[2],pi;

    k=1;
    pi=3.141592654;
    for (gap = n/2; gap > 0; gap /= 2)
        for (i = gap; i < n; i++)
            for(j = i-gap; j>=0 ; j-=gap){

                tan[0]=artan(v[j*2+k],v[j*2]);
                tan[1]=artan(v[(j+gap)*2+k],v[(j+gap)*2]);
                if(tan[0]>tan[1]){
                temp[0] = v[j*2];
                temp[1] = v[j*2+k];
                v[j*2] = v[(j+gap)*2];
                v[j*2+k] = v[(j+gap)*2+k];
                v[(j+gap)*2] = temp[0];
                v[(j+gap)*2+k] = temp[1];
                    }
                }
    }


/* lass5.c */

#include "device.h"
```

```
#include "lassdefs"
#include <gl.h>
#include <math.h>

#define BEZIER  1
#define BSPLINE  2
#define CARDINAL 3



Coord geomxfr[4][4];
Coord geomyfr[4][4];
Coord geomzfr[4][4];
Coord geomxfl[4][4];
Coord geomyfl[4][4];
Coord geomzfl[4][4];
Coord tempxf[4][4];
Coord tempyf[4][4];
Coord tempzf[4][4];

        extern wscalematrix[4][4];
        extern bspline[4][4];
        extern cardinal[4][4];
        extern trans[4][4];
        extern reflect[4][4];
        extern float DPOLY[NPTS+1][2];
        extern short blackvec[3];
        extern short redvec[3];
        extern short greenvec[3];
        extern short bluevec[3];
        extern short whitevec[3];
        extern long cubewin;
        extern float measblock[NPTS+1];
        extern float patches[PATCHMAX][3][4][4];
        extern float slices[MAXSLICES][NPTS+1][3];
        extern int patchcount,mflg, maxhite, minhite;
        extern float sliceht,set1,set2,datablock[20];
        extern float datum1,datum2;
```

```
/* here to store horizontal slices  */

stslice(height,num)
        int num;
        float height;
{
        int i;

        for(i=0;i<NPTS;i++){
        slices[num][i][0]=DPOLY[i][0];
        slices[num][i][1]=DPOLY[i][1];
        }
        slices[num][NPTS][0]=height;
}
/* here to retreive horizontal patch slices */

getslice(count)
        int count;
{
        int i;

        for(i=0;i<NPTS;i++){
          DPOLY[i][0]= slices[count][i][0];
          DPOLY[i][1]= slices[count][i][1];
        }
        DPOLY[NPTS][0] = slices[count][NPTS][0];
}
/* here to get slice polygon from patch data */

getpoly(height)
        float height;
{
        int i,j,k,count;
        float htmin,htmax,w,ww,www;
```

```
count=0;
for(i=0;i<patchcount;i++){
htmin = patches[i][1][1][1];
htmax = patches[i][1][2][1];
if(height>=htmin && height<htmax)break;
} /* end of for i */
count=i;
  w =(height-htmin)/(htmax-htmin);
 ww = w*w;
www = ww*w;

for(i=1;i<NPTS;i++){

  for(j=0;j<4;j++){
   for(k=0;k<4;k++){
  geomxfr[k][j] = patches[count][0][k][j];
  geomyfr[k][j] = patches[count][1][k][j];
  geomzfr[k][j] = -patches[count][2][k][j];
  }
 }
matmult(cardinal,geomxfr,tempxf);
matmult(cardinal,geomyfr,tempyf);
matmult(cardinal,geomzfr,tempzf);

matmult(tempxf,trans,tempxf);
matmult(tempyf,trans,tempyf);
matmult(tempzf,trans,tempzf);

DPOLY[i][0]=www*tempxf[0][1]+ww*tempxf[1][1]+
                    w*tempxf[2][1]+tempxf[3][1];
DPOLY[i][1]=www*tempzf[0][1]+ww*tempzf[1][1]+
                    w*tempzf[2][1]+tempzf[3][1];

if(i==1){
DPOLY[0][0]=www*tempxf[0][2]+ww*tempxf[1][2]+
            w*tempxf[2][2]+tempxf[3][2];
DPOLY[0][1]=www*tempzf[0][2]+ww*tempzf[1][2]+
```

```
                                w*tempzf[2][2]+tempzf[3][2];
        }
        count++;
        } /* end of for i */
        DPOLY[NPTS][0]=www*tempyf[0][1]+
                ww*tempyf[1][1]+w*tempyf[2][1]+tempyf[3][1];
        pmeas(measblock);
}
/* here to make large patches (total horiz. body section )
                                from 'n' bspline slices */


bslcptch(count)
        int count;
{
        int i,j,k,l,m;

        clrpatches();
        patchcount=0;
        for(j=0;j<(count-3);j=j+1){
          for(k= -1;k<NPTS-2;k++){
            for(i=0;i<4;i++){
            getslice(i+j);
            for(l=0;l<4;l++){
            m=k+l;
            if(m>=NPTS){
                tempxf[i][3-l]= -DPOLY[m-2][0];
                tempzf[i][3-l]= -DPOLY[m-2][1];
                }
            else if(m<0){
                tempxf[i][3-l]= -DPOLY[-m][0];
                tempzf[i][3-l]= -DPOLY[-m][1];
                }
            else {
                tempxf[i][3-l]= DPOLY[m][0];
                tempzf[i][3-l]= -DPOLY[m][1];
                }
            tempyf[i][3-l]=DPOLY[NPTS][0];
            } /* end of for l */
```

```
        } /* end of for i */
       pstore();
       } /* end of for k */
       } /* end of for j */
/* now to make patch of mirror image */
 if(mflg==1){
       for(j=0;j<(count-3);j=j+1){
         for(k= -1;k<(NPTS-2);k++){
           for(i=0;i<4;i++){
           getslice(i+j);
           for(l=0;l<4;l++){
           m=k+l;
           if(m>=NPTS){
                tempxf[i][l]= DPOLY[m-2][0];
                tempzf[i][l]= -DPOLY[m-2][1];
                }
           else if(m<0){
                tempxf[i][l]= DPOLY[-m][0];
                tempzf[i][l]= -DPOLY[-m][1];
                }
           else {
                tempxf[i][l]= -DPOLY[m][0];
                tempzf[i][l]= -DPOLY[m][1];
                }
           tempyf[i][l]=DPOLY[NPTS][0];
           }
         }
        pstore();
        }
        }
  }
 }
  /* here to calculate volume */


  vol()
  {
        int i;
        float volume,area;
```

```
        volume=0;
        area=0;
        for(i=(int)set1;i<=(int)set2;i++){
        getpoly((float)i);
        bezplot(1);
        volume=volume + measblock[3];
        area=area+measblock[2];
        }
        measblock[4] = volume/10;/* volume in cu cm */
        measblock[5] = area/100;
        pmeas(measblock);
}
/* here to plot a patch */

ptchplt(count)
        int count;
{
        defbasis(CARDINAL,cardinal);
        patchcurves(2,2);
        patchprecision(10,10);
        patchbasis(CARDINAL,CARDINAL);
        c3s(blackvec);

        pget(count);

        patch(geomxfr,geomyfr,geomzfr);
        patch(geomxfl,geomyfl,geomzfl);
}
/* here to store a patch in memory */

pstore()
{
        int i,j;

        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
         patches[patchcount][0][i][j]=tempxf[i][j];
```

```
        patches[patchcount][1][i][j]=tempyf[i][j];
        patches[patchcount][2][i][j]=tempzf[i][j];
        }
      }
      patchcount++;
}
/* here to print patch x,y,z matrices */

geomprnt()
{
      int i;

      printf("geomx\n");
      for(i=0;i<4;i++)
      printf("%4.2f %4.2f %4.2f %4.2f\n",geomxfr[i][0],
            geomxfr[i][1],geomxfr[i][2],geomxfr[i][3]);
      printf("geomyy\n");
      for(i=0;i<4;i++)
      printf("%4.2f %4.2f %4.2f %4.2f\n",geomyfr[i][0],
            geomyfr[i][1],geomyfr[i][2],geomyfr[i][3]);
      printf("geomz\n");
      for(i=0;i<4;i++)
      printf("%4.2f %4.2f %4.2f %4.2f\n",geomzfr[i][0],
            geomzfr[i][1],geomzfr[i][2],geomzfr[i][3]);
}
/* here to get a patch from memory */

pget(count)
      int count;
{
      int i,j;

      if(count<0 || count>(PATCHMAX-2))return;
      for(i=0;i<4;i++){
        for(j=0;j<4;j++){
       geomxfr[i][j]=patches[count][0][i][j];
       geomyfr[i][j]=patches[count][1][i][j];
       geomzfr[i][j]=patches[count][2][i][j];
```

```
        }
       }

       matmult(wscalematrix,geomxfr,geomxfl);
       matmult(wscalematrix,geomyfr,geomyfl);
       matmult(wscalematrix,geomzfr,geomzfl);
 if(mflg==1){
       matmult(reflect,geomxfl,geomxfr);
       matmult(trans,geomyfl,geomyfr);
       matmult(trans,geomzfl,geomzfr);
 }
}
/* here to clear patches[][][][] */

clrpatches()
{
       int i,j,k,l;

       for(i=0;i<PATCHMAX;i++){
         for(j=0;j<3;j++){
          for(k=0;k<4;k++){
           for(l=0;l<4;l++){
           patches[i][j][k][l]=0;
           }
          }
         }
       }
}
```

/* lass6.c */

```
#include "device.h"
#include "lassdefs"
#include <gl.h>
#include <math.h>

       extern short NDATA[NARRAY];
       extern short redvec[3];
```

```
        extern float datablock[20];
        extern float measblock[10];
        extern float deltax,deltay;
        extern int obdatacnt;
        double TAPE[STEPS+1][2];
      float obdata[STEPS*NFRAME][2];



/* here to calculate a linear tape measure distance */


lintape(d1,d2,plot)
        float d1,d2;

{
        int i,dd1,dd2,dd11,dd22,tcnt;
        int inc,flg,temp;
        float poly[3],r0,r1,r2,rr,eta,theta,phi;
        float distnce,fd1,fd2,lxinc,lyinc,hxinc,hyinc;
        float px,py,xx,yy,ftemp,artan();

        poly[2]=0;
        dd1 = (int)d1;
        dd2 = (int)d2;
        fd1 = d1-dd1;
        fd2 = d2-dd2;
        if(dd1<0)dd1=obdatacnt+dd1;
        if(dd1>=obdatacnt)dd1=dd1-obdatacnt;
        if(dd2<0)dd2=obdatacnt+dd2;
        if(dd2>=obdatacnt)dd2=dd2-obdatacnt;
        if(dd2<dd1){
        temp=dd2;
        ftemp=fd2;
        dd2=dd1;
        fd2=fd1;
        dd1=temp;
        fd1=ftemp;
        inc = -1;
```

```
}
else inc = 1;
dd11=dd1+1;
dd22=dd2+1;
if(dd11==obdatacnt)dd11=0;
if(dd22==obdatacnt)dd22=0;
lxinc=(obdata[dd11][0]-obdata[dd1][0])*fd1;
lyinc=(obdata[dd11][1]-obdata[dd1][1])*fd1;
hxinc=(obdata[dd22][0]-obdata[dd2][0])*fd2;
hyinc=(obdata[dd22][1]-obdata[dd2][1])*fd2;
dd11=dd1+inc;
dd22=dd2+inc;
if(dd11==obdatacnt)dd11=0;
if(dd22==obdatacnt)dd22=0;
if(dd11<0)dd11=obdatacnt+dd11;
if(dd22<0)dd22=obdatacnt+dd22;
tcnt=1;
TAPE[0][0]=obdata[dd1][0]+lxinc;
TAPE[0][1]=obdata[dd1][1]+lyinc;
TAPE[1][0]=obdata[dd11][0];
TAPE[1][1]=obdata[dd11][1];
for(i=dd11+inc;;i=i+inc){
if(i>=obdatacnt)i=i-obdatacnt;
if(i<0)i=obdatacnt+i;
if((inc== 1 && (i>=dd2 | | i<=dd1)) | | (inc== -1
&& i>=dd1 && i<=dd2)){
r2=fhypot((obdata[i][0]+hxinc),(obdata[i][1]+hyinc));
                                        /* length of latest vector */
phi=artan((obdata[i][1]+hyinc),(obdata[i][0]+hxinc));
flg=1;
}
else{
r2=fhypot(obdata[i][0], obdata[i][1]);
                                        /* length of latest vector */
phi=artan(obdata[i][1],obdata[i][0]);
flg=0;
}
```

```
 do{
  r0=fhypot(TAPE[tcnt-1][0],TAPE[tcnt-1][1]);
                        /* length of last -1 stored vector */
  eta=artan(TAPE[tcnt-1][1],TAPE[tcnt-1][0]);
                        /* angle of last -1  stored vector */
  r1=fhypot(TAPE[tcnt][0],TAPE[tcnt][1]);
                        /* length of last stored vector */
  theta=artan(TAPE[tcnt][1],TAPE[tcnt][0]);
   rr=(r2*r0*sin(phi-eta))/(r0*sin(theta-eta)+
                                    r2*sin(phi-theta));
  if(rr>=r1)tcnt--;
 }while(rr>r1 && tcnt>0);
 if(tcnt<0)tcnt=obdatacnt+tcnt;
 tcnt++;
 if(tcnt>=obdatacnt)tcnt=tcnt-obdatacnt;
 if(flg==1){
 TAPE[tcnt][0]=obdata[i][0]+hxinc;
 TAPE[tcnt][1]=obdata[i][1]+hyinc;
 break;
 }
 else {
 TAPE[tcnt][0]=obdata[i][0];
 TAPE[tcnt][1]=obdata[i][1];
 }
 }
 c3s(redvec);
 bgnline();
 px=TAPE[0][0];
 py=TAPE[0][1];
 if(plot==1){
 poly[0]=TAPE[0][0]/SCALE;
 poly[1]=TAPE[0][1]/SCALE;
 v3f(poly);
 }
 distnce=0;
 for(i=1;i<=tcnt;i++){
 if(plot==1){
 poly[0]=TAPE[i][0]/SCALE;
```

```
        poly[1]=TAPE[i][1]/SCALE;
        v3f(poly);
        }
        xx=TAPE[i][0];
        yy=TAPE[i][1];
        distnce = distnce + sqrt(((xx-px)*(xx-px))+((yy-py)*
                                                    (yy-py)));
        px=xx;
        py=yy;
        }
        endline();
        if(tcnt<3)measblock[1]=0;
        else measblock[1] = distnce;
        pmeas(measblock);
}
/* here to resolve atan calculations */

float artan(yy,xx)
        float yy,xx;
{
        float phi,pi2;

        pi2=2*PI;
        if(xx>0)phi=fatan(yy/xx);
        else if(xx<0)phi=PI+fatan(yy/xx);
        else if(xx==0 && yy>0)phi= PI/2;
        else if(xx==0 && yy<0)phi= 3*PI/2;
        else if(xx==0 && yy==0)phi= 0;
        if(phi<0)phi=pi2+phi;
        if(phi>pi2)phi=phi-pi2;
        return(phi);
}
```

/* lass7.c */

```
#include "device.h"
#include "lassdefs"
#include <gl.h>
```

```
#include <math.h>


        extern float zoom;
        extern int rotx,roty,rotz,deltax,deltay,kbdflg;
        extern float irismat[4][4];
        extern short blackvec[3];
        extern short greenvec[3];
        extern short whitevec[3];
        extern short redvec[3];
        extern short bluevec[3];
        extern short ltbluevec[3];
        extern short yellowvec[3];
        extern short pinkvec[3];
        extern short purplevec[3];
        extern short greyvec[3];
        extern long keywin;
        extern float datablock[20];
        extern float transx,transy,transz;
        extern float DPOLY[NPTS+1][2];


rotax(mat)
        int mat;
{
        scale(zoom,zoom,zoom);
        rotate((int)roty, 'y');
        rotate((int)rotz, 'z');
        rotate((int)rotx, 'x');
        translate(transx,transy,transz);
        if(mat !=0)getmatrix(irismat);
}


keyboard()
{
        winset(keywin);
        c3s(whitevec);
        clear();
```

```
        c3s(blackvec);
    drawkey(0,0,16," Pclear");
    drawkey(0,1,15," Psave");
    drawkey(0,0,15," Pload");
    drawkey(0,1,14,"Setzero");
    drawkey(0,0,14," F/B");
    drawkey(0,1,13," Marks");
    drawkey(0,0,13," Arms");
    drawkey(0,1,12,"");
    drawkey(0,0,12," Ssave");
    drawkey(5,1,11," Set 2");
    drawkey(5,0,11," Set 1");
    drawkey(0,1,10," Y2set");
    drawkey(0,0,10," Y1set");
    drawkey(0,1,9," X2set");
    drawkey(0,0,9," X1set");
    drawkey(0,1,8,"");
    drawkey(5,0,8," Setdy");
    drawkey(0,1,7,"");
    drawkey(0,0,7,"");
    drawkey(0,1,6," Skin");
    drawkey(0,0,6," Lines");
    drawkey(0,1,5,"Trans y");
    drawkey(0,0,5,"Trans x");
    drawkey(1,1,4,"  File");
    drawkey(0,0,4,"Trans z");
    drawkey(0,1,3," Rot y");
    drawkey(0,0,3," Rot x");
    drawkey(2,1,2," Curve");
    drawkey(0,0,2," Rot z");
    drawkey(0,1,1," Lpp");
    drawkey(0,0,1," Zoom");
    drawkey(0,1,0," Reset");
    drawkey(3,0,0," Quit");
}

keybd2()
{
```

```
        winset(keywin);
        c3s(whitevec);
        clear();
        c3s(blackvec);
    drawkey(0,0,16,"");
    drawkey(0,1,15,"");
    drawkey(0,0,15,"");
    drawkey(0,1,14,"");
    drawkey(0,0,14,"");
    drawkey(0,1,13,"");
    drawkey(0,0,13,"");
    drawkey(0,1,12,"");
    drawkey(0,0,12,"");
    drawkey(0,1,11,"");
    drawkey(0,0,11,"");
    drawkey(0,1,10,"");
    drawkey(0,0,10,"");
    drawkey(0,1,9,"");
    drawkey(0,0,9,"");
    drawkey(0,1,8,"");
    drawkey(0,0,8,"");
    drawkey(0,1,7,"Trans-z");
    drawkey(0,0,7,"Trans+z");
    drawkey(0,1,6,"Trans-y");
    drawkey(0,0,6,"Trans+y");
    drawkey(0,1,5,"Trans-x");
    drawkey(0,0,5,"Trans+x");
    drawkey(0,1,4," Rot-z");
    drawkey(0,0,4," Rot+z");
    drawkey(0,1,3," Rot-y");
    drawkey(0,0,3," Rot+y");
    drawkey(0,1,2," Rot-x");
    drawkey(0,0,2," Rot+x");
    drawkey(0,1,1,"Zoom-");
    drawkey(0,0,1,"Zoom+");
    drawkey(0,1,0," Reset");
    drawkey(4,0,0," Iris");
}
```

```
keybd3a()
{
        winset(keywin);
        c3s(whitevec);
        clear();
        c3s(blackvec);
    drawkey(0,1,19," Patch");
    drawkey(0,1,18,"");
    drawkey(0,0,18," Tglzro");
    drawkey(0,1,17,"Setzero");
    drawkey(0,0,17," Smes");
    drawkey(0,1,16," Dat 2");
    drawkey(0,0,16," Dat 1");
    drawkey(0,1,15," Oblique");
    drawkey(0,0,15," Vol.");
    drawkey(0,1,14," Spline");
    drawkey(0,0,14," Lines");
    drawkey(0,1,13," Getsl");
    drawkey(0,0,13," Rot y");
    drawkey(0,1,12,"Trans y");
    drawkey(0,0,12,"Trans x");
    drawkey(5,1,11," Save");
    drawkey(1,0,11," Recall");
    drawkey(0,1,10," Mirror");
    drawkey(0,0,10," Zflg");
    drawkey(0,1,9,"Cntdn");
    drawkey(0,0,9,"Cntup");
    drawkey(0,1,8," Mload");
    drawkey(0,0,8," Setu");
    drawkey(0,1,7," Regen");
    drawkey(0,0,7,"Shocnt");
    drawkey(0,1,6,"Getcnt");
    drawkey(0,0,6,"Savcnt");
    drawkey(0,1,5," Tcirc");
    drawkey(0,0,5,"Tape");
    drawkey(0,1,4,"");
    drawkey(0,0,4,"");
    drawkey(0,1,3,"");
```

```
    drawkey(0,0,3,"");
    drawkey(0,1,2,"");
    drawkey(0,0,2," Redo");
    drawkey(0,1,1," Pclear");
    drawkey(4,0,1," Iris");
    drawkey(0,1,0," Xyzero");
    drawkey(2,0,0," Keys");
}

keybd3b()
{
        winset(keywin);
        c3s(whitevec);
        clear();
        c3s(blackvec);
  drawkey(0,1,19," Patch");
  drawkey(5,1,18," InP");
  drawkey(5,0,18," Tglzro");
  drawkey(0,1,17,"");
  drawkey(0,0,17," Rev.");
  drawkey(0,1,16," Elipse");
  drawkey(5,0,16," Angle");
  drawkey(0,1,15,"Trans y");
  drawkey(5,0,15,"Trans x");
  drawkey(0,1,14," Cntdn");
  drawkey(0,0,14," Cntup");
  drawkey(0,1,13," Getsl");
  drawkey(5,0,13," Rot y");
  drawkey(0,1,12," Up");
  drawkey(0,0,12," Down");
  drawkey(5,1,11," Updy");
  drawkey(0,0,11," Fit");
 drawkey(0,1,10," Dpy15");
 drawkey(0,0,10," Dpx15");
 drawkey(0,1,9," Dp14");
 drawkey(0,0,9," Dp13");
 drawkey(0,1,8," Dp12");
 drawkey(0,0,8," Dp11");
```

```
    drawkey(0,1,7," Dp10");
    drawkey(0,0,7," Dp9");
    drawkey(0,1,6," Dp8");
    drawkey(0,0,6," Dp7");
    drawkey(0,1,5," Dp6");
    drawkey(0,0,5," Dp5");
    drawkey(0,1,4," Dp4");
    drawkey(0,0,4," Dp3");
    drawkey(0,1,3," Dp2");
    drawkey(0,0,3," Dp1");
    drawkey(0,1,2," Dpy0");
    drawkey(0,0,2," Dpx0");
    drawkey(0,1,1," Pclear");
    drawkey(4,0,1," Iris");
    drawkey(0,1,0," Xyzero");
    drawkey(2,0,0," Keys");
}

drawkey(cc,xx,yy,title)
        int xx,yy,cc;
        char title[20];
{
        float zz;

        winset(keywin);
        zz=(float)yy/2;
        switch(cc){
        case 0:
        c3s(greenvec);
        break;
        case 1:
        c3s(ltbluevec);
        break;
        case 2:
        c3s(yellowvec);
        break;
        case 3:
        c3s(redvec);
```

```
        break;
        case 4:
        c3s(pinkvec);
        break;
        case 5:
        c3s(purplevec);
        break;
        default:
        break;
        }
        rectf(xx,zz,xx+1,zz+0.5);
        c3s(blackvec);
        rect(xx,zz,xx+1,zz+0.5);
        cmov2(xx,zz+0.2);
        charstr(title);
}

readkey()
{
        int xmouse, ymouse, retval;
        float step,charoff;

        winset(keywin);
        step=(float)(YMAXSCREEN-70)*3/80;
        retval=0;
        charoff=9.5;
        ymouse = getvaluator(MOUSEY);
        xmouse = getvaluator(MOUSEX);
        if(xmouse<XMAXSCREEN/20){ /* left side */
          if(0<ymouse && ymouse<step){
          retval=EXITTRANS;
        keyboard();
        cmov2(0,charoff);
        charstr("Exit");
         }
          if(step<ymouse && ymouse<2*step){
          retval=ZOOM;
        keyboard();
```

```
 cmov2(0,charoff);
 charstr("Zoom");
  }
   if(2*step<ymouse && ymouse<3*step){
   retval=ROTZ;
 keyboard();
 cmov2(0,charoff);
 charstr("Rot z");
  }
   if(3*step<ymouse && ymouse<4*step){
   retval=ROTX;
 keyboard();
 cmov2(0,charoff);
 charstr("Rot x");
  }
   if(4*step<ymouse && ymouse<5*step){
   retval=TRANSZ;
 keyboard();
 cmov2(0,charoff);
 charstr("Trans z");
  }
   if(5*step<ymouse && ymouse<6*step){
   retval=TRANSX;
 keyboard();
 cmov2(0,charoff);
 charstr("Trans x");
  }
   if(6*step<ymouse && ymouse<7*step){
   retval=LINES;
 keyboard();
 cmov2(0,charoff);
 charstr("Lines");
  }
   if(7*step<ymouse && ymouse<8*step){
   retval=BLANK;
 keyboard();
 cmov2(0,charoff);
 charstr("");
```

```
  }
   if(8*step<ymouse && ymouse<9*step){
   retval=LINESTEP;
 keyboard();
 cmov2(0,charoff);
 charstr(" Setdx");
  }
   if(9*step<ymouse && ymouse<10*step){
   retval=X1SET;
 keyboard();
 cmov2(0,charoff);
 charstr("X1set");
  }
   if(10*step<ymouse && ymouse<11*step){
   retval=Y1SET;
 keyboard();
 cmov2(0,charoff);
 charstr("Y1set");
  }
   if(11*step<ymouse && ymouse<12*step){
   retval= SET1;
 keyboard();
 cmov2(0,charoff);
 charstr("Set 1");
  }
   if(12*step<ymouse && ymouse<13*step){
   retval=SSAVE;
 keyboard();
 cmov2(0,charoff);
 charstr("Ssave");
  }
   if(13*step<ymouse && ymouse<14*step){
   retval=ARMS;
 keyboard();
 cmov2(0,charoff);
 charstr("Arms");
  }
   if(14*step<ymouse && ymouse<15*step){
```

```
  retval=FB;
 keyboard();
 cmov2(0,charoff);
 charstr("F/B");
  }
   if(15*step<ymouse && ymouse<16*step){
  retval=LPLOAD;
 keyboard();
 cmov2(0,charoff);
 charstr(" Pload");
  }
   if(16*step<ymouse && ymouse<17*step){
  retval=LCLEAR;
 keyboard();
 cmov2(0,charoff);
 charstr(" Pclear");
  }
 }
 if(xmouse>XMAXSCREEN/20 &&
 xmouse<XMAXSCREEN/10){ /* right side */
   if(0<ymouse && ymouse<step){
  retval=RESET;
 keyboard();
 cmov2(0,charoff);
 charstr("Reset");
  }
   if(step<ymouse && ymouse<2*step){
  retval=LPP;
 keyboard();
 cmov2(0,charoff);
 charstr(" Lpp");
  }
   if(2*step<ymouse && ymouse<3*step){
  retval=CURVE;
 keyboard();
 cmov2(0,charoff);
 charstr("Curve");
  }
```

```
    if(3*step<ymouse && ymouse<4*step){
    retval=ROTY;
   keyboard();
   cmov2(0,charoff);
   charstr("Rot y");
    }
    if(4*step<ymouse && ymouse<5*step){
    retval=XFILE;
   keyboard();
   cmov2(0,charoff);
   charstr("File");
    }
    if(5*step<ymouse && ymouse<6*step){
    retval=TRANSY;
   keyboard();
   cmov2(0,charoff);
   charstr("Trans y");
    }
    if(6*step<ymouse && ymouse<7*step){
    retval=SKIN;
   keyboard();
   cmov2(0,charoff);
   charstr("Skin");
    }
    if(7*step<ymouse && ymouse<8*step){
    retval=BLANK;
   keyboard();
   cmov2(0,charoff);
   charstr("");
    }
    if(8*step<ymouse && ymouse<9*step){
    retval=BLANK;
   keyboard();
   cmov2(0,charoff);
   charstr("");
    }
    if(9*step<ymouse && ymouse<10*step){
    retval=X2SET;
```

```
keyboard();
cmov2(0,charoff);
charstr("X2set");
 }
  if(10*step<ymouse && ymouse<11*step){
  retval=Y2SET;
keyboard();
cmov2(0,charoff);
charstr("Y2set");
 }
  if(11*step<ymouse && ymouse<12*step){
  retval=SET2;
keyboard();
cmov2(0,charoff);
charstr("Set 2");
 }
  if(12*step<ymouse && ymouse<13*step){
  retval=BLANK;
keyboard();
cmov2(0,charoff);
charstr("");
 }
  if(13*step<ymouse && ymouse<14*step){
  retval=MARKS;
keyboard();
cmov2(0,charoff);
charstr("Marks");
 }
  if(14*step<ymouse && ymouse<15*step){
  retval=SETZERO;
keyboard();
cmov2(0,charoff);
charstr("Setzero");
 }
  if(15*step<ymouse && ymouse<16*step){
  retval=LPSAVE;
keyboard();
cmov2(0,charoff);
```

```
        charstr(" Psave");
         }
        }
        swapbuffers();
        return(retval);
}

rdkey2()
{
        int xmouse, ymouse, retval;
        float step,charoff;

        winset(keywin);
        step=(float)(YMAXSCREEN-70)*3/80;
        retval=0;
        charoff=9.5;
        ymouse = getvaluator(MOUSEY);
        xmouse = getvaluator(MOUSEX);
        if(xmouse<XMAXSCREEN/20){ /* left side */
          if(0<ymouse && ymouse<step){
          retval=IRIS;
        keybd2();
        cmov2(0,charoff);
        charstr("Iris");
         }
          if(step<ymouse && ymouse<2*step){
         retval=ZOOMP;
        keybd2();
        cmov2(0,charoff);
        charstr("Zoom+");
         }
          if(2*step<ymouse && ymouse<3*step){
         retval=ROTPX;
        keybd2();
        cmov2(0,charoff);
        charstr("Rot+x");
         }
          if(3*step<ymouse && ymouse<4*step){
```

```
  retval=ROTPY;
 keybd2();
 cmov2(0,charoff);
 charstr("Rot+y");
  }
   if(4*step<ymouse && ymouse<5*step){
   retval=ROTPZ;
 keybd2();
 cmov2(0,charoff);
 charstr("Rot+z");
  }
   if(5*step<ymouse && ymouse<6*step){
   retval=TRANSPX;
 keybd2();
 cmov2(0,charoff);
 charstr("Trans+x");
  }
   if(6*step<ymouse && ymouse<7*step){
   retval=TRANSPY;
 keybd2();
 cmov2(0,charoff);
 charstr("Trans+y");
  }
   if(7*step<ymouse && ymouse<8*step){
   retval=TRANSPZ;
 keybd2();
 cmov2(0,charoff);
 charstr("Trans+z");
  }
   if(8*step<ymouse && ymouse<9*step){
   retval=BLANK;
 keybd2();
 cmov2(0,charoff);
 charstr("");
  }
   if(9*step<ymouse && ymouse<10*step){
   retval=BLANK;
 keybd2();
```

```
 cmov2(0,charoff);
 charstr("");
  }
   if(10*step<ymouse && ymouse<11*step){
   retval=BLANK;
 keybd2();
 cmov2(0,charoff);
 charstr("");
  }
   if(11*step<ymouse && ymouse<12*step){
   retval=BLANK;
 keybd2();
 cmov2(0,charoff);
 charstr("");
  }
   if(12*step<ymouse && ymouse<13*step){
   retval=BLANK;
 keybd2();
 cmov2(0,charoff);
 charstr("");
  }
   if(13*step<ymouse && ymouse<14*step){
   retval=BLANK;
 keybd2();
 cmov2(0,charoff);
 charstr("");
  }
   if(14*step<ymouse && ymouse<15*step){
   retval=BLANK;
 keybd2();
 cmov2(0,charoff);
 charstr("");
  }
   if(15*step<ymouse && ymouse<16*step){
   retval=BLANK;
 keybd2();
 cmov2(0,charoff);
 charstr("");
```

```
 }
  if(16*step<ymouse && ymouse<17*step){
  retval=BLANK;
keybd2();
cmov2(0,charoff);
charstr("");
 }
}
if(xmouse>XMAXSCREEN/20
 && xmouse<XMAXSCREEN/10){ /* right side */
  if(0<ymouse && ymouse<step){
  retval=RESET;
keybd2();
cmov2(0,charoff);
charstr("Reset");
 }
  if(step<ymouse && ymouse<2*step){
  retval=ZOOMM;
keybd2();
cmov2(0,charoff);
charstr("Zoom-");
 }
  if(2*step<ymouse && ymouse<3*step){
  retval=ROTMX;
keybd2();
cmov2(0,charoff);
charstr("Rot-x");
 }
  if(3*step<ymouse && ymouse<4*step){
  retval=ROTMY;
keybd2();
cmov2(0,charoff);
charstr("Rot-y");
 }
  if(4*step<ymouse && ymouse<5*step){
  retval=ROTMZ;
keybd2();
cmov2(0,charoff);
```

```
 charstr("Rot-z");
  }
   if(5*step<ymouse && ymouse<6*step){
   retval=TRANSMX;
 keybd2();
 cmov2(0,charoff);
 charstr("Trans-x");
  }
   if(6*step<ymouse && ymouse<7*step){
   retval=TRANSMY;
 keybd2();
 cmov2(0,charoff);
 charstr("Trans-y");
  }
   if(7*step<ymouse && ymouse<8*step){
   retval=TRANSMZ;
 keybd2();
 cmov2(0,charoff);
 charstr("Trans-z");
  }
   if(8*step<ymouse && ymouse<9*step){
   retval=BLANK;
 keybd2();
 cmov2(0,charoff);
 charstr("");
  }
   if(9*step<ymouse && ymouse<10*step){
   retval=BLANK;
 keybd2();
 cmov2(0,charoff);
 charstr("");
  }
   if(10*step<ymouse && ymouse<11*step){
   retval=BLANK;
 keybd2();
 cmov2(0,charoff);
 charstr("");
  }
```

```
        if(11*step<ymouse && ymouse<12*step){
        retval=BLANK;
      keybd2();
      cmov2(0,charoff);
      charstr("");
       }
        if(12*step<ymouse && ymouse<13*step){
        retval=BLANK;
      keybd2();
      cmov2(0,charoff);
      charstr("");
       }
        if(13*step<ymouse && ymouse<14*step){
        retval=BLANK;
      keybd2();
      cmov2(0,charoff);
      charstr("");
       }
        if(14*step<ymouse && ymouse<15*step){
        retval=BLANK;
      keybd2();
      cmov2(0,charoff);
      charstr("");
       }
        if(15*step<ymouse && ymouse<16*step){
        retval=BLANK;
      keybd2();
      cmov2(0,charoff);
      charstr("");
       }
      }
      swapbuffers();
      return(retval);
}


rdkey3a()
{
```

```
 int xmouse, ymouse, retval;
 float step,charoff;

 winset(keywin);
 step=(float)(YMAXSCREEN-70)*3/80;
 retval=0;
 charoff=9.75;
 ymouse = getvaluator(MOUSEY);
 xmouse = getvaluator(MOUSEX);
 if(xmouse<XMAXSCREEN/20){ /* left side */
  if(0<ymouse && ymouse<step){
  retval=KEYS2;
 keybd3b();
  }
  if(step<ymouse && ymouse<2*step){
  retval=IRIS2;
 keyboard();
 cmov2(0,charoff);
 charstr("Iris");
  }
  if(2*step<ymouse && ymouse<3*step){
  retval=REDO;
 keybd3a();
 cmov2(0,charoff);
 charstr("Redo");
  }
  if(3*step<ymouse && ymouse<4*step){
  retval=BLANK;
 keybd3a();
 cmov2(0,charoff);
 charstr("");
  }
  if(4*step<ymouse && ymouse<5*step){
  retval=BLANK;
 keybd3a();
 cmov2(0,charoff);
 charstr("");
  }
```

```
  if(5*step<ymouse && ymouse<6*step){
  retval=TAPEFLG;
keybd3a();
cmov2(0,charoff);
charstr("Tape");
 }
  if(6*step<ymouse && ymouse<7*step){
  retval=SAVCNT;
keybd3a();
cmov2(0,charoff);
charstr("Savcnt");
 }
  if(7*step<ymouse && ymouse<8*step){
  retval=SHOCNT;
keybd3a();
 }
  if(8*step<ymouse && ymouse<9*step){
  retval=SETU;
keybd3a();
cmov2(0,charoff);
charstr("Setu");
 }
  if(9*step<ymouse && ymouse<10*step){
  retval=CNTUP;
keybd3a();
 }
  if(10*step<ymouse && ymouse<11*step){
  retval=ZFLG;
keybd3a();
 }
  if(11*step<ymouse && ymouse<12*step){
  retval=RECALL;
keybd3a();
cmov2(0,charoff);
charstr("Recall");
 }
  if(12*step<ymouse && ymouse<13*step){
  retval=DELTAX;
```

```
 keybd3a();
 cmov2(0,charoff);
 charstr("Trans x");
  }
   if(13*step<ymouse && ymouse<14*step){
   retval=ROTY;
 keybd3a();
 cmov2(0,charoff);
 charstr(" Rot y");
  }
   if(14*step<ymouse && ymouse<15*step){
   retval=LINES;
 keyboard();
  }
   if(15*step<ymouse && ymouse<16*step){
   retval=VOLUME;
 keybd3a();
 cmov2(0,charoff);
 charstr("Vol.");
  }
   if(16*step<ymouse && ymouse<17*step){
   retval=DAT1;
 keybd3a();
 cmov2(0,charoff);
 charstr("Dat 1");
  }
   if(17*step<ymouse && ymouse<18*step){
   retval=SMES;
 keybd3a();
 cmov2(0,charoff);
 charstr("Smes");
  }
   if(18*step<ymouse && ymouse<19*step){
   retval=TGL;
 keybd3a();
  }
}
if(xmouse>XMAXSCREEN/20
```

```
 && xmouse<XMAXSCREEN/10){ /* right side */
  if(0<ymouse && ymouse<step){
  retval=XYZERO;
keybd3a();
cmov2(0,charoff);
charstr("Xyzero");
 }
  if(step<ymouse && ymouse<2*step){
  retval=PCLEAR;
keybd3a();
cmov2(0,charoff);
charstr("Pclear");
 }
  if(2*step<ymouse && ymouse<3*step){
  retval=BLANK;
keybd3a();
cmov2(0,charoff);
charstr("");
 }
  if(3*step<ymouse && ymouse<4*step){
  retval=BLANK;
keybd3a();
cmov2(0,charoff);
charstr("");
 }
  if(4*step<ymouse && ymouse<5*step){
  retval=BLANK;
keybd3a();
cmov2(0,charoff);
charstr("");
 }
  if(5*step<ymouse && ymouse<6*step){
  retval=TCIRC;
keybd3a();
cmov2(0,charoff);
charstr("Tcirc");
 }
  if(6*step<ymouse && ymouse<7*step){
```

```
  retval=GETCNT;
keybd3a();
 }
  if(7*step<ymouse && ymouse<8*step){
  retval=REGEN;
keybd3a();
cmov2(0,charoff);
charstr("Regen");
 }
  if(8*step<ymouse && ymouse<9*step){
  retval=MLOAD;
keybd3a();
cmov2(0,charoff);
charstr("Mload");
 }
  if(9*step<ymouse && ymouse<10*step){
  retval=CNTDN;
keybd3a();
 }
  if(10*step<ymouse && ymouse<11*step){
  retval=MIRROR;
keybd3a();
cmov2(0,charoff);
charstr("Mirror");
 }
  if(11*step<ymouse && ymouse<12*step){
  retval=SAVE;
keybd3a();
cmov2(0,charoff);
charstr("Save");
 }
  if(12*step<ymouse && ymouse<13*step){
  retval=DELTAY;
keybd3a();
cmov2(0,charoff);
charstr("Trans y");
 }
  if(13*step<ymouse && ymouse<14*step){
```

```
 retval=GETSL;
keybd3a();
cmov2(0,charoff);
charstr("Getsl");
 }
 if(14*step<ymouse && ymouse<15*step){
 retval=SPLINE;
keybd3a();
cmov2(0,charoff);
charstr("Spline");
 }
 if(15*step<ymouse && ymouse<16*step){
 retval=OBLIQUE;
keybd3a();
cmov2(0,charoff);
charstr(" Oblique");
 }
 if(16*step<ymouse && ymouse<17*step){
 retval=DAT2;
keybd3a();
cmov2(0,charoff);
charstr("Dat 2");
 }
 if(17*step<ymouse && ymouse<18*step){
 retval=SETZERO;
keybd3a();
cmov2(0,charoff);
charstr("Setzero");
 }
 if(18*step<ymouse && ymouse<19*step){
 retval=BLANK;
keybd3a();
 }
 if(19*step<ymouse && ymouse<20*step){
 retval=PTCH;
keybd3a();
cmov2(0,charoff);
charstr(" Patch");
```

```
        }
        }
        swapbuffers();
        return(retval);
}

rdkey3b()
{
        int xmouse, ymouse, retval;
        float step,charoff;

        winset(keywin);
        step=(float)(YMAXSCREEN-70)*3/80;
        retval=0;
        charoff=9.75;
        ymouse = getvaluator(MOUSEY);
        xmouse = getvaluator(MOUSEX);
        if(xmouse<XMAXSCREEN/20){ /* left side */
          if(0<ymouse && ymouse<step){
          retval=KEYS1;
        keybd3a();
         }
          if(step<ymouse && ymouse<2*step){
          retval=IRIS2;
        keyboard();
        cmov2(0,charoff);
        charstr("Iris");
         }
          if(2*step<ymouse && ymouse<3*step){
         retval=DPX0;
        keybd3b();
        cmov2(0,charoff);
        charstr("Dpx0");
         }
          if(3*step<ymouse && ymouse<4*step){
         retval=DP1;
        keybd3b();
        cmov2(0,charoff);
```

```
charstr("Dp1");
 }
  if(4*step<ymouse && ymouse<5*step){
  retval=DP3;
keybd3b();
cmov2(0,charoff);
charstr("Dp3");
 }
  if(5*step<ymouse && ymouse<6*step){
  retval=DP5;
keybd3b();
cmov2(0,charoff);
charstr("Dp5");
 }
  if(6*step<ymouse && ymouse<7*step){
  retval=DP7;
keybd3b();
cmov2(0,charoff);
charstr("Dp7");
 }
  if(7*step<ymouse && ymouse<8*step){
  retval=DP9;
keybd3b();
cmov2(0,charoff);
charstr("Dp9");
 }
  if(8*step<ymouse && ymouse<9*step){
  retval=DP11;
keybd3b();
cmov2(0,charoff);
charstr("Dp11");
 }
  if(9*step<ymouse && ymouse<10*step){
  retval=DP13;
keybd3b();
cmov2(0,charoff);
charstr("Dp13");
 }
```

```
  if(10*step<ymouse && ymouse<11*step){
  retval=DPX15;
 keybd3b();
  }
  if(11*step<ymouse && ymouse<12*step){
  retval=FITP;
 keybd3b();
 cmov2(0,charoff);
 charstr(" Fit");
  }
  if(12*step<ymouse && ymouse<13*step){
  retval=DOWN;
 keybd3b();
 cmov2(0,charoff);
 charstr("Down");
  }
  if(13*step<ymouse && ymouse<14*step){
  retval=ROTY;
 keybd3b();
 cmov2(0,charoff);
 charstr(" Rot y");
  }
  if(14*step<ymouse && ymouse<15*step){
  retval=CNTUP;
 keybd3b();
  }
  if(15*step<ymouse && ymouse<16*step){
  retval=DELTAX;
 keybd3b();
 cmov2(0,charoff);
 charstr("Trans x");
  }
  if(16*step<ymouse && ymouse<17*step){
  retval=SPANG;
 keybd3b();
  }
  if(17*step<ymouse && ymouse<18*step){
  retval=REV;
```

```
keybd3b();
cmov2(0,charoff);
charstr("Rev");
 }
  if(18*step<ymouse && ymouse<19*step){
  retval=TGL;
keybd3b();
 }
}
else if(xmouse>XMAXSCREEN/20
 && xmouse<XMAXSCREEN/10){ /* right side */
  if(0<ymouse && ymouse<step){
  retval=XYZERO;
keybd3b();
cmov2(0,charoff);
charstr("Xyzero");
 }
  if(step<ymouse && ymouse<2*step){
  retval=BLANK;
keybd3b();
cmov2(0,charoff);
charstr("");
 }
  if(2*step<ymouse && ymouse<3*step){
  retval=DPY0;
keybd3b();
cmov2(0,charoff);
charstr("Dpy0");
 }
  if(3*step<ymouse && ymouse<4*step){
  retval=DP2;
keybd3b();
cmov2(0,charoff);
charstr("Dp2");
 }
  if(4*step<ymouse && ymouse<5*step){
  retval=DP4;
keybd3b();
```

```
cmov2(0,charoff);
charstr("Dp4");
 }
  if(5*step<ymouse && ymouse<6*step){
  retval=DP6;
keybd3b();
cmov2(0,charoff);
charstr("Dp6");
 }
  if(6*step<ymouse && ymouse<7*step){
  retval=DP8;
keybd3b();
cmov2(0,charoff);
charstr("Dp8");
 }
  if(7*step<ymouse && ymouse<8*step){
  retval=DP10;
keybd3b();
cmov2(0,charoff);
charstr("Dp10");
 }
  if(8*step<ymouse && ymouse<9*step){
  retval=DP12;
keybd3b();
cmov2(0,charoff);
charstr("Dp12");
 }
  if(9*step<ymouse && ymouse<10*step){
  retval=DP14;
keybd3b();
cmov2(0,charoff);
charstr("Dp14");
 }
  if(10*step<ymouse && ymouse<11*step){
  retval=DPY15;
keybd3b();
cmov2(0,charoff);
charstr("Dpy15");
```

```
 }
  if(11*step<ymouse && ymouse<12*step){
  retval=UP5;
keybd3b();
cmov2(0,charoff);
charstr("Updy");
 }
  if(12*step<ymouse && ymouse<13*step){
  retval=UP;
keybd3b();
cmov2(0,charoff);
charstr(" Up");
 }
  if(13*step<ymouse && ymouse<14*step){
  retval=GETSL;
keybd3b();
cmov2(0,charoff);
charstr("Getsl");
 }
  if(14*step<ymouse && ymouse<15*step){
  retval=CNTDN;
keybd3b();
 }
  if(15*step<ymouse && ymouse<16*step){
  retval=DELTAY;
keybd3b();
cmov2(0,charoff);
charstr("Trans y");
 }
  if(16*step<ymouse && ymouse<17*step){
  retval=ELIPSE;
keybd3b();
cmov2(0,charoff);
charstr("Elipse");
 }
  if(17*step<ymouse && ymouse<18*step){
  retval=BLANK;
keybd3b();
```

```
        cmov2(0,charoff);
        charstr("");
         }
          if(18*step<ymouse && ymouse<19*step){
          retval=INP;
        keybd3b();
         }
          if(19*step<ymouse && ymouse<20*step){
          retval=PTCH;
        keybd3b();
        cmov2(0,charoff);
        charstr(" Patch");
         }
        }
        swapbuffers();
        return(retval);}

pickx()
{
   Device val;
        int xmouse, retval;
        float charoff;


        retval=0;
        charoff=9.75;
        qreset();
        if(qread(&val)==LEFTMOUSE){
        if(fpoint(0)==1){
         retval=DPY0;
         keybd3b();
         kbdflg=4;
          cmov2(0,charoff);
          charstr("Dpy0");
         }
       else if(fpoint(1)==1){
         retval=DP1;
         keybd3b();
```

```
  kbdflg=4;
   cmov2(0,charoff);
   charstr("Dp1");
  }
 else if(fpoint(2)==1){
  retval=DP2;
  keybd3b();
  kbdflg=4;
   cmov2(0,charoff);
   charstr("Dp2");
  }
 else if(fpoint(3)==1){
  retval=DP3;
  keybd3b();
  kbdflg=4;
   cmov2(0,charoff);
   charstr("Dp3");
  }
 else if(fpoint(4)==1){
  retval=DP4;
  keybd3b();
  kbdflg=4;
   cmov2(0,charoff);
   charstr("Dp4");
  }
 else if(fpoint(5)==1){
  retval=DP5;
  keybd3b();
  kbdflg=4;
   cmov2(0,charoff);
   charstr("Dp5");
  }
 else if(fpoint(6)==1){
  retval=DP6;
  keybd3b();
  kbdflg=4;
   cmov2(0,charoff);
   charstr("Dp6");
```

```
  }
 else if(fpoint(7)==1){
  retval=DP7;
  keybd3b();
  kbdflg=4;
   cmov2(0,charoff);
   charstr("Dp7");
  }
 else if(fpoint(8)==1){
  retval=DP8;
  keybd3b();
  kbdflg=4;
   cmov2(0,charoff);
   charstr("Dp8");
  }
 else if(fpoint(9)==1){
  retval=DP9;
  keybd3b();
  kbdflg=4;
   cmov2(0,charoff);
   charstr("Dp9");
  }
 else if(fpoint(10)==1){
  retval=DP10;
  keybd3b();
  kbdflg=4;
   cmov2(0,charoff);
   charstr("Dp10");
  }
 else if(fpoint(11)==1){
  retval=DP11;
  keybd3b();
  kbdflg=4;
   cmov2(0,charoff);
   charstr("Dp11");
  }
 else if(fpoint(12)==1){
  retval=DP12;
```

```
        keybd3b();
        kbdflg=4;
        cmov2(0,charoff);
        charstr("Dp12");
        }
      else if(fpoint(13)==1){
        retval=DP13;
        keybd3b();
        kbdflg=4;
        cmov2(0,charoff);
        charstr("Dp13");
        }
      else if(fpoint(14)==1){
        retval=DP14;
        keybd3b();
        kbdflg=4;
        cmov2(0,charoff);
        charstr("Dp14");
        }
      else if(fpoint(15)==1){
        retval=DPY15;
        keybd3b();
        kbdflg=4;
        cmov2(0,charoff);
        charstr("Dpy15");
        }
      }
      else retval=0;
      return(retval);
}

setzero()
{
      datablock[6]= rotx/10;
      datablock[7]= roty/10;
      datablock[8]= rotz/10;
      datablock[9]= transx*SCALE;
      datablock[10]= transy*SCALE;
```

```
        datablock[11]= transz*SCALE;
        datablock[15]= datablock[13];
        datablock[16]= datablock[14];
        printdata(datablock);
}
/*


/ * lass8.c */
 *      - Compiling with DISPLAY_LIST_VERSION defined produces a
version
 *      named "dlnurbs" that uses display lists.
 */
#include "device.h"
#include <gl.h>
#include <sys/types.h>
#include <stdio.h>
#include <gl/gl.h>
#include <gl/device.h>
#include "lassdefs"



        long zfar;

float  idmat[4][4] = {
        {1.0, 0.0, 0.0, 0.0},
        {0.0, 1.0, 0.0, 0.0},
        {0.0, 0.0, 1.0, 0.0},
        {0.0, 0.0, 0.0, 1.0},
        };


        extern float  MINIHEAD[20];
        extern float patches[PATCHMAX][3][4][4];
        extern float transx,transy,transz,zoom;
        extern float datablock[20];
        extern int maxhite, minhite, maxangl, minangl;
```

```
        extern long nurbswin,keywin;
        extern int patchflg,rotyflg;
        extern int rotx,roty,rotz;
        float xx[4][4],yy[4][4],zz[4][4];
        int patchcount,wscale;
        Object torso = 1;
        float bpatches[PATCHMAX][3][4][4];


initskin()
{
    init_windows();
    init_view();
    make_lights();


    set_scene();

    winset(keywin);
    winpop();
    keybd2();

    zoom=1.0;
   roty=roty+1800;
    fillcart(5);
   drawpatches();
}


nplot()
{
    int command;
    float transparam,zoomtemp;
   long dev;
   short val;

        while(patchflg){
        drawobj();
        while(qtest()) {
```

```
    dev=qread(&val);
    switch(dev) {
        case ESCKEY:
        if(val){
        patchflg=0;
        winclose(nurbswin);
        }
          break;
        case WINQUIT:
        patchflg=0;
          gexit();
          dglclose(-1);
          exit(0);
          break;
        case LEFTMOUSE:
switch (command = rdkey2()){
        case 0:
             break;
        case TRANSPY:
              if(val){
        winset(nurbswin);
        transy=transy+1.0;
              }
             break;
        case TRANSPX:
              if(val){
        winset(nurbswin);
        transx=transx+1.0;
              }
            break;
        case TRANSPZ:
              if(val){
        winset(nurbswin);
        transz=transz+1.0;
              }
            break;
        case TRANSMY:
              if(val){
```

```
winset(nurbswin);
transy=transy-1.0;
        }
      break;
 case TRANSMX:
        if(val){
winset(nurbswin);
transx=transx-1.0;
        }
      break;
 case TRANSMZ:
        if(val){
winset(nurbswin);
transz=transz-1.0;
        }
      break;
case ZOOMP:
zoom=zoom+0.2;
if(zoom>10)zoom=10;
      break;
case ZOOMM:
zoom=zoom-0.2;
if(zoom<0.2)zoom=0.2;
      break;
case ROTPX:
        if(val){
winset(nurbswin);
rotx=rotx+100.0;
        }
        break;
case ROTPY:
        if(val){
winset(nurbswin);
roty=roty+100.0;
        }
        break;
case ROTPZ:
        if(val){
```

```
            winset(nurbswin);
            rotz=rotz+100.0;
                    }
                break;
            case ROTMX:
                    if(val){
            winset(nurbswin);
            rotx=rotx-100.0;
                    }
                    break;
            case ROTMY:
                    if(val){
            winset(nurbswin);
            roty=roty-100.0;
                    }
                    break;
            case ROTMZ:
                    if(val){
            winset(nurbswin);
            rotz=rotz-100.0;
                    }
                break;
            case IRIS:
            if(val){
            patchflg=0;
            winclose(nurbswin);
            }
              break;
            case RESET:
            if(val){
        transx=0; transy= MINUSY; transz=0;
        rotx=0; roty=0; rotz=0;
            zoom=1;
            }
              break;
            default:
              break;
            }
```

```
        }
      }
    }
}


init_windows()
/*-------------------------------------------------------
 * Initialize all windows
 *-------------------------------------------------------
 */
{
   foreground();
   prefposition(0, XMAXSCREEN, 0, YMAXSCREEN-30);
   nurbswin = winopen("nurbs");
   wintitle("CARDINAL Surface");
  doublebuffer();
  RGBmode();
   shademodel(GOURAUD);
  gconfig();
  zbuffer( TRUE );

   glcompat(GLC_ZRANGEMAP, 0);
  zfar = getgdesc(GD_ZMAX);

  wscale = SCALE;
}



init_view()
/*-------------------------------------------------------
 * Initialize view and lighting mode
 *-------------------------------------------------------
 */
{
   mmode(MPROJECTION);
```

```
    ortho(MINUSX,PLUSX,MINUSY,PLUSY,PLUSZ,MINUSZ);

    mmode(MVIEWING);
    loadmatrix(idmat);

}


set_scene()
/*-------------------------------------------------------
 * Clear screen
 *-------------------------------------------------------
 */
{
   winset(nurbswin);
   lmbind(MATERIAL, 0);
   RGBcolor(150,150,150);
   lmbind(MATERIAL, 1);
  czclear(0x00407d00, zfar);
}

drawobj()
/*-------------------------------------------------------
 * Draw object torso
 *-------------------------------------------------------
 */
        set_scene();
        pushmatrix();
        rotax(1);
        callobj(torso);
        swapbuffers();
        popmatrix();
}

drawpatches()
/*-------------------------------------------------------
* Draw series of patches
```

```
*-------------------------------------------------
*/
{
        makeobj(torso);
        drawpoly();
        closeobj();
}

make_lights()
/*-------------------------------------------------
 * Define material, light, and model
 *-------------------------------------------------
 */
{
   /* initialize model and light to default */
  lmdef(DEFLMODEL,1,0,0);
  lmdef(DEFLIGHT,1,0,0);
   /* define material #1 */
  {
     static float array[] = {
                EMISSION, 0.0, 0.0, 0.0,
                AMBIENT,  0.0, 0.0, 0.0,
                DIFFUSE,  0.8, 0.7, 0.5,
                SPECULAR,  0.0, 0.0, 0.0,
                SHININESS, 0.0,
                ALPHA, 1.0,
                LMNULL
     };
      lmdef(DEFMATERIAL, 1, sizeof(array), array);
  }
  /* define light source #2 */
  {
     static float lsource2[] = {
                AMBIENT, 0.5, 0.5, 0.5,
                LCOLOR,  0.5, 0.5, 0.5,
                POSITION,  -0.5, 0.2, 5.0, 0,
                SPOTDIRECTION,  -0.5, 0.5, 0.5,
                SPOTLIGHT, 0.0, 60.0,
```

```
            LMNULL
        };
        lmdef(DEFLIGHT, 2, sizeof(lsource2), lsource2);
    }
    /* define light source #3 */
    {
        static float lsource3[] = {
            AMBIENT, 0.5, 0.5, 0.5,
            LCOLOR,  1.0, 0.75, 0.75,
            POSITION,  0.0, 0.0, 1.5, 0.0,
            LMNULL
        };
        lmdef(DEFLIGHT, 3, sizeof(lsource3), lsource3);
    }

    /* turn on lighting */


    lmbind(LIGHT1, 2);
    lmbind(LIGHT2, 3);
    lmbind(LMODEL, 1);
    lmbind(MATERIAL, 1);
}
```

/* lass9.c */

```
#include "device.h"
#include "lassdefs"
#include <gl.h>
#include <math.h>


/* Progs for oblique slice measurement from cardinal splines */

        extern float DPOLY[NPTS+1][2];
        extern Matrix cardinal[];
        extern float obdata[STEPS*NFRAME][2];
        extern float set1,set2,datum1,datum2;
```

```
        extern int obdatacnt,mflg,patchflg;
        extern float temp[4][4];

        extern float irismat[4][4];.
        extern float trans[4][4];
        extern float measblock[NPTS];
        extern float transx,transy,transz;
        extern int rotx,roty,rotz;
        extern short blackvec[3];
        extern short redvec[3];
        extern short greenvec[3];
        extern short bluevec[3];
        extern short whitevec[3];
        extern int maxhite, minhite;
        float apoly[3];
        float cart[NFRAME*VSTEP][STEPS+10][3];
        int cartrowcnt;

/* here to fill obdata array */

fillcart(vstep)
        int vstep;
{
        int i,j,step;


        step=STEPS/2;
        cartrowcnt=0;
        for(i=(int)set1;i<(int)set2;i=i+vstep){
        getpoly((float)i);
        for(j=0;j<step;j++){
        cardret(j,step);
        cart[cartrowcnt][j][0]= apoly[0];
        cart[cartrowcnt][j][1]= (float)i;
        cart[cartrowcnt][j][2]= -apoly[1];

        cart[cartrowcnt][STEPS-j-1][0]= -apoly[0];
        cart[cartrowcnt][STEPS-j-1][1]= (float)i;
```

```
        cart[cartrowcnt][STEPS-j-1][2]= -apoly[1];
        }
        cartrowcnt++;
        }
}
/* here to draw polygons */

drawpoly()
{
        float vert[3],vect1x,vect1y,vect1z,vect2x,vect2y,vect2z;
        float normal[3],denom,scale;
        int i,j,k,l,m,n,o,step;


        if(mflg==1)step=STEPS;
        else step=STEPS/2+1;
        scale=SCALE;
        for(i=0;i<cartrowcnt-3;i++){
        bgnqstrip();
        for(j=0;j<step;j++){
        for(k=1;k>=0;k--){
        l=i+k;
        m=j;
        n=m-1;
        o=m+1;
        if(o>=STEPS)o=o-STEPS;
        if(n<0)n=STEPS+n;
        vect1x=cart[l][o][0]-cart[l][n][0];
        vect1y=cart[l][o][1]-cart[l][n][1];
        vect1z=cart[l][o][2]-cart[l][n][2];
        vect2x=cart[l+1][m][0]-cart[l-1][m][0];
        vect2y=cart[l+1][m][1]-cart[l-1][m][1];
        vect2z=cart[l+1][m][2]-cart[l-1][m][2];
        normal[0]=(vect1y*vect2z)-(vect1z*vect2y);
        normal[1]=(vect1z*vect2x)-(vect1x*vect2z);
        normal[2]=(vect1x*vect2y)-(vect1y*vect2x);

        denom = sqrt(normal[0]*normal[0] +
```

```
                    normal[1]*normal[1] + normal[2]*normal[2]);
        normal[0]=normal[0]/denom;
        normal[1]=normal[1]/denom;
        normal[2]=normal[2]/denom;
        vert[0]=cart[l][m][0]/scale;
        vert[1]=cart[l][m][1]/scale;
        vert[2]=cart[l][m][2]/scale;
        n3f(normal);
        v3f(vert);
        }
        }
        endqstrip();
        }
}
/* here to return a point from a Cardinal curve */

cardret(point,maxpoints)
        int point,maxpoints;
{
        int s,u;
        float frangle[4],temp[4],npoints,fracpnt,interval;


        apoly[0]=0;
        apoly[1]=0;
        apoly[2]=0;

        interval=NPTS-1;
        npoints = (float) maxpoints/interval;
        s= (int)point/npoints;
        fracpnt = (point-s*npoints)/npoints;
        s=s-1;
        frangle[3]=1.0;
        frangle[2]=fracpnt;
        frangle[1]=frangle[2]*frangle[2];
        frangle[0]=frangle[1]*frangle[2];
        colmat(frangle,cardinal,temp);
          for(u=0;u<4;u++){
```

```
        if((u+s)<0){
        apoly[0]=apoly[0]-temp[0]*DPOLY[1][0];
        apoly[1]=apoly[1]+temp[0]*DPOLY[1][1];
        }
        else if ((u+s)>=0 && (u+s)<NPTS ){
        apoly[0]=apoly[0]+temp[u]*DPOLY[u+s][0];
        apoly[1]=apoly[1]+temp[u]*DPOLY[u+s][1];
        }
        else if((u+s)>=NPTS){
        apoly[0]=apoly[0]-temp[3]*DPOLY[NPTS-2][0];
        apoly[1]=apoly[1]+temp[3]*DPOLY[NPTS-2][1];
        }
      } /* end of for u */
}
/* here to scan the surface for x=0 */


obltrig()
{
      int  i, j dens, tdens, xflg;
      float xmean, ymean, ptemp[4], tempp[4], wscale,tol;

      dens=1;
      obdatacnt=0;
      xmean=0;
      ymean=0;

      ptemp[3]=1;
      tol=0.002;
      wscale=SCALE;
      for(j=0;j<(cartrowcnt-dens);j++){
        for(i=0;i<(STEPS-dens);i++){
            ptemp[0]= cart[j][i][0]/wscale;
            ptemp[1]= cart[j][i][1]/wscale;
            ptemp[2]= cart[j][i][2]/wscale;
        colmat(ptemp,irismat,tempp);
        if(tempp[0]> -tol && tempp[0]<tol){
        obdata[obdatacnt][1]=tempp[1]*PLUSY*wscale;
        obdata[obdatacnt][0]=tempp[2]*PLUSX*wscale;
```

```
            obdatacnt ++;
            } /* end of if */
          } /* end of j */
        } /* end of i */
          for(j=0;j<obdatacnt;j++){
                xmean=xmean+obdata[j][0];
                ymean=ymean+obdata[j][1];
                }
        xmean=xmean/obdatacnt;
        ymean=ymean/obdatacnt;
          for(j=0;j<obdatacnt;j++){
         obdata[j][0]= (obdata[j][0]-xmean);
         obdata[j][1]= (obdata[j][1]-ymean);
                }
        if(obdatacnt>(STEPS*NFRAME))obdatacnt=STEPS*
                NFRAME;
        shell(obdata,obdatacnt);
        patchflg=3;
}
/* here to plot oblique slice data */


obref(datum)
        float datum;
{
        float corr,xinc,yinc,datmf;
        float xscale, yscale, wscale;
        int datm,datmm;

        xscale=50;
        yscale=50;
        datm = (int)datum;
        datmf=datum-datm;
        apoly[2]=0;
        if(datm <0)datm = obdatacnt + datm;
        if(datm >=obdatacnt)datm = datm - obdatacnt;
        datmm=datm+1;
        if(datmm==obdatacnt)datmm=0;
        xinc=(obdata[datmm][0]-obdata[datm][0])*datmf;
```

```
        yinc=(obdata[datmm][1]-obdata[datm][1])*datmf;
        bgnline();
        apoly[0] = 0;
        apoly[1] = 0;
        v3f(apoly);
        apoly[0] = (obdata[datm][0]+xinc)/xscale;
        apoly[1] = (obdata[datm][1]+yinc)/yscale;
        v3f(apoly);
        endline();
}
/* here to calculate the distance between datum1 & datum2 */

plotobdata(datum1,datum2)
        float datum1,datum2;
{
        int  j;
        float xscale, yscale, wscale;

        loadmatrix(trans);
        ortho(MINUSX,PLUSX,MINUSY,PLUSY,
                                        MINUSZ,PLUSZ);
        xscale=50;
        yscale=50;
        translate(0,0,0);
        bgnline();
          for(j=0;j<obdatacnt;j++){
                apoly[0]=(obdata[j][0])/xscale;
                apoly[1]=(obdata[j][1])/yscale;
                apoly[2]=0;
        v3f(apoly);
         }
        apoly[0]=obdata[0][0]/xscale;
        apoly[1]=obdata[0][1]/yscale;
        apoly[2]=0;
        v3f(apoly);
        endline();
        c3s(blackvec);
        obref(datum1);
```

```
        c3s(redvec);
        obref(datum2);
        lintape(datum1,datum2,1);
        dist(datum1,datum2);
}
/* here to plot ref lines on oblique slice data */

dist(d1,d2)
        float d1,d2;
{
        int i,dd1,dd2,dd11,dd22,temp,inc,flg;
        float distnce,px,py,xx,yy,fd1,fd2,lxinc,hxinc,lyinc,hyinc,ftemp;


        dd1 = (int)d1;
        dd2 = (int)d2;
        fd1=d1-dd1;
        fd2=d2-dd2;
        if(dd1<0)dd1=obdatacnt+dd1;
        if(dd1>=obdatacnt)dd1=dd1-obdatacnt;
        if(dd2<0)dd2=obdatacnt+dd2;
        if(dd2>=obdatacnt)dd2=dd2-obdatacnt;
        if(dd2<dd1){
        temp=dd2;
        ftemp=fd2;
        dd2=dd1;
        fd2=fd1;
        dd1=temp;
        fd1=ftemp;
        inc = -1;
        }
        else inc = 1;
        dd11=dd1+1;
        dd22=dd2+1;
        if(dd11==obdatacnt)dd11=0;
        if(dd22==obdatacnt)dd22=0;
        lxinc=(obdata[dd11][0]-obdata[dd1][0])*fd1;
        lyinc=(obdata[dd11][1]-obdata[dd1][1])*fd1;
```

```
        hxinc=(obdata[dd22][0]-obdata[dd2][0])*fd2;
        hyinc=(obdata[dd22][1]-obdata[dd2][1])*fd2;
        distnce = 0;
        px = (obdata[dd1][0]+lxinc);
        py = (obdata[dd1][1]+lyinc);
        for(i=dd1+inc;;i=i+inc){
        if(i>=obdatacnt)i=i-obdatacnt;
        if(i<0)i=obdatacnt+i;
        if((inc==1 && (i>=dd2 || i<=dd1)) || (inc== -1
        && i>=dd1  && i<=dd2)){
        xx = obdata[i][0]+hxinc;
        yy = obdata[i][1]+hyinc;
        flg=1;
        }
        else {
        xx = obdata[i][0];
        yy = obdata[i][1];
        flg=0;
        }
        distnce = distnce + fhypot((xx-px),(yy-py));
        if(flg==1)break;
        px = xx;
        py = yy;
        }
        measblock[0] = distnce;
        pmeas(measblock);
}

printobdata()
{
        int i, j;


         for(j=0;j<obdatacnt;j++){
                for(i=0;i<2;i++){
                printf("%f  ",obdata[j][i]);
         }
        printf("  %d\n",j);
```

```
        }
        printf("\n");
}
```

APPENDIX 7

Source code listings of ancilliary programs

/* avrge.c */

```
#include <stdio.h>
#include "device.h"
#include <gl.h>
#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>
#define NPTS 16 /* number of points in curve fit polygon */


        float slices1[50][NPTS+1][2];
        float slices2[50][NPTS+1][2];
        int dcount,spangcnt,filecnt,lflg;
        long textwin;
        short blackvec[3] = { 0, 0, 0};
        short whitevec[3] = {255,255,255};
        char *name;


main()
        {
        inittextwin();
  qdevice(ESCKEY);
  qdevice(KEYBD);
  qdevice(RETKEY);
        dcount=50;
        lflg=1;
        filecnt=0;
         while(lflg==1){
         slicelod();
         addslice();
         filecnt++;
```

```
        }
       }
/* here to load  patches slice data */

slicelod()
{
 FILE *fopen(),*file_ptr;
 char *string1,string2[50],*string3,string4[50],buffer[20];
  int count,i,j,k;
 float xx,yy;

  string1 = "Enter file name ";
 string3 = "r";

      for(i=0;i<50;i++){
      string2[i]='\0';
      string4[i]='\0';
      }
      i=0;
    . while(string1[i] !='\0'){
      string2[i]=string1[i];
      i++;
      }

 name = &string4[0];
      winset(textwin);
      winpop();
      c3s(whitevec);
      clear();
      c3s(blackvec);
 putline(string2,1.0,5.0);
 count = getline(string2,i,50);
 sscanf(&string2[i],"%s",name);
 winpush();
 file_ptr=fopen(name,string3);
      fscanf(file_ptr,"%d",&dcount);
      fscanf(file_ptr,"%d",&spangcnt);
        for(i=0;i<dcount;i++){
```

```
            for(j=0;j<=NPTS;j++){
             for(k=0;k<2;k++){
        fscanf(file_ptr,"%f",&xx);
        slices1[i][j][k]=xx;
          }
         }
        }
        fclose(file_ptr);
}
/* here to save  patches slice data  */

slicesav()
 {
 FILE *fopen(),*file_ptr;
  char *string1,string2[50],*string3,string4[50];
  int count,i,j;

  string1 = "Enter output file name ";
  string3 = "w";

        for(i=0;i<50;i++){
        string2[i]='\0';
        string4[i]='\0';
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }

  name = &string4[0];
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
  putline(string2,1.0,5.0);
  count = getline(string2,i,50);
```

```
  sscanf(&string2[i],"%s",name);
  winpush();

  file_ptr=fopen(name,string3);
       fprintf(file_ptr,"%d",dcount);
       fprintf(file_ptr," %d\n",spangcnt);
         for(i=0;i<dcount;i++){
           for(j=0;j<=NPTS;j++){
          fprintf(file_ptr,"%8.1f%8.1f",slices2[i][j][0],slices2[i][j][1]);
         }
       fprintf(file_ptr,"\n");
         }

       fclose(file_ptr);
}
/* here to get line into s, return length */

int getline(s,cnt,lim)
       char s[];
       int cnt,lim;
{
       Device val;
       long dev;
       int c,flg,i;

       flg=1; i=cnt;
       while(flg==1){
       while(qtest()){
       dev=qread(&val);
       switch(dev){
       case KEYBD:
       if(val=='\b'){
       i--;
       s[i]= '\0';
       c3s(whitevec);
       clear();
       c3s(blackvec);
       } /* end of if val */
```

```
        else {
        s[i++]=val;
        }
        if(val=='\n')flg=0;
        putline(s,1.0,5.0);
        qreset();
        break;
        case ESCKEY:
        lflg = 0;
        divslice();
        slicesav();
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
   return(i);
}
/* here to write a line to a window */

putline(s,x,y)
        char s[];
        Coord x,y;
{
        cmov2(x,y);
        charstr(s);
}

inittextwin()
{
```

```
   prefposition(2*
XMAXSCREEN/5,4*XMAXSCREEN/5,4*YMAXSCREEN/10,5*YMAXS
CREEN/10);
   textwin = winopen("Slice plot.");
       RGBmode();
       shademodel(FLAT);
       ortho2(0,15,0,10);
       foreground();
       gconfig();
}
/* here to clear slice array  */

cslice()
{
       int i,j;


       for(i=0;i<dcount;i++){
         for(j=0;j<=NPTS;j++){
         slices2[i][j][0] = 0.0;
         slices2[i][j][1] = 0.0;
        }
       }
}
/* here to add slices1 array to slices2 array */

addslice()
{
       int i,j;


       for(i=0;i<dcount;i++){
         for(j=0;j<=NPTS;j++){
         slices2[i][j][0] = slices2[i][j][0] + slices1[i][j][0];
         slices2[i][j][1] = slices2[i][j][1] + slices1[i][j][1];
        }
       }
}
```

```
/* here to divide slices2 array by filecnt */

divslice()
{
      int i,j;


      for(i=0;i<dcount;i++){
        for(j=0;j<=NPTS;j++){
        slices2[i][j][0] = slices2[i][j][0]/filecnt;
        slices2[i][j][1] = slices2[i][j][1]/filecnt;
       }
      }
}
```

/* gradsplot3.c */

```
/* Same as gradsplot1.c but with masterplot. */
/* Creates a plotfile for H.P.Plotter from 16 spline points */
/* to give true surface distance from centre front in x direction. */
#include <stdio.h>
#include "device.h"
#include <gl.h>
#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>
#define NPTS 16 /* number of points in curve fit polygon */
#define NSLICES 50 /* maximum number of slices */
#define PI 3.141592654
#define PLOTUNIT 40 /* number of plotter units per mm. */
#define HWIDTH 350 /* half width of surrounding rectangle in mm's.
*/
#define HDEPTH 378 /* half height of surrounding rectangle in mm's.
*/
#define MAXLINES 39 /* maximum number of vertical lines. */

Matrix cardinal = {
      {-0.5, 1.5,-1.5, 0.5},
```

```
        { 1.0,-2.5, 2.0,-0.5},
        {-0.5, 0.0, 0.5, 0.0},
        { 0.0, 1.0, 0.0, 0.0},
        };


        short blackvec[3] = { 0, 0, 0};
        short redvec[3] = { 255, 0, 0};
        short greenvec[3] = { 73, 206, 130};
        short bluevec[3] = { 0, 0, 255};
        short whitevec[3] = {255,255,255};
        long textwin;
        float  MINIHEAD[20];
        float slices[NSLICES][NPTS+1][3];
        float plotfile[NSLICES][MAXLINES];
        float heights[NSLICES];
        float apoly[3];
        int dcount,bytecount,mflg;
        int maxhite, minhite;
        char *name;
        FILE *file_ptr;


main()
        {
        inittextwin();
  qdevice(ESCKEY);
  qdevice(KEYBD);
  qdevice(RETKEY);
        slicelod();
        calcarray();
        sliceplot();
        masterget();
        calcarray();
        masterplot();
        }

inittextwin()
{
```

```
   prefposition(2*
XMAXSCREEN/5,4*XMAXSCREEN/5,4*YMAXSCREEN/10,5*YMAXS
CREEN/10);
   textwin = winopen("Slice plot.");
       RGBmode();
       shademodel(FLAT);
       ortho2(0,15,0,10);
       foreground();
       gconfig();
}
/* here to load  patches slice data (36 line text file)*/

slicelod()
{
 FILE *fopen();
  char *string1,string2[50],*string3,string4[50];
  int count,i,j,k,m,spangcnt;
  float xx;

   string1 = "Enter file name ";
  string3 = "r";

       for(i=0;i<50;i++){
       string2[i]='\0';
       string4[i]='\0';
       }
       i=0;
       while(string1[i] !='\0'){
       string2[i]=string1[i];
       i++;
       }

  name = &string4[0];
       winset(textwin);
       winpop();
       c3s(whitevec);
       clear();
       c3s(blackvec);
```

```
   putline(string2,1.0,5.0);
   count = getline(string2,i,50);
   sscanf(&string2[i],"%s",name);
   winpush();
  file_ptr=fopen(name,string3);
        fscanf(file_ptr,"%d",&dcount);
        fscanf(file_ptr,"%d",&spangcnt);
          for(i=0;i<dcount;i++){
            for(j=0;j<=NPTS;j++){
             for(k=0;k<2;k++){
        fscanf(file_ptr,"%f",&xx);
        slices[i][j][k]=xx;
           } /* end of for k */
         } /* end of for j */
        } /* end of for i */
/* Here to input y value i.e., heights */
        for(m=0;m<dcount;m++){
          for(j=0;j<NPTS;j++){
         heights[m]=slices[m][NPTS][0];
         slices[m][j][2]=heights[m];
         } /* end of for j */
        } /* end of for m */

        fclose(file_ptr);
}


masterget()
{
  FILE *fopen(),*file_ptr2;
   char *string3,*mastername;
  int i,j,k,m,spangcnt;
  float xx;

  string3 = "r";
  mastername = "av34b.xmfac";

file_ptr2=fopen(mastername,string3);
      fscanf(file_ptr2,"%d",&dcount);
```

```
        fscanf(file_ptr2,"%d",&spangcnt);
          for(i=0;i<dcount;i++){
            for(j=0;j<=NPTS;j++){
             for(k=0;k<2;k++){
                fscanf(file_ptr2,"%f",&xx);
                slices[i][j][k]=xx;
            } /* end of for k */
           } /* end of for j */
        } /* end of for i */
/* Here to input y value i.e., heights */
        for(m=0;m<dcount;m++){
          for(j=0;j<NPTS;j++){
         slices[m][j][2]=heights[m];
         } /* end of for j */
        } /* end of for m */

        fclose(file_ptr2);
}
/* here to calculate plot array */

calcarray()
{

  int count,i,j,height,dens,theta,thetatemp;
 float oldpoly[2],dist,pi2,phi;


        dens=720;

          for(j=0;j<dcount;j++){
         dist = 0;
         thetatemp = 0;

         height = (int)heights[j];
         cardret(0,dens,j);
        oldpoly[0]=apoly[0];
        oldpoly[1]=apoly[1];
        plotfile[j][0]=(float)height;
```

```
        count = 1;
          for(i=1;i<=dens;i++){
          cardret(i,dens,j);
         dist = fhypot((apoly[0]-oldpoly[0]),(apoly[1]-oldpoly[1])) + dist;

      pi2=2*PI;
      if(apoly[1]>0)phi=fatan(apoly[0]/apoly[1]);
      else if(apoly[1]<0)phi=PI+fatan(apoly[0]/apoly[1]);
      else if(apoly[1]==0 && apoly[0]>0)phi= PI/2;
      else if(apoly[1]==0 && apoly[0]<0)phi= 3*PI/2;
      else if(apoly[1]==0 && apoly[0]==0)phi= 0;
      if(phi<=0)phi=pi2+phi;
      if(phi>pi2)phi=phi-pi2;
      theta=  phi * 180/PI;
           if((theta*10) % 48 < 10 && thetatemp != theta  && count <
MAXLINES){
          thetatemp = theta ;
          plotfile[j][count]=dist;
          count++;
          } /* end of if theta */
        oldpoly[0]=apoly[0];
        oldpoly[1]=apoly[1];
        } /* end of for i */
       } /* end of for j */
}
/* here to create a plot file */


sliceplot()
{
 FILE *fopen();
  char *string1,string2[50],*string3,string4[50];
  char *string5,string6[50],*string7,string8[50];
   int count,i,j,height,penno,edge1,edge2,htmin,htmax;

  int plotw, ploth;

  string1 = "Enter plot file name ";
  string3 = "w";
```

```
  string5 = "Enter title.";
  string7 = "Enter issue No.";


      plotw = HWIDTH * PLOTUNIT;
      ploth = HDEPTH * PLOTUNIT;
      for(i=0;i<50;i++){
      string2[i]='\0';
      string4[i]='\0';
      string6[i]='\0';
      string8[i]='\0';
      }
      i=0;
      while(string1[i] !='\0'){
      string2[i]=string1[i];
      i++;
      }
      winset(textwin);
      winpop();
      c3s(whitevec);
      clear();
      c3s(blackvec);
      putline(string2,1.0,5.0);
      count = getline(string2,i,50);
      sscanf(&string2[i],"%s",string4);
      count=quest(string5,string6);
      count=quest(string7,string8);
      winpush();
      file_ptr=fopen(string4,string3);
     fprintf(file_ptr,"IN;SP1;PA;FS6;\n");
      fprintf(file_ptr,"IP%d,%d,%d,%d;\n",-plotw,-ploth,plotw,ploth);
      fprintf(file_ptr,"SC %d,40,%d,40,2;\n",-HWIDTH,-HDEPTH);
      fprintf(file_ptr,"PA%d,%d;EA%d,%d;\n",-HWIDTH,-
HDEPTH,HWIDTH,HDEPTH);
      logo(1,1,string6,string8);
      penno=1;
      edge1= -HWIDTH+25;/* left edge */
      edge2= -HDEPTH+100;/* bottom */
```

```
        htmin=(int)heights[0];
        htmax=(int)heights[dcount-1];
        fprintf(file_ptr,"DI0,1;DT\3;LO16;PU%d,0;LB    Centre
Front\3\n",(int)-(HWIDTH-10));
        fprintf(file_ptr,"DI0,1;DT\3;LO14;PU%d,0;LB    Centre
Back\3\n",(int)HWIDTH-10);
/* here to plot horizontal lines */
       fprintf(file_ptr,"SP%d;\n",penno);
         for(j=2;j<dcount-1;j++){
         height = (int)heights[j];
       fprintf(file_ptr,"DI1,0;DT\3;LO14;PU%d,%d;LB%d\3
\n",edge1+100,height-htmin+edge2,j+1);
       fprintf(file_ptr,"PU%d,%d;",edge1,height-htmin+edge2);
       fprintf(file_ptr,"PD%d,%d;\n",(int)plotfile[j][MAXLINES-
1]+edge1,height-htmin+edge2);
        }
/* here to plot vertical lines */
       penno=2;
      fprintf(file_ptr,"SP%d;\n",penno);
     fprintf(file_ptr,"CV1;\n");
       fprintf(file_ptr,"PU%d,%d;\n",edge1,edge2);
       fprintf(file_ptr,"DI1,0;DT\3;LO16;LB%d\3 \n",0);
       fprintf(file_ptr,"PU%d,%d;\n",edge1,edge2);
       fprintf(file_ptr,"LT6;PD%d,%d;\n",edge1,htmax-htmin+edge2);
         for(i=1;i<MAXLINES;i++){
         if(i%5==0)fprintf(file_ptr,"LT6;");
         else fprintf(file_ptr,"LT\n;");
         fprintf(file_ptr,"PU%d,%d;\n",edge1+(int)plotfile[0][i],edge2);
         fprintf(file_ptr,"DI1,0;DT\3;LO16;LB%d\3 \n",i);
         fprintf(file_ptr,"PU%d,%d;\n",edge1+(int)plotfile[0][i],edge2);
         fprintf(file_ptr,"PD");
           for(j=0;j<dcount-1;j++){

fprintf(file_ptr,"%d,%d,",edge1+(int)plotfile[j][i],(int)plotfile[j][0]-
htmin+edge2);
           if(j%10==0)fprintf(file_ptr,"\n");
         }
       }
```

```
}
/* here to plot master ref */

masterplot()
{
        int i,j,penno,edge1,edge2,htmin;


        penno=3;
        edge1= -HWIDTH+25;/* left edge */
        edge2= -HDEPTH+100;/* bottom */
        htmin=(int)heights[0];


       fprintf(file_ptr,"\nSP%d;\n",penno);
      fprintf(file_ptr,"CV1;\n");
        for(i=1;i<MAXLINES;i++){
        if(i%5==0)fprintf(file_ptr,"LT6;");
        else fprintf(file_ptr,"LT\n;");
        fprintf(file_ptr,"PU%d,%d;\n",edge1+(int)plotfile[0][i],edge2);
        fprintf(file_ptr,"DI1,0;DT\3;LO14;LB%d\3 \n",i);
        fprintf(file_ptr,"PU%d,%d;\n",edge1+(int)plotfile[0][i],edge2);
        fprintf(file_ptr,"PD");
          for(j=0;j<dcount-1;j++){
          fprintf(file_ptr,"%d,%d,",edge1+(int)plotfile[j][i],(int)heights[j]-
htmin+edge2);
          if(j%10==0)fprintf(file_ptr,"\n");
         }
       }
      fprintf(file_ptr,"CV0;\n");
       fprintf(file_ptr,"PU;PG;");
       fclose(file_ptr);
}
/* here to draw logo */

logo(page,maxpage,title,issue)
        int page,maxpage;
        char title[],issue[];
```

```
{
        int aa,bb,cc,dd,xedge,yedge,labelx,notex;

        xedge= -HWIDTH;/* left edge of drawing */
        labelx= HWIDTH-200;/* inside edge of label */
        yedge= -HDEPTH;/* bottom line of frame */
        notex= -HWIDTH+20; /* start of notes */

        aa= yedge;
        bb= yedge+50;
        cc= HWIDTH-200;
        dd= HWIDTH;

        fprintf(file_ptr,"SP1;PU%d,%d;EA%d,%d;\n",cc,bb,dd,aa);
        fprintf(file_ptr,"DI1,0;");
        aa=yedge +40;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LBLoughborough
Anthropometric Shadow Scanner\3 \n",labelx,aa);
        aa=yedge +25;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB%s\3
\n",labelx,aa,title);
        aa=yedge +10;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LBPage %d of %d. Issue
%s \3 \n",labelx,aa,page,maxpage,issue);
        aa=yedge +80;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB1. Use for horizontal
measurements only.\3 \n",notex,aa);
        aa=yedge +65;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB2. Red plot is average
size 34b for reference.\3 \n",notex,aa);
        aa=yedge +50;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB3. Blue plot represents
actual data.\3 \n",notex,aa);
        aa=yedge +35;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB4. Numbers refer to
line numbers in data file.\3 \n",notex,aa);
}
/* here to get line into s, return length */
```

```
int getline(s,cnt,lim)
        char s[];
        int cnt,lim;
{
        Device val;
        long dev;
        int c,flg,i;

        flg=1; i=cnt;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
        case KEYBD:
        if(val=='\b'){
        i--;
        s[i]= '\0';
        c3s(whitevec);
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        s[i++]=val;
        }
        if(val=='\n')flg=0;
        putline(s,1.0,5.0);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
```

```
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
  return(i);
}
/* here to write a line to a window */

putline(s,x,y)
        char s[];
        Coord x,y;
{
        cmov2(x,y);
        charstr(s);
}
/* here to return a point from a Cardinal curve */
cardret(point,maxpoints,sliceno)
        int point,maxpoints,sliceno;
{
    .   int s,u;
        float frangle[4],temp[4],npoints,fracpnt,interval;

        if(point==maxpoints){
        apoly[0]=slices[sliceno][NPTS-1][0];
        apoly[1]=slices[sliceno][NPTS-1][1];
        apoly[2]=0;
        }
        else {
        apoly[0]=0;
        apoly[1]=0;
        apoly[2]=0;

        interval=NPTS-1;
        npoints = (float) maxpoints/interval;
        s= point/npoints;
        fracpnt = (point-s*npoints)/npoints;
        s=s-1;
```

```
        frangle[3]=1.0;
        frangle[2]=fracpnt;
        frangle[1]=frangle[2]*frangle[2];
        frangle[0]=frangle[1]*frangle[2];
        colmat(frangle,cardinal,temp);
          for(u=0;u<4;u++){
           if(s<0 && u==0){
           apoly[0]=apoly[0]-temp[0]*slices[sliceno][1][0];
           apoly[1]=apoly[1]+temp[0]*slices[sliceno][1][1];
           }
            else if ((u+s)>=0 && (u+s)<NPTS ){
           apoly[0]=apoly[0]+temp[u]*slices[sliceno][u+s][0];
           apoly[1]=apoly[1]+temp[u]*slices[sliceno][u+s][1];
           }
            else if((u+s)==NPTS){
           apoly[0]=apoly[0]-temp[3]*slices[sliceno][NPTS-2][0];
           apoly[1]=apoly[1]+temp[3]*slices[sliceno][NPTS-2][1];
           }
        } /* end of for u */
        } /* end of else */
}
/*matrix multiply 4 X 1 array times 4 X 4 array, A * B result in R */

colmat(mata,matb,matr)
        float mata[4],matb[4][4],matr[4];
{
        int i,j;
        float temp[4];

        for(i=0;i<4;i++){
        temp[i]=0;
          for(j=0;j<4;j++){
         temp[i]=temp[i]+mata[j]*matb[j][i];
         }
        }
        for(i=0;i<4;i++)matr[i]=temp[i];
}
/* here to print slice array */
```

```
pslice()
{
        int i,j;

        for(i=0;i<dcount;i++){
          for(j=0;j<NPTS;j++){
                          printf("%d      %7.2f      %7.2f      %7.2f
",j,slices[i][j][0],slices[i][j][1],slices[i][j][2]);
           if(j%2==0)printf("\n");
          }
        printf("\n");
        }
}
/* here to print plotfile array */

arrprnt()
{
        int i,j;

        for(i=0;i<dcount;i++){
          for(j=0;j<MAXLINES;j++){
           printf("%7.2f  ",plotfile[i][j]);
          if(j % 8==0 || j>=MAXLINES-1)printf("\n");
          }
        printf("\n");
        }
}
/* here to ask a question & get an answer */

quest(infile,outfile)
        char infile[],outfile[];
{
        Device val;
        long dev;
        int c,flg,i;

        winset(textwin);
        c3s(whitevec);
```

```
clear();
c3s(blackvec);
cmov2(1.0,7.0);
charstr(infile);
cmov2(1.0,4.0);
flg=1; i=0;
while(flg==1){
while(qtest()){
dev=qread(&val);
switch(dev){
case KEYBD:
if(val=='\b'){
i--;
outfile[i]= '\0';
c3s(whitevec);
clear();
c3s(blackvec);
} /* end of if val */
else {
outfile[i++]=val;
}
if(val=='\n')flg=0;
winset(textwin);
c3s(whitevec);
clear();
c3s(blackvec);
cmov2(1.0,7.0);
charstr(infile);
cmov2(1.0,4.0);
charstr(outfile);
qreset();
break;
case ESCKEY:
greset();
gexit();
exit(0);
dglclose(-1);
break;
```

```
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
   } /* end of while flg */
   return(i);
}
```

/* gradsplot4.c */

```
/* Same as gradsplot1.c but with masterplot. */
/* Creates a plotfile for H.P.Plotter from 16 spline points */
/* to give true surface distance from centre back in x direction. */
#include <stdio.h>
#include "device.h"
#include <gl.h>
#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>
#define NPTS 16 /* number of points in curve fit polygon */
#define NSLICES 50 /* maximum number of slices */
#define PI 3.141592654
#define PLOTUNIT 40 /* number of plotter units per mm. */
#define HWIDTH 350 /* half width of surrounding rectangle in mm's.
*/
#define HDEPTH 378 /* half height of surrounding rectangle in mm's.
*/
#define MAXLINES 39 /* maximum number of vertical lines. */

Matrix cardinal = {
        {-0.5, 1.5,-1.5, 0.5},
        { 1.0,-2.5, 2.0,-0.5},
        {-0.5, 0.0, 0.5, 0.0},
        { 0.0, 1.0, 0.0, 0.0},
        };
```

```
        short blackvec[3] = { 0, 0, 0};
        short redvec[3] = { 255, 0, 0};
        short greenvec[3] = { 73, 206, 130};
        short bluevec[3] = { 0, 0, 255};
        short whitevec[3] = {255,255,255};
        long textwin;
        float  MINIHEAD[20];
        float slices[NSLICES][NPTS+1][3];
        float plotfile[NSLICES][MAXLINES];
        float heights[NSLICES];
        float apoly[3];
        int dcount,bytecount,mflg;
        int maxhite, minhite;
        char *name;
        FILE *file_ptr;


main()
        {
        inittextwin();
  qdevice(ESCKEY);
  qdevice(KEYBD);
  qdevice(RETKEY);
        slicelod();
        calcarray();
        sliceplot();
        masterget();
        calcarray();
        masterplot();
        }


inittextwin()
{
   prefposition(2*
XMAXSCREEN/5,4*XMAXSCREEN/5,4*YMAXSCREEN/10,5*YMAXS
CREEN/10);
   textwin = winopen("Slice plot.");
```

```
        RGBmode();
        shademodel(FLAT);
        ortho2(0,15,0,10);
        foreground();
        gconfig();
}
/* here to load  patches slice data (36 line text file)*/

slicelod()
{
 FILE *fopen();
  char *string1,string2[50],*string3,string4[50];
   int count,i,j,k,m,spangcnt;
  float xx;

   string1 = "Enter file name ";
  string3 = "r";

        for(i=0;i<50;i++){
        string2[i]='\0';
        string4[i]='\0';
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }

  name = &string4[0];
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
  putline(string2,1.0,5.0);
  count = getline(string2,i,50);
  sscanf(&string2[i],"%s",name);
  winpush();
```

```
  file_ptr=fopen(name,string3);
        fscanf(file_ptr,"%d",&dcount);
        fscanf(file_ptr,"%d",&spangcnt);
          for(i=0;i<dcount;i++){
            for(j=0;j<=NPTS;j++){
             for(k=0;k<2;k++){
        fscanf(file_ptr,"%f",&xx);
        slices[i][j][k]=xx;
           } /* end of for k */
          } /* end of for j */
        } /* end of for i */
/* Here to input y value i.e., heights */
        for(m=0;m<dcount;m++){
          for(j=0;j<NPTS;j++){
         heights[m]=slices[m][NPTS][0];
         slices[m][j][2]=heights[m];
         } /* end of for j */
        } /* end of for m */

        fclose(file_ptr);
}

masterget()
{
 FILE *fopen(),*file_ptr2;
  char *string3,*mastername;
 int i,j,k,m,spangcnt;
 float xx;

 string3 = "r";
 mastername = "av34b.xmfac";

file_ptr2=fopen(mastername,string3);
       fscanf(file_ptr2,"%d",&dcount);
       fscanf(file_ptr2,"%d",&spangcnt);
         for(i=0;i<dcount;i++){
           for(j=0;j<=NPTS;j++){
            for(k=0;k<2;k++){
```

```
                fscanf(file_ptr2,"%f",&xx);
                slices[i][j][k]=xx;
              } /* end of for k */
            } /* end of for j */
          } /* end of for i */
 /* Here to input y value i.e., heights */
          for(m=0;m<dcount;m++){
            for(j=0;j<NPTS;j++){
           slices[m][j][2]=heights[m];
            } /* end of for j */
          } /* end of for m */

          fclose(file_ptr2);
}
/* here to calculate plot array */

calcarray()
{

  int count,i,j,height,dens,theta,thetatemp;
 float oldpoly[2],dist,pi2,phi;


        dens=720;

          for(j=0;j<dcount;j++){
         dist = 0;
         thetatemp = 0;

         height = (int)heights[j];
         cardret(0,dens,j);
        oldpoly[0]=apoly[0];
        oldpoly[1]=apoly[1];
        plotfile[j][0]=(float)height;
        count = 1;
          for(i=0;i<=dens;i++){
         cardret(i,dens,j);
         dist = fhypot((apoly[0]-oldpoly[0]),(apoly[1]-oldpoly[1])) + dist;
```

```
        pi2=2*PI;
        if(apoly[1]>0)phi=fatan(apoly[0]/apoly[1]);
        else if(apoly[1]<0)phi=PI+fatan(apoly[0]/apoly[1]);
        else if(apoly[1]==0 && apoly[0]>0)phi= PI/2;
        else if(apoly[1]==0 && apoly[0]<0)phi= 3*PI/2;
        else if(apoly[1]==0 && apoly[0]==0)phi= 0;
        if(phi<=0)phi=pi2+phi;
        if(phi>pi2)phi=phi-pi2;


        theta=  phi * 180/PI;
            if((theta*10) % 48 < 10 && thetatemp != theta   && count <
                                                    MAXLINES){
            thetatemp = theta ;
            plotfile[j][count]=dist;
/*
printf("%d %d %d\n",j,count,theta);
*/

            count++;
           } /* end of if theta */
          oldpoly[0]=apoly[0];
          oldpoly[1]=apoly[1];
          } /* end of for i */
        } /* end of for j */
}
/* here to resolve atan calculations */


float artan(yy,xx)
        float yy,xx;
{
        float phi,pi2;

        pi2=2*PI;
        if(xx>0)phi=fatan(yy/xx);
        else if(xx<0)phi=PI+fatan(yy/xx);
        else if(xx==0 && yy>0)phi= PI/2;
```

```
        else if(xx==0 && yy<0)phi= 3*PI/2;
        else if(xx==0 && yy==0)phi= 0;
        phi=3*PI/2-phi; /* not in normal artan routine */
        if(phi<=0)phi=pi2+phi;
        if(phi>pi2)phi=phi-pi2;
        return(phi);
}
/* here to create a plot file */

sliceplot()
{
 FILE *fopen();
  char *string1,string2[50],*string3,string4[50];
  char *string5,string6[50],*string7,string8[50];
   int count,i,j,height,penno,edge1,edge2,htmin,htmax;

  int plotw, ploth;

  string1 = "Enter plot file name ";
  string3 = "w";
  string5 = "Enter title.";
  string7 = "Enter issue No.";


      plotw = HWIDTH * PLOTUNIT;
      ploth = HDEPTH * PLOTUNIT;
      for(i=0;i<50;i++){
      string2[i]='\0';
      string4[i]='\0';
      string6[i]='\0';
      string8[i]='\0';
      }
      i=0;
      while(string1[i] !='\0'){
      string2[i]=string1[i];
      i++;
      }
      winset(textwin);
```

```
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
        putline(string2,1.0,5.0);
        count = getline(string2,i,50);
        sscanf(&string2[i],"%s",string4);
        count=quest(string5,string6);
        count=quest(string7,string8);
        winpush();
        file_ptr=fopen(string4,string3);
       fprintf(file_ptr,"IN;SP1;PA;FS6;\n");
       fprintf(file_ptr,"IP%d,%d,%d,%d;\n",-plotw,-ploth,plotw,ploth);
       fprintf(file_ptr,"SC %d,40,%d,40,2;\n",-HWIDTH,-HDEPTH);
       fprintf(file_ptr,"PA%d,%d;EA%d,%d;\n",-HWIDTH,-
 HDEPTH,HWIDTH,HDEPTH);
       logo(1,1,string6,string8);
       penno=1;
       edge1= -HWIDTH+25;/* left edge */
       edge2= -HDEPTH+100;/* bottom */
       htmin=(int)heights[0];
       htmax=(int)heights[dcount-1];
       fprintf(file_ptr,"DI0,1;DT\3;LO16;PU%d,0;LB    Centre
Front\3\n",(int)(HWIDTH-10));
       fprintf(file_ptr,"DI0,1;DT\3;LO14;PU%d,0;LB    Centre
Back\3\n",(int)-(HWIDTH-10));
/* here to plot horizontal lines */
      fprintf(file_ptr,"SP%d;\n",penno);
        for(j=2;j<dcount-1;j++){
        height = (int)heights[j];
      fprintf(file_ptr,"DI1,0;DT\3;LO14;PU%d,%d;LB%d\3
\n",edge1+100,height-htmin+edge2,j+1);
      fprintf(file_ptr,"PU%d,%d;",edge1,height-htmin+edge2);
      fprintf(file_ptr,"PD%d,%d;\n",(int)(edge1+plotfile[j][MAXLINES
-1]),height-htmin+edge2);
       }
/* here to plot vertical lines */
      penno=2;
```

```
        fprintf(file_ptr,"SP%d;\n",penno);
        fprintf(file_ptr,"CV1;\n");
          for(i=0;i<MAXLINES;i++){
          if(i%5==0)fprintf(file_ptr,"LT6;");
          else fprintf(file_ptr,"LT;");
          if(i==0){

fprintf(file_ptr,"PU%d,%d;\n",(int)(edge1+plotfile[0][MAXLINES-
1]),edge2);
           fprintf(file_ptr,"DI1,0;DT\3;LO16;LB%d\3 \n",i);

fprintf(file_ptr,"PU%d,%d;\n",(int)(edge1+plotfile[0][MAXLINES-
1]),edge2);
           fprintf(file_ptr,"PD");
          }
          else{

fprintf(file_ptr,"PU%d,%d;\n",(int)(edge1+(plotfile[0][MAXLINES-1]-
plotfile[0][i])),edge2);
           fprintf(file_ptr,"DI1,0;DT\3;LO16;LB%d\3 \n",i);

fprintf(file_ptr,"PU%d,%d;\n",(int)(edge1+(plotfile[0][MAXLINES-1]-
plotfile[0][i])),edge2);
           fprintf(file_ptr,"PD");
          }
       for(j=2;j<dcount-2;j++){
       if(i==0)fprintf(file_ptr,"%d,%d,",(int)(edge1+plotfile[j][MAXLIN
ES-1]),(int)plotfile[j][0]-htmin+edge2);
       else fprintf(file_ptr,"%d,%d,",(int)(edge1+(plotfile[j][MAXLINES-
1]-plotfile[j][i])),(int)plotfile[j][0]-htmin+edge2);
       if(j%10==0)fprintf(file_ptr,"\n");
       } /* end of for j */
       fprintf(file_ptr,"%d,%d;\n",(int)(edge1+(plotfile[dcount-
2][MAXLINES-1]-plotfile[dcount-2][i])),(int)plotfile[dcount-2][0]-
htmin+edge2);
      } /* end of for i */
}
/* here to plot master ref */
```

```
masterplot()
{
        int i,j,penno,edge1,edge2,htmin;


        penno=3;
        edge1= -HWIDTH+25;/* left edge */
        edge2= -HDEPTH+100;/* bottom */
        htmin=(int)heights[0];


       fprintf(file_ptr,"\nSP%d;\n",penno);
       fprintf(file_ptr,"CV1;\n");
         for(i=0;i<MAXLINES;i++){
         if(i%5==0)fprintf(file_ptr,"LT6;");
         else fprintf(file_ptr,"LT;");
          if(i==0){

fprintf(file_ptr,"PU%d,%d;\n",(int)(edge1+plotfile[0][MAXLINES-
1]),edge2);
           fprintf(file_ptr,"DI1,0;DT\3;LO14;LB%d\3 \n",i);

fprintf(file_ptr,"PU%d,%d;\n",(int)(edge1+plotfile[0][MAXLINES-
1]),edge2);
           fprintf(file_ptr,"PD");
          }
          else{

fprintf(file_ptr,"PU%d,%d;\n",(int)(edge1+(plotfile[0][MAXLINES-1]-
plotfile[0][i])),edge2);
           fprintf(file_ptr,"DI1,0;DT\3;LO14;LB%d\3 \n",i);

fprintf(file_ptr,"PU%d,%d;\n",(int)(edge1+(plotfile[0][MAXLINES-1]-
plotfile[0][i])),edge2);
           fprintf(file_ptr,"PD");
          }
           for(j=2;j<dcount-2;j++){
```

```
        if(i==0)fprintf(file_ptr,"%d,%d,",(int)(edge1+plotfile[j][MAXLIN
ES-1]),(int)plotfile[j][0]-htmin+edge2);
        else fprintf(file_ptr,"%d,%d,",(int)(edge1+(plotfile[j][MAXLINES-
1]-plotfile[j][i])),(int)plotfile[j][0]-htmin+edge2);
        if(j%10==0)fprintf(file_ptr,"\n");
          }
        fprintf(file_ptr,"%d,%d;\n",(int)(edge1+(plotfile[dcount-
2][MAXLINES-1]-plotfile[dcount-2][i])),(int)plotfile[dcount-2][0]-
htmin+edge2);
         }
       fprintf(file_ptr,"CV0;\n");
        fprintf(file_ptr,"PU;PG;");
        fclose(file_ptr);
}
/* here to draw logo */

logo(page,maxpage,title,issue)
        int page,maxpage;
        char title[],issue[];
{
        int aa,bb,cc,dd,xedge,yedge,labelx,notex;

        xedge= -HWIDTH;/* left edge of drawing */
        labelx= HWIDTH-200;/* inside edge of label */
        yedge= -HDEPTH;/* bottom line of frame */
        notex= -HWIDTH+20; /* start of notes */

        aa= yedge;
        bb= yedge+50;
        cc= HWIDTH-200;
        dd= HWIDTH;

        fprintf(file_ptr,"SP1;PU%d,%d;EA%d,%d;\n",cc,bb,dd,aa);
        fprintf(file_ptr,"DI1,0;");
        aa=yedge +40;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LBLoughborough
Anthropometric Shadow Scanner\3 \n",labelx,aa);
        aa=yedge +25;
```

```
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB%s\3
\n",labelx,aa,title);
        aa=yedge +10;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LBPage %d of %d. Issue
%s \3 \n",labelx,aa,page,maxpage,issue);
        aa=yedge +80;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB1. Use for horizontal
measurements only.\3 \n",notex,aa);
        aa=yedge +65;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB2. Red plot is average
size 34b for reference.\3 \n",notex,aa);
        aa=yedge +50;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB3. Blue plot represents
actual data.\3 \n",notex,aa);
        aa=yedge +35;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB4. Numbers refer to
line numbers in data file.\3 \n",notex,aa);
}
/* here to get line into s, return length */

int getline(s,cnt,lim)
        char s[];
        int cnt,lim;
{
        Device val;
        long dev;
        int c,flg,i;

        flg=1; i=cnt;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
        case KEYBD:
        if(val=='\b'){
        i--;
        s[i]= '\0';
        c3s(whitevec);
```

```
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        s[i++]=val;
        }
        if(val=='\n')flg=0;
        putline(s,1.0,5.0);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
   } /* end of while flg */
   return(i);
}
/* here to write a line to a window */

putline(s,x,y)
        char s[];
        Coord x,y;
{
        cmov2(x,y);
        charstr(s);
}
/* here to return a point from a Cardinal curve */
cardret(point,maxpoints,sliceno)
        int point,maxpoints,sliceno;
```

```
{
    int s,u;
    float frangle[4],temp[4],npoints,fracpnt,interval;

    if(point==maxpoints){
    apoly[0]=slices[sliceno][NPTS-1][0];
    apoly[1]=slices[sliceno][NPTS-1][1];
    apoly[2]=0;
    }
    else {
    apoly[0]=0;
    apoly[1]=0;
    apoly[2]=0;

    interval=NPTS-1;
    npoints = (float) maxpoints/interval;
    s= point/npoints;
    fracpnt = (point-s*npoints)/npoints;
    s=s-1;

    frangle[3]=1.0;
    frangle[2]=fracpnt;
    frangle[1]=frangle[2]*frangle[2];
    frangle[0]=frangle[1]*frangle[2];
    colmat(frangle,cardinal,temp);
      for(u=0;u<4;u++){
       if(s<0 && u==0){
      apoly[0]=apoly[0]-temp[0]*slices[sliceno][1][0];
      apoly[1]=apoly[1]+temp[0]*slices[sliceno][1][1];
      }
       else if ((u+s)>=0 && (u+s)<NPTS ){
      apoly[0]=apoly[0]+temp[u]*slices[sliceno][u+s][0];
      apoly[1]=apoly[1]+temp[u]*slices[sliceno][u+s][1];
      }
       else if((u+s)==NPTS){
      apoly[0]=apoly[0]-temp[3]*slices[sliceno][NPTS-2][0];
      apoly[1]=apoly[1]+temp[3]*slices[sliceno][NPTS-2][1];
      }
```

```
        } /* end of for u */
        } /* end of else */
 }
 /*matrix multiply 4 X 1 array times 4 X 4 array, A * B result in R */

 colmat(mata,matb,matr)
        float mata[4],matb[4][4],matr[4];
 {
        int i,j;
        float temp[4];

        for(i=0;i<4;i++){
        temp[i]=0;
          for(j=0;j<4;j++){
         temp[i]=temp[i]+mata[j]*matb[j][i];
         }
        }
        for(i=0;i<4;i++)matr[i]=temp[i];
}
/* here to print slice array */

pslice()
{
       int i,j;

       for(i=0;i<dcount;i++){
         for(j=0;j<NPTS;j++){
                          printf("%d     %7.2f     %7.2f     %7.2f
",j,slices[i][j][0],slices[i][j][1],slices[i][j][2]);
           if(j%2==0)printf("\n");
         }
       printf("\n");
       }
}
/* here to print plotfile array */
```

```
arrprnt()
{
        int i,j;

        for(i=0;i<dcount;i++){
          for(j=0;j<MAXLINES;j++){
           printf("%7.2f  ",plotfile[i][j]);
          if(j % 8==0 | | j>=MAXLINES-1)printf("\n");
         }
        printf("\n");
        }
}
/* here to ask a question & get an answer */

quest(infile,outfile)
        char infile[],outfile[];
{
        Device val;
        long dev;
        int c,flg,i;

        winset(textwin);
        c3s(whitevec);
        clear();
        c3s(blackvec);
        cmov2(1.0,7.0);
        charstr(infile);
        cmov2(1.0,4.0);
        flg=1; i=0;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
        case KEYBD:
        if(val=='\b'){
        i--;
        outfile[i]= '\0';
        c3s(whitevec);
```

```
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        outfile[i++]=val;
        }
        if(val=='\n')flg=0;
        winset(textwin);
        c3s(whitevec);
        clear();
        c3s(blackvec);
        cmov2(1.0,7.0);
        charstr(infile);
        cmov2(1.0,4.0);
        charstr(outfile);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
  return(i);
}
```

/* gradsplotvert.c */

/* Creates a plotfile for H.P.Plotter from 16 spline points */

```
#include <stdio.h>
#include "device.h"
#include <gl.h>
#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>
#define NPTS 16 /* number of points in curve fit polygon */
#define VLINES 37 /* number of vertical lines */
#define NSLICES 50 /* maximum number of slices */
#define UMAX 10000 /* maximum u value allowed */
#define DEF 14 /* binary digit resolution */
#define PI 3.141592654
#define RAD PI/180
#define PLOTUNIT 40 /* number of plotter units per mm. */
#define HWIDTH 265 /* half width of surrounding rectangle in mm's.
*/
#define HDEPTH 500 /* half height of surrounding rectangle in mm's.
*/

Matrix cardinal = {
        {-0.5, 1.5,-1.5, 0.5},
        { 1.0,-2.5, 2.0,-0.5},
        {-0.5, 0.0, 0.5, 0.0},
        { 0.0, 1.0, 0.0, 0.0},
        };


        short blackvec[3] = { 0, 0, 0};
        short redvec[3] = { 255, 0, 0};
        short greenvec[3] = { 73, 206, 130};
        short bluevec[3] = { 0, 0, 255};
        short whitevec[3] = {255,255,255};
        long textwin;
        float  MINIHEAD[20];
        float slices[NSLICES][NPTS+1][3];
        float angfile[NSLICES][VLINES+1];
        float plotfile[NSLICES][38];
        float heights[NSLICES];
```

```
        float vdist[NSLICES][VLINES];
        float apoly[3];
        int dcount,bytecount,mflg;
        int maxhite, minhite;
        char *name;
        FILE *file_ptr;


main()
        {
        inittextwin();
  qdevice(ESCKEY);
  qdevice(KEYBD);
  qdevice(RETKEY);
        slicelod();
        fixfile();
        vertdist();
        sliceplot();

        masterget();
        fixfile();
        vertdist();
        masterplot();

        }


inittextwin()
{
  prefposition(2*
XMAXSCREEN/5,4*XMAXSCREEN/5,4*YMAXSCREEN/10,5*YMAXS
CREEN/10);
  textwin = winopen("Slice plot.");
        RGBmode();
        shademodel(FLAT);
        ortho2(0,15,0,10);
        foreground();
        gconfig();
}
/* here to load  patches slice data (36 line text file)*/
```

```
slicelod()
{
 FILE *fopen(),*file_ptr;
 char *string1,string2[50],*string3,string4[50],buffer[20];
  int count,i,j,k,m,spangcnt;
 float xx;

  string1 = "Enter file name ";
 string3 = "r";

       for(i=0;i<50;i++){
       string2[i]='\0';
       string4[i]='\0';
       }
       i=0;
       while(string1[i] !='\0'){
       string2[i]=string1[i];
       i++;
       }

 name = &string4[0];
       winset(textwin);
       winpop();
       c3s(whitevec);
       clear();
       c3s(blackvec);
 putline(string2,1.0,5.0);
 count = getline(string2,i,50);
 sscanf(&string2[i],"%s",name);
 winpush();
file_ptr=fopen(name,string3);
      fscanf(file_ptr,"%d",&dcount);
      fscanf(file_ptr,"%d",&spangcnt);
        for(i=0;i<dcount;i++){
          for(j=0;j<=NPTS;j++){
           for(k=0;k<2;k++){
      fscanf(file_ptr,"%f",&xx);
      slices[i][j][k]=xx;
```

```
          } /* end of for k */
         } /* end of for j */
        } /* end of for i */
 /* Here to input y value i.e., heights */
        for(m=0;m<dcount;m++){
          for(j=0;j<NPTS;j++){
         heights[m]=slices[m][NPTS][0];
         slices[m][j][2]=heights[m];
         } /* end of for j */
        } /* end of for m */

        fclose(file_ptr);
}

masterget()
{
  FILE *fopen(),*file_ptr;
   char *string3,*mastername;
   int count,i,j,k,m,spangcnt;
  float xx;

  string3 = "r";
  mastername = "av34b.xmfac";

 file_ptr=fopen(mastername,string3);
       fscanf(file_ptr,"%d",&dcount);
       fscanf(file_ptr,"%d",&spangcnt);
         for(i=0;i<dcount;i++){
           for(j=0;j<=NPTS;j++){
            for(k=0;k<2;k++){
               fscanf(file_ptr,"%f",&xx);
               slices[i][j][k]=xx;
         } /* end of for k */
        } /* end of for j */
       } /* end of for i */
/* Here to input y value i.e., heights */
       for(m=0;m<dcount;m++){
         for(j=0;j<NPTS;j++){
```

```
        slices[m][j][2]=heights[m];
        } /* end of for j */
       } /* end of for m */

       fclose(file_ptr);
}
/* here to calculate plot array */

calcarray()
{

  int count,i,j,height,dens,theta,thetatemp;
 float oldpoly[2],dist,pi2,phi;


       dens=720;

        for(j=0;j<dcount;j++){
       dist = 0;
       thetatemp = 0;

       height = (int)heights[j];
       cardret(0,dens,j);
       oldpoly[0]=apoly[0];
       oldpoly[1]=apoly[1];
       plotfile[j][0]=(float)height;
       count = 1;
         for(i=1;i<=dens;i++){
         cardret(i,dens,j);
         dist = fhypot((apoly[0]-oldpoly[0]),(apoly[1]-oldpoly[1])) + dist;

      pi2=2*PI;
      if(apoly[1]>0)phi=fatan(apoly[0]/apoly[1]);
      else if(apoly[1]<0)phi=PI+fatan(apoly[0]/apoly[1]);
      else if(apoly[1]==0 && apoly[0]>0)phi= PI/2;
      else if(apoly[1]==0 && apoly[0]<0)phi= 3*PI/2;
      else if(apoly[1]==0 && apoly[0]==0)phi= 0;
      if(phi<0)phi=pi2+phi;
```

```
        if(phi>pi2)phi=phi-pi2;

        theta=  phi * 180/PI;
            if(theta % 5 < 1 && thetatemp != theta   && count < 38){
            thetatemp = theta ;
            plotfile[j][count]=dist;
            count++;
            } /* end of if theta */
           oldpoly[0]=apoly[0];
           oldpoly[1]=apoly[1];
           } /* end of for i */
         } /* end of for j */
}
/* here to create a plot file */

sliceplot()
{
 FILE *fopen();
  char *string1,string2[50],*string3,string4[50];
  char *string5,string6[50],*string7,string8[50];
   int count,i,j,height,dens,penno,edge1,edge2,htmin,htmax,htlast;
  int plotw, ploth;
  float poly[2],oldpoly[2],distance;

  string1 = "Enter plot file name ";
  string3 = "w";
  string5 = "Enter title.";
  string7 = "Enter issue No.";

        dens=75;
        plotw = HWIDTH * PLOTUNIT;
        ploth = HDEPTH * PLOTUNIT;
        distance = 0;
        for(i=0;i<50;i++){
        string2[i]='\0';
        string4[i]='\0';
        string6[i]='\0';
        string8[i]='\0';
```

```
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
        putline(string2,1.0,5.0);
        count = getline(string2,i,50);
        sscanf(&string2[i],"%s",string4);
        count=quest(string5,string6);
        count=quest(string7,string8);
        winpush();
        file_ptr=fopen(string4,string3);
       fprintf(file_ptr,"IN;SP1;PA;FS6;\n");
       fprintf(file_ptr,"RO90;\n");
       fprintf(file_ptr,"IP%d,%d,%d,%d;\n",-plotw,-ploth,plotw,ploth);
       fprintf(file_ptr,"SC %d,40,%d,40,2;\n",-HWIDTH,-HDEPTH);
       fprintf(file_ptr,"PA%d,%d;EA%d,%d;\n",-HWIDTH,-
HDEPTH,HWIDTH,HDEPTH);
       logo(2,2,string6,string8);
       penno=1;
       edge1= -HWIDTH+15;/* left edge */
       edge2= -HDEPTH+100;/* bottom */
       htmin=(int)heights[0];
       htmax=(int)heights[dcount-1];
       fprintf(file_ptr,"DI0,1;DT\3;LO16;PU%d,0;LB    Centre
Back\3\n",(int)-(edge1+10));
      fprintf(file_ptr,"DI0,1;DT\3;LO14;PU%d,0;LB    Centre
Front\3\n",(int)edge1+10);
/* here to plot horizontal lines */
      fprintf(file_ptr,"SP%d;\n",penno);
      fprintf(file_ptr,"CV1;\n");
        for(j=2;j<dcount-1;j++){
```

```
          height = (int)heights[j];
        fprintf(file_ptr,"DI1,0;DT\3;LO14;PU%d,%d;LB    %d\3
 \n",10,(int)(vdist[j][VLINES/2]+edge2),j+1);
        fprintf(file_ptr,"PU%d,%d;\n",edge1+34,(int)vdist[j][0]+edge2);
        fprintf(file_ptr,"PD");
           for(i=0;i<VLINES;i++){

 fprintf(file_ptr,"%d,%d,",(int)edge1+34+i*12,(int)vdist[j][i]+edge2);
          }
         }
 /* here to plot vertical lines */
       penno=2;
       fprintf(file_ptr,"SP%d;\n",penno);
       fprintf(file_ptr,"CV1;\n");
         for(i=0;i<VLINES;i++){
         if(i%5==0)fprintf(file_ptr,"LT6;");
         else fprintf(file_ptr,"LT;");
         fprintf(file_ptr,"PU%d,%d;\n",(int)edge1+34+i*12,(int)edge2);
         fprintf(file_ptr,"DI1,0;DT\3;LO16;LB%d\3 \n",i);

 fprintf(file_ptr,"PD%d,%d;\n",(int)edge1+34+i*12,(int)vdist[dcount-
1][i]+edge2);
        }
}
/* here to plot master ref */


masterplot()
{
      int i,j,penno,edge1,edge2,htmin,htmax,height;


      penno=3;
      edge1= -HWIDTH+15;/* left edge */
      edge2= -HDEPTH+100;/* bottom */
      htmin=(int)heights[0];
      htmax=(int)heights[dcount-1];

/* here to plot horizontal lines */
```

```
        fprintf(file_ptr,"SP%d;\n",penno);
        fprintf(file_ptr,"CV1;\n");
          for(j=2;j<dcount-1;j++){
          height = (int)heights[j];
        fprintf(file_ptr,"DI1,0;DT\3;LO14;PU%d,%d;LB%d.\3   \n",-
10,(int)(vdist[j][VLINES/2]+edge2),j+1);
        fprintf(file_ptr,"PU%d,%d;\n",edge1+34,(int)vdist[j][0]+edge2);
        fprintf(file_ptr,"PD");
           for(i=0;i<VLINES;i++){

fprintf(file_ptr,"%d,%d,",(int)edge1+34+i*12,(int)vdist[j][i]+edge2);
          }
         }
       fprintf(file_ptr,"CV0;\n");
        fprintf(file_ptr,"PU;PG;");
        fclose(file_ptr);
}
/* here to draw logo */

logo(page,maxpage,title,issue)
        int page,maxpage;
        char title[],issue[];
{
        int aa,bb,cc,dd,xedge,yedge,labelx,notex;

        xedge= -HWIDTH;/* left edge of drawing */
        labelx= HWIDTH-200;/* inside edge of label */
        yedge= -HDEPTH;/* bottom line of frame */
        notex= -HWIDTH+20; /* start of notes */

        aa= yedge;
        bb= yedge+50;
        cc= HWIDTH-200;
        dd= HWIDTH;

        fprintf(file_ptr,"SP1;PU%d,%d;EA%d,%d;\n",cc,bb,dd,aa);
        fprintf(file_ptr,"DI1,0;");
        aa=yedge +40;
```

```
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LBLoughborough
 Anthropometric Shadow Scanner\3 \n",labelx,aa);
        aa=yedge +25;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB%s\3
 \n",labelx,aa,title);
        aa=yedge +10;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LBPage %d of %d. Issue
 %s \3 \n",labelx,aa,page,maxpage,issue);
        aa=yedge +80;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB1. Use for vertical
 measurements only.\3 \n",notex,aa);
        aa=yedge +65;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB2. Red plot is average
 size 34b for reference.\3 \n",notex,aa);
        aa=yedge +50;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB3. Blue plot represents
 actual data.\3 \n",notex,aa);
        aa=yedge +35;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB4. Numbers refer to
 line numbers in data file.\3 \n",notex,aa);
}
/* here to get line into s, return length */

int getline(s,cnt,lim)
        char s[];
        int cnt,lim;
{
        Device val;
        long dev;
        int c,flg,i;

        flg=1; i=cnt;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
        case KEYBD:
        if(val=='\b'){
```

```
        i--;
        s[i]= '\0';
        c3s(whitevec);
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        s[i++]=val;
        }
        if(val=='\n')flg=0;
        putline(s,1.0,5.0);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
   return(i);
}
/* here to write a line to a window */

putline(s,x,y)
        char s[];
        Coord x,y;
{
        cmov2(x,y);
        charstr(s);
}
```

```
/* here to return a point from a Cardinal curve */
cardret(point,maxpoints,sliceno)
        int point,maxpoints,sliceno;
{
        int s,u;
        float frangle[4],temp[4],npoints,fracpnt,interval;


        apoly[0]=0;
        apoly[1]=0;
        apoly[2]=0;

        interval=NPTS-1;
        npoints = (float) maxpoints/interval;
        s= point/npoints;
        fracpnt = (point-s*npoints)/npoints;
        s=s-1;

        frangle[3]=1.0;
        frangle[2]=fracpnt;
        frangle[1]=frangle[2]*frangle[2];
        frangle[0]=frangle[1]*frangle[2];
        colmat(frangle,cardinal,temp);
          for(u=0;u<4;u++){
           if(s<0 && u==0){
           apoly[0]=apoly[0]-temp[0]*slices[sliceno][1][0];
           apoly[1]=apoly[1]+temp[0]*slices[sliceno][1][1];
           }
            else if ((u+s)>=0 && (u+s)<NPTS ){
           apoly[0]=apoly[0]+temp[u]*slices[sliceno][u+s][0];
           apoly[1]=apoly[1]+temp[u]*slices[sliceno][u+s][1];
           }
            else if((u+s)==NPTS){
           apoly[0]=apoly[0]-temp[3]*slices[sliceno][NPTS-2][0];
           apoly[1]=apoly[1]+temp[3]*slices[sliceno][NPTS-2][1];
           }
      } /* end of for u */
}
```

```
 /* here to create file with definate angles */

 fixfile()
 {
        int i,j;
        float angle;

          for(i=0;i<dcount;i++){
            for(j=0;j<VLINES;j++){
             angle=j*5;
            xyangle(angle,i);
            angfile[i][j]= fhypot(apoly[0],apoly[1]);
           } /* end of for j */
          } /* end of for i */
 }
 /* here to find vertical surface distance. */

 vertdist()
 {
        int i,j;
        float dist;

          for(j=0;j<VLINES;j++){
          vdist[0][j]=0;
           for(i=1;i<dcount;i++){
          dist= fhypot(angfile[i][j]-angfile[i-1][j],heights[i]-heights[i-1]);
          vdist[i][j]= vdist[i-1][j]+dist;
          } /* end of for i */

        } /* end of for j */
 }
 /* here to find x,y values for a particular angle from cubic curve */
 xyangle(angle,slice)
        float angle;
        int slice;
 {
        int i,j,k;
        float aa,bb,cc,xx,point;
```

```
        double artan();


        angle=(270-angle)*RAD;
  /*
  printf("angle=%f slice=%d\n",angle*180/PI,slice);
  */
        aa=0.0;
        bb=UMAX;
        cc=(bb-aa)/2;
           for(i=0;i<=DEF;i++){
           cardret((int)cc,UMAX,slice);
          xx = artan(apoly[1],apoly[0]);
          if(angle<xx)aa=cc;
          else bb=cc;
          cc=aa+(bb-aa)/2;
          }
 }
 /*matrix multiply 4 X 1 array times 4 X 4 array, A * B result in R */

 colmat(mata,matb,matr)
       float mata[4],matb[4][4],matr[4];
 {
       int i,j;
       float temp[4];

       for(i=0;i<4;i++){
       temp[i]=0;
         for(j=0;j<4;j++){
        temp[i]=temp[i]+mata[j]*matb[j][i];
        }
       }
      for(i=0;i<4;i++)matr[i]=temp[i];
}
/* here to print slice array */

pslice()
{
```

```
        int i,j;

        for(i=0;i<dcount;i++){
          for(j=0;j<=NPTS;j++){
                        printf("%d     %7.2f     %7.2f     %7.2f
",j,slices[i][j][0],slices[i][j][1],slices[i][j][2]);
           if(j%2==0)printf("\n");
          }
        printf("\n");
        }
}
/* here to print plotfile array */

arrprnt()
{
        int i,j;

        for(i=0;i<dcount;i++){
          for(j=0;j<38;j++){
            printf("%7.2f  ",plotfile[i][j]);
          if(j % 8==0 || j>=37)printf("\n");
          }
        printf("\n");
        }
}
/* here to ask a question & get an answer */

quest(infile,outfile)
        char infile[],outfile[];
{
        Device val;
        long dev;
        int c,flg,i;

        winset(textwin);
        c3s(whitevec);
        clear();
        c3s(blackvec);
```

```
cmov2(1.0,7.0);
charstr(infile);
cmov2(1.0,4.0);
flg=1; i=0;
while(flg==1){
while(qtest()){
dev=qread(&val);
switch(dev){
case KEYBD:
if(val=='\b'){
i--;
outfile[i]= '\0';
c3s(whitevec);
clear();
c3s(blackvec);
} /* end of if val */
else {
outfile[i++]=val;
}
if(val=='\n')flg=0;
winset(textwin);
c3s(whitevec);
clear();
c3s(blackvec);
cmov2(1.0,7.0);
charstr(infile);
cmov2(1.0,4.0);
charstr(outfile);
qreset();
break;
case ESCKEY:
greset();
gexit();
exit(0);
dglclose(-1);
break;
case RETKEY:
flg=0;
```

```
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
  return(i);
}
/* here to resolve atan calculations */

double artan(yy,xx)
        float yy,xx;

{
        float phi,pi2;

        pi2=2*PI;
        if(xx>0)phi=fatan(yy/xx);
        else if(xx<0)phi=PI+fatan(yy/xx);
        else if(xx==0 && yy>0)phi= PI/2;
        else if(xx==0 && yy<0)phi= 3*PI/2;
        else if(xx==0 && yy==0)phi= 0;
        if(phi<0)phi=pi2+phi;
        if(phi>pi2)phi=phi-pi2;
        return(float)(phi);
}
```

/*Isometric3.c*/

```
/* Creates a plotfile to give an isometric view
      on H.P.Plotter. Horizontal slices correspond to file slices.*/
#include <stdio.h>
#include "device.h"
#include <gl.h>
#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>
#define NPTS 16 /* number of points in curve fit polygon */
```

```
#define NFRAME 420 /* no. of vertical intervals in 2.1 metres
2100/VSTEP */
#define PATCHMAX 1600 /* size of patches array */
#define ANGLESTART 0 /* angle start of vertical scan */
#define ANGLESTOP 96 /* angle stop of vertical scan */
#define ANGLESTEP 4.8 /* angle increment of vertical scan */
#define VSTEP 5 /* increment of vertical scan */
#define NSLICES 50 /* maximum number of slices */
#define PI 3.141592654
#define PLOTUNIT 40 /* number of plotter units per mm. */
#define HWIDTH 350 /* half width of surrounding rectangle in mm's.
*/
#define HDEPTH 378 /* half height of surrounding rectangle in mm's.
*/


Coord geomxfr[4][4];
Coord geomyfr[4][4];
Coord geomzfr[4][4];
Coord tempxf[4][4];
Coord tempyf[4][4];
Coord tempzf[4][4];

float trans[4][4] = {
        { 1,  0,  0, 0 },
        { 0,  1,  0, 0 },
        { 0,  0,  1, 0 },
        { 0,  0,  0, 1 },
        };
Matrix cardinal = {
        {-0.5, 1.5,-1.5, 0.5},
        { 1.0,-2.5, 2.0,-0.5},
        {-0.5, 0.0, 0.5, 0.0},
        { 0.0, 1.0, 0.0, 0.0},
        };


        short blackvec[3] = { 0, 0, 0};
```

```
        short redvec[3] = { 255, 0, 0};
        short greenvec[3] = { 73, 206, 130};
        short bluevec[3] = { 0, 0, 255};
        short whitevec[3] = {255,255,255};
        long textwin;
        float patches[PATCHMAX][3][4][4];
        float DPOLY[NPTS+1][2];
        float slices[(ANGLESTOP-ANGLESTART)][NPTS+1][2];
        float vslice[(ANGLESTOP-ANGLESTART)][NFRAME][3];
        float heights[NSLICES];
        float apoly[3];
        float secangle;
        float set1,set2;
        int dcount,mflg,vcnt,hcnt,spangcnt;
        int patchcount;
        char *name;
        FILE *file_ptr;


main()
    {
        int i,step;
        float angle;

        inittextwin();
    qdevice(ESCKEY);
    qdevice(KEYBD);
    qdevice(RETKEY);
        slicelod();
        anglein();
        bslcptch(dcount);
        hcnt=0;
        vcnt=0;
        angle=ANGLESTART;
        step=(int)(ANGLESTEP+0.5);
        for(i=ANGLESTART;i<=ANGLESTOP;i=i+step){
        getvslice(angle);
        angle=angle+ANGLESTEP;
        }
```

```
        sliceplot();
        }

initextwin()
{
   prefposition(2*
XMAXSCREEN/5,4*XMAXSCREEN/5,4*YMAXSCREEN/10,5*YMAXS
CREEN/10);
    textwin = winopen("Slice plot.");
        RGBmode();
        shademodel(FLAT);
        ortho2(0,15,0,10);
        foreground();
        gconfig();
}
/* here to load slice data */

slicelod()
{
 FILE *fopen();
  char *string1,string2[50],*string3,string4[50];
   int count,i,j,k,m;
  float xx;

    string1 = "Enter file name ";
   string3 = "r";

        for(i=0;i<50;i++){
        string2[i]='\0';
        string4[i]='\0';
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }

   name = &string4[0];
```

```
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
   putline(string2,1.0,5.0);
   count = getline(string2,i,50);
   sscanf(&string2[i],"%s",name);
   winpush();
   file_ptr=fopen(name,string3);
        fscanf(file_ptr,"%d",&dcount);
        fscanf(file_ptr,"%d",&spangcnt);
          for(i=0;i<dcount;i++){
            for(j=0;j<=NPTS;j++){
             for(k=0;k<2;k++){
        fscanf(file_ptr,"%f",&xx);
        slices[i][j][k]=xx;
           } /* end of for k */
          } /* end of for j */
        } /* end of for i */
/* Here to input y value i.e., heights */
        for(m=0;m<dcount;m++){
          for(j=0;j<NPTS;j++){
         heights[m]=slices[m][NPTS][0];
          } /* end of for j */
        } /* end of for m */
        fclose(file_ptr);
        set1=slices[0][NPTS][0];
        set2=slices[dcount-1][NPTS][0];
}

/* here to create a plot file */

sliceplot()
{
 FILE *fopen();
  char *string1,string2[50],*string3,string4[50];
  char *string5,string6[50],*string7,string8[50];
```

```
int count,i,j,penno;
int plotw, ploth;



string1 = "Enter plot file name ";
string3 = "w";
string5 = "Enter title.";
string7 = "Enter issue No.";

    for(i=0;i<50;i++){
    string2[i]='\0';
    string4[i]='\0';
    string6[i]='\0';
    string8[i]='\0';
    }
    plotw = HWIDTH * PLOTUNIT;
    ploth = HDEPTH * PLOTUNIT;
    i=0;
    while(string1[i] !='\0'){
    string2[i]=string1[i];
    i++;
    }
    winset(textwin);
    winpop();
    c3s(whitevec);
    clear();
    c3s(blackvec);
    putline(string2,1.0,5.0);
    count = getline(string2,i,50);
    sscanf(&string2[i],"%s",string4);
    count=quest(string5,string6);
    count=quest(string7,string8);
    winpush();
    file_ptr=fopen(string4,string3);
   fprintf(file_ptr,"IN;SP1;PA;FS6;\n");
    fprintf(file_ptr,"IP%d,%d,%d,%d;\n",-plotw,-ploth,plotw,ploth);
    fprintf(file_ptr,"SC  %d,40,%d,40,2;\n",-HWIDTH,-HDEPTH);
```

```
        fprintf(file_ptr,"PA%d,%d;EA%d,%d;\n",-HWIDTH,-
  HDEPTH,HWIDTH,HDEPTH);
        logo(1,1,string6,string8);
  /* here to plot right side */
        for(j=0;j<hcnt;j++){
        if(j%5==0){
          fprintf(file_ptr,"LT6;");
         penno = 3;
         fprintf(file_ptr,"SP%d;\n",penno);
         }
        else {
          fprintf(file_ptr,"LT;");
         penno = 2;
         fprintf(file_ptr,"SP%d;\n",penno);
         }
       fprintf(file_ptr,"PU%8.2f,%8.2f;",vslice[j][0][0],vslice[j][0][2]);
       fprintf(file_ptr,"DI1,0;DT\3;LO16;LB%d\3 \n",j);
         for(i=0;i<vcnt;i++){
         fprintf(file_ptr,"PD%8.2f,%8.2f;",vslice[j][i][0],vslice[j][i][2]);
          if(i%5==0)fprintf(file_ptr,"\n");
         }
       }
       fprintf(file_ptr,"\n");
 /* here to plot left side */
       for(j=1;j<hcnt;j++){
       if(j%5==0){
         fprintf(file_ptr,"LT6;");
        penno = 3;
        fprintf(file_ptr,"SP%d;\n",penno);
        }
       else {
         fprintf(file_ptr,"LT;");
        penno = 2;
        fprintf(file_ptr,"SP%d;\n",penno);
        }
      fprintf(file_ptr,"PU%8.2f,%8.2f;",-vslice[j][0][0],vslice[j][0][2]);
      fprintf(file_ptr,"DI1,0;DT\3;LO16;LB%d\3 \n",j);
        for(i=0;i<vcnt;i++){
```

```
        fprintf(file_ptr,"PD%8.2f,%8.2f;",-vslice[j][i][0],vslice[j][i][2]);
         if(i%5==0)fprintf(file_ptr,"\n");
        }
       }
       fprintf(file_ptr,"\n");
/* here to plot slices */
       penno = 1;
       fprintf(file_ptr,"SP%d;\n",penno);
       fprintf(file_ptr,"LT;");
       for(i=2;i<vcnt-1;i++){
       fprintf(file_ptr,"PU%8.2f,%8.2f;",vslice[0][i][0],vslice[0][i][2]);
       fprintf(file_ptr,"DI1,0;DT\3;LO14;LB%d\3 \n",i+1);
         for(j=0;j<hcnt;j++){
        fprintf(file_ptr,"PD%8.2f,%8.2f;",vslice[j][i][0],vslice[j][i][2]);
         if(j%5==0)fprintf(file_ptr,"\n");
        }
         fprintf(file_ptr,"\n");
         fprintf(file_ptr,"PU%8.2f,%8.2f;",-vslice[hcnt-1][i][0],vslice[hcnt-
1][i][2]);
       for(j=hcnt-1;j>=0;j--){
        fprintf(file_ptr,"PD%8.2f,%8.2f;",-vslice[j][i][0],vslice[j][i][2]);
         if(j%5==0)fprintf(file_ptr,"\n");
        }
       }
       fprintf(file_ptr,"\n");
       fprintf(file_ptr,"PU;PG;SP0;");
       fclose(file_ptr);
}
/* here to get line into s, return length */

int getline(s,cnt,lim)
       char s[];
       int cnt,lim;
{
       Device val;
       long dev;
       int flg,i;
```

```
        flg=1; i=cnt;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
        case KEYBD:
        if(val=='\b'){
        i--;
        s[i]= '\0';
        c3s(whitevec);
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        s[i++]=val;
        }
        if(val=='\n')flg=0;
        putline(s,1.0,5.0);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
  return(i);
}
/* here to write a line to a window */
```

```
putline(s,x,y)
        char s[];
        Coord x,y;
{
        cmov2(x,y);
        charstr(s);
}
/* here to return a point from a Cardinal curve */
cardret(point,maxpoints)
        int point,maxpoints;
{
        int s,u;
        float frangle[4],temp[4],npoints,fracpnt,interval;


        apoly[0]=0;
        apoly[1]=0;
        apoly[2]=0;

        interval=NPTS-1;
        npoints = (float) maxpoints/interval;
        s= (int)point/npoints;
        fracpnt = (point-s*npoints)/npoints;
        s=s-1;

        frangle[3]=1.0;
        frangle[2]=fracpnt;
        frangle[1]=frangle[2]*frangle[2];
        frangle[0]=frangle[1]*frangle[2];

        colmat(frangle,cardinal,temp);

         for(u=0;u<4;u++){
          if((u+s)<0){
          apoly[0]=apoly[0]-temp[0]*DPOLY[1][0];
          apoly[1]=apoly[1]+temp[0]*DPOLY[1][1];
          }
          else if ((u+s)>=0 && (u+s)<NPTS ){
```

```
        apoly[0]=apoly[0]+temp[u]*DPOLY[u+s][0];
        apoly[1]=apoly[1]+temp[u]*DPOLY[u+s][1];
        }
         else if((u+s)>=NPTS){
        apoly[0]=apoly[0]-temp[3]*DPOLY[NPTS-2][0];
        apoly[1]=apoly[1]+temp[3]*DPOLY[NPTS-2][1];
        }
      } /* end of for u */
      apoly[2]=DPOLY[NPTS][0];
}

/*matrix multiply 4 X 1 array times 4 X 4 array, A * B result in R */

colmat(mata,matb,matr)
      float mata[4],matb[4][4],matr[4];
{
      int i,j;
      float temp[4];

    . for(i=0;i<4;i++){
      temp[i]=0;
       for(j=0;j<4;j++){
      temp[i]=temp[i]+mata[j]*matb[j][i];
       }
      }
      for(i=0;i<4;i++)matr[i]=temp[i];
}
/* here to print slice array */

pslice()
{
      int i,j;

      for(i=0;i<dcount;i++){
        for(j=0;j<11;j++){
         printf("%7.2f %7.2f  ",slices[i][j][0],slices[i][j][1]);
        if(j==3 || j==7 || j==10)printf("\n");
       }
```

```
        }
}
/* here to create a vertical slice file from patch data */

getvslice(intang)
        float intang;
{
        int i,j,resltn,step;
        float xx,yy,zz;
        float angle,phi;
        float artan(),sintheta,costheta;


        sintheta = sin(secangle);
        costheta = cos(secangle);
        angle=(float)intang*PI/180;
        resltn=750;
        vcnt=0;
        for(i=0;i<dcount;i++){
        getpoly(heights[i]);
        step=0;
          for(j=0;j<resltn;j++){
         cardret(j,resltn);
         xx=apoly[0];
         yy=apoly[1];
         zz=apoly[2] - ((set1 + set2)/2);
         phi= (float)artan(yy,xx);
           if(phi>=angle-0.009 && phi<angle+0.009){
                vslice[hcnt][vcnt][0]=xx;
                vslice[hcnt][vcnt][1]=zz*sintheta + yy*costheta;
                vslice[hcnt][vcnt][2]=zz*costheta - yy*sintheta;
           vcnt++;
           step++;
           }
           if(step>0)break;
         } /* end of for j */
        } /* end of for i */
        hcnt++;
```

```
printf("angle=%f hcnt=%d vcnt=%d\n",intang,hcnt,vcnt);
}
/* here to get slice polygon from patch data */

getpoly(height)
        float height;
{
        int i,j,k,count;
        float htmin,htmax,w,ww,www,maxmin;

        count=0;
        for(i=0;i<patchcount;i++){
        htmin = patches[i][1][1][1];
        htmax = patches[i][1][2][1];

        if(height>=htmin && height<=htmax)break;
        } /* end of for i */
        count=i;
          maxmin=htmax-htmin;
         if(maxmin!=0) w =(height-htmin)/maxmin;
         else w=0;
         ww = w*w;
        www = ww*w;

       for(i=1;i<NPTS;i++){
         for(j=0;j<4;j++){
          for(k=0;k<4;k++){
         geomxfr[k][j] = patches[count][0][k][j];
         geomyfr[k][j] = patches[count][1][k][j];
         geomzfr[k][j] = -patches[count][2][k][j];
          }
         }
       matmult(cardinal,geomxfr,tempxf);
       matmult(cardinal,geomyfr,tempyf);
       matmult(cardinal,geomzfr,tempzf);

       matmult(tempxf,trans,tempxf);
       matmult(tempyf,trans,tempyf);
```

```
        matmult(tempzf,trans,tempzf);

        DPOLY[i][0]=www*tempxf[0][1]+ww*tempxf[1][1]+w*tempxf[2][1]+
tempxf[3][1];
        DPOLY[i][1]=www*tempzf[0][1]+ww*tempzf[1][1]+w*tempzf[2][1]+
tempzf[3][1];

        if(i==1){
        DPOLY[0][0]=www*tempxf[0][2]+ww*tempxf[1][2]+w*tempxf[2][2]
+tempxf[3][2];
        DPOLY[0][1]=www*tempzf[0][2]+ww*tempzf[1][2]+w*tempzf[2][2]
+tempzf[3][2];
        }
        count++;
        } /* end of for i */
        DPOLY[NPTS][0]=www*tempyf[0][1]+ww*tempyf[1][1]+w*tempyf[
2][1]+tempyf[3][1];
}
/* here to input angle */

anglein()
{
        char *string1,string2[50],string3[50];
        int i;

        string1 = "Enter angle of section.";
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
        for(i=0;i<50;i++){
        string2[i]='\0';
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
```

```
        }
    putline(string2,1.0,5.0);
    getline(string2,i,50);
    sscanf(&string2[i],"%s",string3);
    secangle=atof(string3);
   secangle= -secangle*PI/180;
    winpush();
}
/* here to print vslice */

vslprnt()
{
        int i,j;

        for(i=0;i<vcnt;i++){
          for(j=0;j<=(ANGLESTOP-ANGLESTART);j++){
          printf("vslice[%d][%d][0]=%f\n",j,i,vslice[j][i][0]);
          printf("vslice[%d][%d][1]=%f\n",j,i,vslice[j][i][1]);
          printf("vslice[%d][%d][2]=%f\n",j,i,vslice[j][i][2]);
         }
        }
}
/* matrix multiply of 4 X 4 array , A * B result in R*/

matmult(mata,matb,matr)
        float mata[4][4],matb[4][4],matr[4][4];
{
        int i,j,k;
        float temp[4][4];

        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
          temp[i][j]=0;
            for(k=0;k<4;k++){
            temp[i][j]=temp[i][j]+mata[i][k]*matb[k][j];
            }
          }
        }
```

```
        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
         matr[i][j]=temp[i][j];
          }
        }
}
/* here to make large patches (total horiz. body section ) from 'n'
bspline slices */

bslcptch(count)
        int count;
{
        int i,j,k,l,m;


        for(j=0;j<(count-3);j=j+1){
          for(k= -1;k<NPTS-2;k++){
           for(i=0;i<4;i++){
           getslice(i+j);
           for(l=0;l<4;l++){
           m=k+l;
           if(m>=NPTS){
                tempxf[i][3-l]= -DPOLY[m-2][0];
                tempzf[i][3-l]= -DPOLY[m-2][1];
                }
           else if(m<0){
                tempxf[i][3-l]= -DPOLY[-m][0];
                tempzf[i][3-l]= -DPOLY[-m][1];
                }
           else {
                tempxf[i][3-l]= DPOLY[m][0];
                tempzf[i][3-l]= -DPOLY[m][1];
                }
           tempyf[i][3-l]=DPOLY[NPTS][0];
           } /* end of for l */
          } /* end of for i */
        pstore();
        } /* end of for k */
```

```
        } /* end of for j */
 /* now to make patch of mirror image */
  if(mflg==1){
        for(j=0;j<(count-3);j=j+1){
          for(k= -1;k<(NPTS-2);k++){
           for(i=0;i<4;i++){
           getslice(i+j);
           for(l=0;l<4;l++){
           m=k+l;
           if(m>=NPTS){
                tempxf[i][l]= DPOLY[m-2][0];
                tempzf[i][l]= -DPOLY[m-2][1];
                }
           else if(m<0){
                tempxf[i][l]= DPOLY[-m][0];
                tempzf[i][l]= -DPOLY[-m][1];
                }
           else {
                tempxf[i][l]= -DPOLY[m][0];
                tempzf[i][l]= -DPOLY[m][1];
                }
           tempyf[i][l]=DPOLY[NPTS][0];
           }
          }
        pstore();
        }
        }
 }
}
/* here to retreive horizontal patch slices */

getslice(count)
        int count;
{
        int i;

        for(i=0;i<NPTS;i++){
          DPOLY[i][0]= slices[count][i][0];
```

```
        DPOLY[i][1]= slices[count][i][1];
      }
      DPOLY[NPTS][0] = slices[count][NPTS][0];
}
/* here to store a patch in memory */

pstore()
{
      int i,j;

      for(i=0;i<4;i++){
        for(j=0;j<4;j++){
       patches[patchcount][0][i][j]=tempxf[i][j];
       patches[patchcount][1][i][j]=tempyf[i][j];
       patches[patchcount][2][i][j]=tempzf[i][j];
       }
      }
      patchcount++;
}
/* here to resolve atan calculations */

float artan(yy,xx)
      float yy,xx;
{
      float phi,pi2;

      pi2=2*PI;
      if(xx>0)phi=fatan(yy/xx);
      else if(xx<0)phi=PI+fatan(yy/xx);
      else if(xx==0 && yy>0)phi= PI/2;
      else if(xx==0 && yy<0)phi= 3*PI/2;
      else if(xx==0 && yy==0)phi= 0;
      phi=3*PI/2-phi; /* not in normal artan routine */
      if(phi<0)phi=pi2+phi;
      if(phi>pi2)phi=phi-pi2;
      return(phi);
}
/* here to draw logo */
```

```
logo(page,maxpage,title,issue)
        int page,maxpage;
        char title[],issue[];
{
        int aa,bb,cc,dd,xedge,yedge,labelx,notex;

        xedge= -HWIDTH;/* left edge of drawing */
        labelx= HWIDTH-200;/* inside edge of label */
        yedge= -HDEPTH;/* bottom line of frame */
        notex= -HWIDTH+20; /* start of notes */

        aa= yedge;
        bb= yedge+50;
        cc= HWIDTH-200;
        dd= HWIDTH;

        fprintf(file_ptr,"SP1;PU%d,%d;EA%d,%d;\n",cc,bb,dd,aa);
        fprintf(file_ptr,"DI1,0;");
        aa=yedge +40;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LBLoughborough
Anthropometric Shadow Scanner\3 \n",labelx,aa);
        aa=yedge +25;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB%s\3
\n",labelx,aa,title);
        aa=yedge +10;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LBPage %d of %d. Issue
%s \3 \n",labelx,aa,page,maxpage,issue);
}
/* here to ask a question & get an answer */

quest(infile,outfile)
        char infile[],outfile[];
{
        Device val;
        long dev;
        int flg,i;

        winset(textwin);
```

```
c3s(whitevec);
clear();
c3s(blackvec);
cmov2(1.0,7.0);
charstr(infile);
cmov2(1.0,4.0);
flg=1; i=0;
while(flg==1){
while(qtest()){
dev=qread(&val);
switch(dev){
case KEYBD:
if(val=='\b'){
i--;
outfile[i]= '\0';
c3s(whitevec);
clear();
c3s(blackvec);
} /* end of if val */
else {
outfile[i++]=val;
}
if(val=='\n')flg=0;
winset(textwin);
c3s(whitevec);
clear();
c3s(blackvec);
cmov2(1.0,7.0);
charstr(infile);
cmov2(1.0,4.0);
charstr(outfile);
qreset();
break;
case ESCKEY:
greset();
gexit();
exit(0);
dglclose(-1);
```

```
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
  return(i);
}
```

/* isometric4.c */

```
/* Creates a plotfile to give an isometric view
      on H.P.Plotter. Horizontal slices correspond to file slices.*/
#include <stdio.h>
#include "device.h"
#include <gl.h>
#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>
#define NPTS 16 /* number of points in curve fit polygon */
#define NFRAME 420 /* no. of vertical intervals in 2.1 metres
2100/VSTEP */
#define PATCHMAX 1600 /* size of patches array */
#define ANGLESTART 91 /* angle start of vertical scan */
#define ANGLESTOP 188 /* angle stop of vertical scan */
#define ANGLESTEP 4.8 /* angle increment of vertical scan */
#define VSTEP 5 /* increment of vertical scan */
#define NSLICES 50 /* maximum number of slices */
#define PI 3.141592654
#define PLOTUNIT 40 /* number of plotter units per mm. */
#define HWIDTH 350 /* half width of surrounding rectangle in mm's.
*/
#define HDEPTH 378 /* half height of surrounding rectangle in mm's.
*/
```

```
Coord geomxfr[4][4];
Coord geomyfr[4][4];
Coord geomzfr[4][4];
Coord tempxf[4][4];
Coord tempyf[4][4];
Coord tempzf[4][4];

float trans[4][4] = {
        { 1,  0,  0, 0 },
        { 0,  1,  0, 0 },
        { 0,  0,  1, 0 },
        { 0,  0,  0, 1 },
        };
Matrix cardinal = {
        {-0.5, 1.5,-1.5, 0.5},
        { 1.0,-2.5, 2.0,-0.5},
        {-0.5, 0.0, 0.5, 0.0},
        { 0.0, 1.0, 0.0, 0.0},
        };


        short blackvec[3] = { 0, 0, 0};
        short redvec[3] = { 255, 0, 0};
        short greenvec[3] = { 73, 206, 130};
        short bluevec[3] = { 0, 0, 255};
        short whitevec[3] = {255,255,255};
        long textwin;
        float patches[PATCHMAX][3][4][4];
        float DPOLY[NPTS+1][2];
        float slices[(ANGLESTOP-ANGLESTART)][NPTS+1][2];
        float vslice[(ANGLESTOP-ANGLESTART)][NFRAME][3];
        float heights[NSLICES];
        float apoly[3];
        float secangle;
        float set1,set2;
        int dcount,mflg,vcnt,hcnt,spangcnt;
        int patchcount;
```

```
        char *name;
        FILE *file_ptr;


main()
        {
        int i,step;
        float angle;

        inittextwin();
   qdevice(ESCKEY);
   qdevice(KEYBD);
   qdevice(RETKEY);
        slicelod();
        anglein();
        bslcptch(dcount);
        hcnt=0;
        vcnt=0;
        angle=(float)ANGLESTART+0.2;
        step=(int)(ANGLESTEP+0.5);
        for(i=ANGLESTART;i<=ANGLESTOP;i=i+step){
        getvslice(angle);
        angle=angle+ANGLESTEP;
        }
        sliceplot();
        }


inittextwin()
{
   prefposition(2*
XMAXSCREEN/5,4*XMAXSCREEN/5,4*YMAXSCREEN/10,5*YMAXS
CREEN/10);
   textwin = winopen("Slice plot.");
        RGBmode();
        shademodel(FLAT);
        ortho2(0,15,0,10);
        foreground();
        gconfig();
}
```

```
/* here to load slice data */

slicelod()
{
 FILE *fopen();
  char *string1,string2[50],*string3,string4[50];
  int count,i,j,k,m;
 float xx;

  string1 = "Enter file name ";
 string3 = "r";

       for(i=0;i<50;i++){
       string2[i]='\0';
       string4[i]='\0';
       }
       i=0;
       while(string1[i] !='\0'){
       string2[i]=string1[i];
       i++;
       }

 name = &string4[0];
       winset(textwin);
       winpop();
       c3s(whitevec);
       clear();
       c3s(blackvec);
 putline(string2,1.0,5.0);
 count = getline(string2,i,50);
 sscanf(&string2[i],"%s",name);
 winpush();
file_ptr=fopen(name,string3);
      fscanf(file_ptr,"%d",&dcount);
      fscanf(file_ptr,"%d",&spangcnt);
        for(i=0;i<dcount;i++){
          for(j=0;j<=NPTS;j++){
           for(k=0;k<2;k++){
```

```
        fscanf(file_ptr,"%f",&xx);
        slices[i][j][k]=xx;
          } /* end of for k */
         } /* end of for j */
        } /* end of for i */
/* Here to input y value i.e., heights */
        for(m=0;m<dcount;m++){
          for(j=0;j<NPTS;j++){
         heights[m]=slices[m][NPTS][0];
         } /* end of for j */
        } /* end of for m */
        fclose(file_ptr);
        set1=slices[0][NPTS][0];
        set2=slices[dcount-1][NPTS][0];
}

/* here to create a plot file */

sliceplot()
{
 FILE *fopen();
  char *string1,string2[50],*string3,string4[50];
  char *string5,string6[50],*string7,string8[50];
   int count,i,j,penno;
  int plotw, ploth;



  string1 = "Enter plot file name ";
  string3 = "w";
  string5 = "Enter title.";
  string7 = "Enter issue No.";

      for(i=0;i<50;i++){
      string2[i]='\0';
      string4[i]='\0';
      string6[i]='\0';
      string8[i]='\0';
```

```
        }
        plotw = HWIDTH * PLOTUNIT;
        ploth = HDEPTH * PLOTUNIT;
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
        putline(string2,1.0,5.0);
        count = getline(string2,i,50);
        sscanf(&string2[i],"%s",string4);
        count=quest(string5,string6);
        count=quest(string7,string8);
        winpush();
        file_ptr=fopen(string4,string3);
      fprintf(file_ptr,"IN;SP1;PA;FS6;\n");
       fprintf(file_ptr,"IP%d,%d,%d,%d;\n",-plotw,-ploth,plotw,ploth);
      fprintf(file_ptr,"SC  %d,40,%d,40,2;\n",-HWIDTH,-HDEPTH);
      fprintf(file_ptr,"PA%d,%d;EA%d,%d;\n",-HWIDTH,-
HDEPTH,HWIDTH,HDEPTH);
      logo(1,1,string6,string8);
/* here to plot right side */
      for(j=0;j<hcnt;j++){
      if((j+19)%5==0){
        fprintf(file_ptr,"LT6;");
       penno = 3;
       fprintf(file_ptr,"SP%d;\n",penno);
       }
      else {
        fprintf(file_ptr,"LT;");
       penno = 2;
      fprintf(file_ptr,"SP%d;\n",penno);
       }
```

```
        fprintf(file_ptr,"PU%8.2f,%8.2f;",vslice[j][0][0],vslice[j][0][2]);
        fprintf(file_ptr,"DI1,0;DT\3;LO16;LB%d\3 \n",j+19);
          for(i=0;i<vcnt;i++){
         fprintf(file_ptr,"PD%8.2f,%8.2f;",vslice[j][i][0],vslice[j][i][2]);
          if(i%5==0)fprintf(file_ptr,"\n");
         }
        }
        fprintf(file_ptr,"\n");
/* here to plot left side */
        for(j=0;j<hcnt;j++){
        if((j+19)%5==0){
          fprintf(file_ptr,"LT6;");
         penno = 3;
         fprintf(file_ptr,"SP%d;\n",penno);
         }
        else {
          fprintf(file_ptr,"LT;");
         penno = 2;
         fprintf(file_ptr,"SP%d;\n",penno);
         }
       fprintf(file_ptr,"PU%8.2f,%8.2f;",-vslice[j][0][0],vslice[j][0][2]);
       fprintf(file_ptr,"DI1,0;DT\3;LO16;LB%d\3 \n",j+19);
         for(i=0;i<vcnt;i++){
         fprintf(file_ptr,"PD%8.2f,%8.2f;",-vslice[j][i][0],vslice[j][i][2]);
          if(i%5==0)fprintf(file_ptr,"\n");
         }
        }
        fprintf(file_ptr,"\n");
/* here to plot slices */
       penno = 1;
       fprintf(file_ptr,"SP%d;\n",penno);
       fprintf(file_ptr,"LT;");
       for(i=2;i<vcnt-1;i++){
       fprintf(file_ptr,"PU%8.2f,%8.2f;",vslice[0][i][0],vslice[0][i][2]);
       fprintf(file_ptr,"DI1,0;DT\3;LO14;LB%d\3 \n",i+1);
         for(j=0;j<hcnt;j++){
         fprintf(file_ptr,"PD%8.2f,%8.2f;",vslice[j][i][0],vslice[j][i][2]);
          if(j%5==0)fprintf(file_ptr,"\n");
```

```
        }
         fprintf(file_ptr,"\n");
         fprintf(file_ptr,"PU%8.2f,%8.2f;",-vslice[hcnt-1][i][0],vslice[hcnt-
1][i][2]);
         for(j=hcnt-1;j>=0;j--){
         fprintf(file_ptr,"PD%8.2f,%8.2f;",-vslice[j][i][0],vslice[j][i][2]);
         if(j%5==0)fprintf(file_ptr,"\n");
        }
       }
       fprintf(file_ptr,"\n");
       fprintf(file_ptr,"PU;PG;SP0;");
       fclose(file_ptr);
}
/* here to get line into s, return length */

int getline(s,cnt,lim)
       char s[];
       int cnt,lim;
{
       Device val;
       long dev;
       int flg,i;

       flg=1; i=cnt;
       while(flg==1){
       while(qtest()){
       dev=qread(&val);
       switch(dev){
       case KEYBD:
       if(val=='\b'){
       i--;
       s[i]= '\0';
       c3s(whitevec);
       clear();
       c3s(blackvec);
       } /* end of if val */
       else {
       s[i++]=val;
```

```
        }
        if(val=='\n')flg=0;
        putline(s,1.0,5.0);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
  return(i);
}
/* here to write a line to a window */

putline(s,x,y)
        char s[];
        Coord x,y;
{
        cmov2(x,y);
        charstr(s);
}
/* here to return a point from a Cardinal curve */

cardret(point,maxpoints)
        int point,maxpoints;
{
        int s,u;
        float frangle[4],temp[4],npoints,fracpnt,interval;
```

```
        apoly[0]=0;
        apoly[1]=0;
        apoly[2]=0;


        interval=NPTS-1;
        npoints = (float) maxpoints/interval;
        s= (int)point/npoints;
        fracpnt = (point-s*npoints)/npoints;
        s=s-1;

        frangle[3]=1.0;
        frangle[2]=fracpnt;
        frangle[1]=frangle[2]*frangle[2];
        frangle[0]=frangle[1]*frangle[2];

        colmat(frangle,cardinal,temp);

          for(u=0;u<4;u++){
           if((u+s)<0){
          apoly[0]=apoly[0]-temp[0]*DPOLY[1][0];
          apoly[1]=apoly[1]+temp[0]*DPOLY[1][1];
          }
           else if ((u+s)>=0 && (u+s)<NPTS ){
          apoly[0]=apoly[0]+temp[u]*DPOLY[u+s][0];
          apoly[1]=apoly[1]+temp[u]*DPOLY[u+s][1];
          }
           else if((u+s)>=NPTS){
          apoly[0]=apoly[0]-temp[3]*DPOLY[NPTS-2][0];
          apoly[1]=apoly[1]+temp[3]*DPOLY[NPTS-2][1];
          }
        } /* end of for u */
        apoly[2]=DPOLY[NPTS][0];
}

/*matrix multiply 4 X 1 array times 4 X 4 array, A * B result in R */
```

```
colmat(mata,matb,matr)
        float mata[4],matb[4][4],matr[4];
{
        int i,j;
        float temp[4];

        for(i=0;i<4;i++){
        temp[i]=0;
          for(j=0;j<4;j++){
         temp[i]=temp[i]+mata[j]*matb[j][i];
         }
        }
        for(i=0;i<4;i++)matr[i]=temp[i];
}
/* here to print slice array */

pslice()
{
        int i,j;

        for(i=0;i<dcount;i++){
          for(j=0;j<11;j++){
           printf("%7.2f %7.2f  ",slices[i][j][0],slices[i][j][1]);
          if(j==3 || j==7 || j==10)printf("\n");
         }
        }
}
/* here to create a vertical slice file from patch data */

getvslice(intang)
        float intang;
{
        int i,j,resltn,step;
        float xx,yy,zz;
        float angle,phi;
        float artan(),sintheta,costheta;
```

```
        sintheta = sin(secangle);
        costheta = cos(secangle);
        angle=(float)intang*PI/180;
        if(intang>=180)angle=PI;
        resltn=750;
        vcnt=0;
        for(i=0;i<dcount;i++){
        getpoly(heights[i]);
        step=0;
          for(j=0;j<=resltn;j++){
          cardret(j,resltn);
         xx=apoly[0];
         yy=apoly[1];
         zz=apoly[2] - ((set1 + set2)/2);
         phi= (float)artan(yy,xx);
           if(phi>=angle-0.009 && phi<angle+0.009){
                vslice[hcnt][vcnt][0]=xx;
                vslice[hcnt][vcnt][1]=zz*sintheta + yy*costheta;
                vslice[hcnt][vcnt][2]=zz*costheta - yy*sintheta;
           vcnt++;
           step++;
           }
           if(step>0)break;
         } /* end of for j */
       } /* end of for i */
       hcnt++;
printf("angle=%f hcnt=%d vcnt=%d\n",angle*180/PI,hcnt,vcnt);
}
/* here to get slice polygon from patch data */


getpoly(height)
       float height;
{
       int i,j,k,count;
       float htmin,htmax,w,ww,www,maxmin;

       count=0;
       for(i=0;i<patchcount;i++){
```

```
htmin = patches[i][1][1][1];
htmax = patches[i][1][2][1];

if(height>=htmin && height<=htmax)break;
} /* end of for i */
count=i;
  maxmin=htmax-htmin;
 if(maxmin!=0) w =(height-htmin)/maxmin;
 else w=0;
 ww = w*w;
www = ww*w;

for(i=1;i<NPTS;i++){
  for(j=0;j<4;j++){
   for(k=0;k<4;k++){
  geomxfr[k][j] = patches[count][0][k][j];
  geomyfr[k][j] = patches[count][1][k][j];
  geomzfr[k][j] = -patches[count][2][k][j];
  }
 }
matmult(cardinal,geomxfr,tempxf);
matmult(cardinal,geomyfr,tempyf);
matmult(cardinal,geomzfr,tempzf);

matmult(tempxf,trans,tempxf);
matmult(tempyf,trans,tempyf);
matmult(tempzf,trans,tempzf);

DPOLY[i][0]=www*tempxf[0][1]+ww*tempxf[1][1]+w*tempxf[2][1]+
tempxf[3][1];
DPOLY[i][1]=www*tempzf[0][1]+ww*tempzf[1][1]+w*tempzf[2][1]+
tempzf[3][1];

if(i==1){
DPOLY[0][0]=www*tempxf[0][2]+ww*tempxf[1][2]+w*tempxf[2][2]
+tempxf[3][2];
DPOLY[0][1]=www*tempzf[0][2]+ww*tempzf[1][2]+w*tempzf[2][2]
+tempzf[3][2];
```

```
        }
        count++;
        } /* end of for i */
        DPOLY[NPTS][0]=www*tempyf[0][1]+ww*tempyf[1][1]+w*tempyf[
 2][1]+tempyf[3][1];
 }
 /* here to input angle */

 anglein()
 {
        char *string1,string2[50],string3[50];
        int i;

        string1 = "Enter angle of section.";
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
        for(i=0;i<50;i++){
        string2[i]='\0';
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }
  putline(string2,1.0,5.0);
  getline(string2,i,50);
  sscanf(&string2[i],"%s",string3);
  secangle=atof(string3);
  secangle= -secangle*PI/180;
  winpush();
}
/* here to print vslice */

vslprnt()
{
```

```
        int i,j;

        for(i=0;i<vcnt;i++){
          for(j=0;j<=(ANGLESTOP-ANGLESTART);j++){
          printf("vslice[%d][%d][0]=%f\n",j,i,vslice[j][i][0]);
          printf("vslice[%d][%d][1]=%f\n",j,i,vslice[j][i][1]);
          printf("vslice[%d][%d][2]=%f\n",j,i,vslice[j][i][2]);
          }
        }
}
/* matrix multiply of 4 X 4 array , A * B result in R*/

matmult(mata,matb,matr)
        float mata[4][4],matb[4][4],matr[4][4];
{
        int i,j,k;
        float temp[4][4];

        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
          temp[i][j]=0;
            for(k=0;k<4;k++){
          temp[i][j]=temp[i][j]+mata[i][k]*matb[k][j];
            }
          }
        }
        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
          matr[i][j]=temp[i][j];
          }
        }
}
/* here to make large patches (total horiz. body section ) from 'n'
bspline slices */

bslcptch(count)
        int count;
{
```

```
int i,j,k,l,m;


for(j=0;j<(count-3);j=j+1){'
  for(k= -1;k<NPTS-2;k++){
   for(i=0;i<4;i++){
   getslice(i+j);
   for(l=0;l<4;l++){
   m=k+l;
   if(m>=NPTS){
        tempxf[i][3-l]= -DPOLY[m-2][0];
        tempzf[i][3-l]= -DPOLY[m-2][1];
        }
   else if(m<0){
        tempxf[i][3-l]= -DPOLY[-m][0];
        tempzf[i][3-l]= -DPOLY[-m][1];
        }
   else {
        tempxf[i][3-l]= DPOLY[m][0];
        tempzf[i][3-l]= -DPOLY[m][1];
        }
   tempyf[i][3-l]=DPOLY[NPTS][0];
   } /* end of for l */
  } /* end of for i */
 pstore();
 } /* end of for k */
 } /* end of for j */
/* now to make patch of mirror image */
 if(mflg==1){
 for(j=0;j<(count-3);j=j+1){
  for(k= -1;k<(NPTS-2);k++){
   for(i=0;i<4;i++){
   getslice(i+j);
   for(l=0;l<4;l++){
   m=k+l;
   if(m>=NPTS){
        tempxf[i][l]= DPOLY[m-2][0];
        tempzf[i][l]= -DPOLY[m-2][1];
```

```
            }
          else if(m<0){
            tempxf[i][l]= DPOLY[-m][0];
            tempzf[i][l]= -DPOLY[-m][1];
            }
          else {
            tempxf[i][l]= -DPOLY[m][0];
            tempzf[i][l]= -DPOLY[m][1];
            }
          tempyf[i][l]=DPOLY[NPTS][0];
          }
        }
       pstore();
       }
       }
 }
}
/* here to retreive horizontal patch slices */

getslice(count)
    int count;
{
       int i;

       for(i=0;i<NPTS;i++){
         DPOLY[i][0]= slices[count][i][0];
         DPOLY[i][1]= slices[count][i][1];
       }
       DPOLY[NPTS][0] = slices[count][NPTS][0];
}
/* here to store a patch in memory */

pstore()
{
      int i,j;

      for(i=0;i<4;i++){
        for(j=0;j<4;j++){
```

```
            patches[patchcount][0][i][j]=tempxf[i][j];
            patches[patchcount][1][i][j]=tempyf[i][j];
            patches[patchcount][2][i][j]=tempzf[i][j];
            }
          }
          patchcount++;
}
/* here to resolve atan calculations */

float artan(yy,xx)
          float yy,xx;
{
          float phi,pi2;

          pi2=2*PI;
          if(xx>0)phi=fatan(yy/xx);
          else if(xx<0)phi=PI+fatan(yy/xx);
          else if(xx==0 && yy>0)phi= PI/2;
          else if(xx==0 && yy<0)phi= 3*PI/2;
          else if(xx==0 && yy==0)phi= 0;
          phi=3*PI/2-phi; /* not in normal artan routine */
          if(phi<0)phi=pi2+phi;
          if(phi>pi2)phi=phi-pi2;
          return(phi);
}
/* here to draw logo */

logo(page,maxpage,title,issue)
          int page,maxpage;
          char title[],issue[];
{
          int aa,bb,cc,dd,xedge,yedge,labelx,notex;

          xedge= -HWIDTH;/* left edge of drawing */
          labelx= HWIDTH-200;/* inside edge of label */
          yedge= -HDEPTH;/* bottom line of frame */
          notex= -HWIDTH+20; /* start of notes */
```

```
        aa= yedge;
        bb= yedge+50;
        cc= HWIDTH-200;
        dd= HWIDTH;

        fprintf(file_ptr,"SP1;PU%d,%d;EA%d,%d;\n",cc,bb,dd,aa);
        fprintf(file_ptr,"DI1,0;");
        aa=yedge +40;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LBLoughborough
Anthropometric Shadow Scanner\3 \n",labelx,aa);
        aa=yedge +25;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LB%s\3
\n",labelx,aa,title);
        aa=yedge +10;
        fprintf(file_ptr,"DT\3;LO11;PU%d,%d;LBPage %d of %d. Issue
%s \3 \n",labelx,aa,page,maxpage,issue);
}
/* here to ask a question & get an answer */

quest(infile,outfile)
        char infile[],outfile[];
{
        Device val;
        long dev;
        int flg,i;

        winset(textwin);
        c3s(whitevec);
        clear();
        c3s(blackvec);
        cmov2(1.0,7.0);
        charstr(infile);
        cmov2(1.0,4.0);
        flg=1; i=0;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
```

```
        case KEYBD:
        if(val=='\b'){
        i--;
        outfile[i]= '\0';
        c3s(whitevec);
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        outfile[i++]=val;
        }
        if(val=='\n')flg=0;
        winset(textwin);
        c3s(whitevec);
        clear();
        c3s(blackvec);
        cmov2(1.0,7.0);
        charstr(infile);
        cmov2(1.0,4.0);
        charstr(outfile);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
  return(i);
}
```

```
/* masterfix.c */
#include "device.h"
#include "lassdefs"
#include <gl.h>
#include <math.h>
#define MAXINFILE 20
#define MAXSHAPEFILE 100

        float interpsl[4*MAXSLICES][NPTS+1][2];
        int mcount,spangcnt,dcount;
        short blackvec[3] = { 0, 0, 0};
        short redvec[3] = { 255, 0, 0};
        short greenvec[3] = { 73, 206, 130};
        short bluevec[3] = { 0, 0, 255};
        short whitevec[3] = {255,255,255};
        char *name;
        long textwin;


main()
{
        inittextwin();
        qdevice(ESCKEY);
        qdevice(KEYBD);
        qdevice(RETKEY);
        masterlod();
        fixfile();
        mastersav();
}
/* here to load master shape data */


masterlod()
{
 FILE *fopen(),*file_ptr;
 char *string1,string2[50],*string3,string4[50],buffer[20];
  int count,i,j,k,m;
 float xx,yy,temp;
```

```
  string1 = "Enter input file name ";
 string3 = "r";

       for(i=0;i<50;i++){
       string2[i]='\0';
       string4[i]='\0';
       }
       i=0;
       while(string1[i] !='\0'){
       string2[i]=string1[i];
       i++;
       }

 name = &string4[0];
       winset(textwin);
       winpop();
       c3s(whitevec);
       clear();
       c3s(blackvec);
 putline(string2,1.0,5.0);
 count = getline(string2,i,50);
 sscanf(&string2[i],"%s",name);
 winpush();
file_ptr=fopen(name,string3);
      fscanf(file_ptr,"%d",&mcount);
      fscanf(file_ptr,"%d",&spangcnt);
        for(i=0;i<mcount;i++){
          for(j=0;j<=NPTS;j++){
           for(k=0;k<2;k++){
      fscanf(file_ptr,"%f",&xx);
      interpsl[i][j][k]=xx;
        } /* end of for k */
       } /* end of for j */
      } /* end of for i */
      fclose(file_ptr);

/* here to save  master shape data  */
```

```
mastersav()
 {
 FILE *fopen(),*file_ptr;
 char *string1,string2[50],*string3,string4[50];
 int count,i,j;

 string1 = "Enter output file name ";
 string3 = "w";

      for(i=0;i<50;i++){
      string2[i]='\0';
      string4[i]='\0';
      }
      i=0;
      while(string1[i] !='\0'){
      string2[i]=string1[i];
      i++;
      }

name = &string4[0];
      winset(textwin);
      winpop();
      c3s(whitevec);
      clear();
      c3s(blackvec);
putline(string2,1.0,5.0);
count = getline(string2,i,50);
sscanf(&string2[i],"%s",name);
winpush();

file_ptr=fopen(name,string3);
     fprintf(file_ptr,"%d",mcount);
     fprintf(file_ptr," %d\n",spangcnt);
       for(i=0;i<mcount;i++){
         for(j=0;j<(NPTS+1);j++){
        fprintf(file_ptr,"%8.1f%8.1f%",interpsl[i][j][0],interpsl[i][j][1]);
       }
     fprintf(file_ptr,"\n");
```

```
        }

        fclose(file_ptr);
}

fixfile()
{
        int i,j;
        float yy;


          for(i=0;i<mcount;i++){
          yy= (interpsl[i][0][1] + interpsl[i][NPTS-1][1])/2;
            for(j=0;j<NPTS;j++){
          interpsl[i][j][1]=interpsl[i][j][1] - yy;
          }
         }
}

inittextwin()
{       .
                        prefposition(0,      XMAXSCREEN/3,
YMAXSCREEN/10,2*YMAXSCREEN/10);
   textwin = winopen("text");
        RGBmode();
        shademodel(FLAT);
        ortho2(0,15,0,10);
        foreground();
        gconfig();
        winpush();
}
/* here to get line into s, return length */

int getline(s,cnt,lim)
        char s[];
        int cnt,lim;
{
```

```
        Device val;
        long dev;
        int c,flg,i;

        flg=1; i=cnt;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
        case KEYBD:
        if(val=='\b'){
        i--;
        s[i]= '\0';
        c3s(whitevec);
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        s[i++]=val;
        }
        if(val=='\n')flg=0;
        putline(s,1.0,5.0);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
} /* end of while flg */
```

```
    return(i);
}
/* here to write a line to a window */

putline(s,x,y)
        char s[];
        Coord x,y;
{
        cmov2(x,y);
        charstr(s);
}
```

/* newangle.c */

```
#include "device.h"
#include "lassdefs"
#include <gl.h>
#include <math.h>
#define UMAX 10000 /* maximum u value allowed */
#define DEF 14 /* binary digit resolution */
#define SLICEMAX 200 /* maximum slices in file */


Matrix cardinal = {
        {-0.5, 1.5,-1.5, 0.5},
        { 1.0,-2.5, 2.0,-0.5},
        {-0.5, 0.0, 0.5, 0.0},
        { 0.0, 1.0, 0.0, 0.0},
        };

float spangles[NPTS] =
        { 0.0,  12.0, 24.0, 36.0, 48.0,  60.0,  72.0,  84.0,  96.0, 108.0, 120.0, 132.0,
144.0, 156.0, 168.0, 180.0 };

        float interpsl[SLICEMAX][NPTS+1][2];
        float outfile[SLICEMAX][NPTS+1][2];
        float apoly[2];
        int mcount,spangcnt,dcount;
```

```
        short blackvec[3] = { 0, 0, 0};
        short redvec[3] = { 255, 0, 0};
        short greenvec[3] = { 73, 206, 130};
        short bluevec[3] = { 0, 0, 255};
        short whitevec[3] = {255,255,255};
        char *name;
        long textwin;


main()
{
        inittextwin();
        qdevice(ESCKEY);
        qdevice(KEYBD);
        qdevice(RETKEY);
        masterlod();
        fixfile();
        mastersav();
}
/* here to load master shape data */

masterlod()
{
 FILE *fopen(),*file_ptr;
 char *string1,string2[50],*string3,string4[50];
 int count,i,j,k;
 float xx;

  string1 = "Enter input file name ";
 string3 = "r";

      for(i=0;i<50;i++){
      string2[i]='\0';
      string4[i]='\0';
      }
      i=0;
      while(string1[i] !='\0'){
      string2[i]=string1[i];
      i++;
```

```
        }

   name = &string4[0];
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
   putline(string2,1.0,5.0);
   count = getline(string2,i,50);
   sscanf(&string2[i],"%s",name);
   winpush();
   file_ptr=fopen(name,string3);
        fscanf(file_ptr,"%d",&mcount);
        fscanf(file_ptr,"%d",&spangcnt);
          for(i=0;i<mcount;i++){
            for(j=0;j<=NPTS;j++){
             for(k=0;k<2;k++){
        fscanf(file_ptr,"%f",&xx);
        interpsl[i][j][k]=xx;
          } /* end of for k */
         } /* end of for j */
        } /* end of for i */
        fclose(file_ptr);
}
/* here to save  master shape data  */


mastersav()
 {
 FILE *fopen(),*file_ptr;
 char *string1,string2[50],*string3,string4[50];
 int count,i,j;

 string1 = "Enter output file name ";
 string3 = "w";

      for(i=0;i<50;i++){
      string2[i]='\0';
```

```
        string4[i]='\0';
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }

  name = &string4[0];
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
   putline(string2,1.0,5.0);
   count = getline(string2,i,50);
   sscanf(&string2[i],"%s",name);
   winpush();

   file_ptr=fopen(name,string3);
        fprintf(file_ptr,"%d",mcount);
        fprintf(file_ptr," %d\n",spangcnt);
          for(i=0;i<mcount;i++){
            for(j=0;j<=NPTS;j++){
           fprintf(file_ptr,"%8.1f%8.1f%",outfile[i][j][0],outfile[i][j][1]);
          }
        fprintf(file_ptr,"\n");
          }

        fclose(file_ptr);
}

fixfile()
{
        int i,j;
        float angle;
```

```
            for(i=0;i<mcount;i++){
              for(j=0;j<NPTS;j++){
               angle=spangles[j];
               xyangle(angle,i);
              outfile[i][j][0]=apoly[0];
              outfile[i][j][1]=apoly[1];
             } /* end of for j */
            outfile[i][NPTS][0]=interpsl[i][NPTS][0];
            outfile[i][NPTS][1]=interpsl[i][NPTS][1];
           } /* end of for i */
}
/* here to find x,y values for a particular angle from cubic curve */

xyangle(angle,slice)
        float angle;
        int slice;
{
        int i,j,k;
        float aa,bb,cc,xx,point;
        double artan();


        angle=(270-angle)*RAD;
/*
printf("angle=%f slice=%d\n",angle*180/PI,slice);
*/
        aa=0.0;
        bb=UMAX;
        cc=(bb-aa)/2;
           for(i=0;i<=DEF;i++){
           cardret((int)cc,UMAX,slice);
          xx = artan(apoly[1],apoly[0]);
          if(angle<xx)aa=cc;
          else bb=cc;
          cc=aa+(bb-aa)/2;
          }
}
/* here to return a point from a Cardinal curve */
```

```
cardret(point,maxpoints,sliceno)
        int point,maxpoints,sliceno;
{
        int s,u;
        float frangle[4],temp[4],npoints,fracpnt,interval;


        apoly[0]=0;
        apoly[1]=0;


        interval=NPTS-1;
        npoints = (float) maxpoints/interval;
        s= point/npoints;
        fracpnt = (point-s*npoints)/npoints;
        s=s-1;

        frangle[3]=1.0;
        frangle[2]=fracpnt;
        frangle[1]=frangle[2]*frangle[2];
        frangle[0]=frangle[1]*frangle[2];
        colmat(frangle,cardinal,temp);
          for(u=0;u<4;u++){
           if(s<0 && u==0){
           apoly[0]=apoly[0]-temp[0]*interpsl[sliceno][1][0];
           apoly[1]=apoly[1]+temp[0]*interpsl[sliceno][1][1];
           }
            else if ((u+s)>=0 && (u+s)<NPTS ){
           apoly[0]=apoly[0]+temp[u]*interpsl[sliceno][u+s][0];
           apoly[1]=apoly[1]+temp[u]*interpsl[sliceno][u+s][1];
           }
            else if((u+s)==NPTS){
           apoly[0]=apoly[0]-temp[3]*interpsl[sliceno][NPTS-2][0];
           apoly[1]=apoly[1]+temp[3]*interpsl[sliceno][NPTS-2][1];
           }
        } /* end of for u */
}
```

```
/* here to resolve atan calculations */

double artan(yy,xx)
        float yy,xx;

{
        float phi,pi2;

        pi2=2*PI;
        if(xx>0)phi=fatan(yy/xx);
        else if(xx<0)phi=PI+fatan(yy/xx);
        else if(xx==0 && yy>0)phi= PI/2;
        else if(xx==0 && yy<0)phi= 3*PI/2;
        else if(xx==0 && yy==0)phi= 0;
        if(phi<0)phi=pi2+phi;
        if(phi>pi2)phi=phi-pi2;
        return(float)(phi);
}
/* matrix multiply of 4 X 4 array , A * B result in R*/

matmult(mata,matb,matr)
        float mata[4][4],matb[4][4],matr[4][4];
{
        int i,j,k;
        float temp[4][4];

        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
         temp[i][j]=0;
           for(k=0;k<4;k++){
          temp[i][j]=temp[i][j]+mata[i][k]*matb[k][j];
          }
         }
        }
        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
         matr[i][j]=temp[i][j];
         }
```

```
        }
}
/*matrix multiply 4 X 4 array times 4 X 1 array, A * B result in R */

matcol(mata,matb,matr)
        float mata[4][4],matb[4],matr[4];
{
        int i,j;
        float temp[4];

        for(i=0;i<4;i++){
        temp[i]=0;
          for(j=0;j<4;j++){
         temp[i]=temp[i]+mata[i][j]*matb[j];
         }
        }
        for(i=0;i<4;i++)matr[i]=temp[i];
}
/*matrix multiply 4 X 1 array times 4 X 4 array, A * B result in R */

colmat(mata,matb,matr)
        float mata[4],matb[4][4],matr[4];
{
        int i,j;
        float temp[4];

        for(i=0;i<4;i++){
        temp[i]=0;
          for(j=0;j<4;j++){
         temp[i]=temp[i]+mata[j]*matb[j][i];
         }
        }
        for(i=0;i<4;i++)matr[i]=temp[i];
}

inittextwin()
{
```

```
   prefposition(0,
XMAXSCREEN/3,YMAXSCREEN/10,2*YMAXSCREEN/10);
   textwin = winopen("text");
        RGBmode();
        shademodel(FLAT);
        ortho2(0,15,0,10);
        foreground();
        gconfig();
        winpush();
}
/* here to get line into s, return length */

int getline(s,cnt,lim)
        char s[];
        int cnt;
{
        Device val;
        long dev;
        int flg,i;

        flg=1; i=cnt;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
        case KEYBD:
        if(val=='\b'){
        i--;
        s[i]= '\0';
        c3s(whitevec);
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        s[i++]=val;
        }
        if(val=='\n')flg=0;
        putline(s,1.0,5.0);
```

```
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
  return(i);
}
/* here to write a line to a window */

putline(s,x,y)
        char s[];
        Coord x,y;
{
        cmov2(x,y);
        charstr(s);
}
```

/* splot.c */

```
/* Creates a plotfile for H.P.Plotter from 16 spline points */
#include <stdio.h>
#include "device.h"
#include <gl.h>
#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>
#define NPTS 16 /* number of points in curve fit polygon */
```

```
#define PI 3.141592654

Matrix cardinal = {
        {-0.5, 1.5,-1.5, 0.5},
        { 1.0,-2.5, 2.0,-0.5},
        {-0.5, 0.0, 0.5, 0.0},
        { 0.0, 1.0, 0.0, 0.0},
        };


        short blackvec[3] = { 0, 0, 0};
        short redvec[3] = { 255, 0, 0};
        short greenvec[3] = { 73, 206, 130};
        short bluevec[3] = { 0, 0, 255};
        short whitevec[3] = {255,255,255};
        long textwin;
        float MINIHEAD[20];
        float slices[50][NPTS+1][3];
        float elipse[8][2];
        float apoly[3];
        int dcount,bytecount,mflg;
        int maxhite, minhite;
        char *name;
        FILE *file_ptr;


main()
        {
        inittextwin();
  qdevice(ESCKEY);
  qdevice(KEYBD);
  qdevice(RETKEY);
        slicelod();
        sliceplot();
        }


inittextwin()
{
```

```
   prefposition(2*
XMAXSCREEN/5,4*XMAXSCREEN/5,4*YMAXSCREEN/10,5*YMAXS
CREEN/10);
   textwin = winopen("Slice plot.");
       RGBmode();
       shademodel(FLAT);
       ortho2(0,15,0,10);
       foreground();
       gconfig();
}
/* here to load  slice data */

slicelod()
{
 FILE *fopen();
  char *string1,string2[50],*string3,string4[50];
  int bytecount,count,i;

  string1 = "Enter source file name ";
  string3 = "r";

       for(i=0;i<50;i++){
       string2[i]='\0';
       string4[i]='\0';
       }
       i=0;
       while(string1[i] !='\0'){
       string2[i]=string1[i];
       i++;
       }

 name = &string4[0];
       winset(textwin);
       c3s(whitevec);
       clear();
       c3s(blackvec);

 putline(string2,1.0,5.0);
```

```
  count = getline(string2,i,50);
  sscanf(&string2[i],"%s",name);
 file_ptr=fopen(name,string3);
 fread(MINIHEAD,sizeof(*MINIHEAD),20,file_ptr);
dcount =(int) MINIHEAD[1];
mflg =(int) MINIHEAD[2];
bytecount = fread(slices,sizeof(*slices),dcount,file_ptr);
      fclose(file_ptr);
}
/* here to create a plot file */

sliceplot()
{
 FILE *fopen(),*file_ptr;
 char *string1,string2[50],*string3,string4[50];
 char *string5,string6[50],*string7,string8[50];
  int count,i,j,height,dens;
 float theta,sinsq,cossq,aa,bb,radius,pi2;
 float poly[2];

 string1 = "Enter plot file name ";
 string3 = "w";
 string5 = "Enter title.";
 string7 = "Enter issue No.";
      dens=75;
      for(i=0;i<50;i++){
      string2[i]='\0';
      string4[i]='\0';
      string6[i]='\0';
      string8[i]='\0';
      }
      i=0;
      while(string1[i] !='\0'){
      string2[i]=string1[i];
      i++;
      }
      winset(textwin);
      c3s(whitevec);
```

```
        clear();
        c3s(blackvec);

        putline(string2,1.0,5.0);
        count = getline(string2,i,50);
        sscanf(&string2[i],"%s",string4);
        file_ptr=fopen(string4,string3);
        count=quest(string5,string6);
        count=quest(string7,string8);
       fprintf(file_ptr,"IN;PA;\n");
      fprintf(file_ptr,"IP-10640,-6680,10640,6680;\n");
      fprintf(file_ptr,"PS21280;\n");
       fprintf(file_ptr,"SC-281,40,-281,40,2;\n");
         for(j=0;j<dcount;j++){
         logo(j+1,dcount,string6,string8);
       fprintf(file_ptr,"SP2;\n");
        height = (int)slices[j][NPTS][0];
         fprintf(file_ptr,"DT\3;LO14;PU0,0;LBFile %s  Height = %d\3
\n",string4,height);
        refax(file_ptr);
        cardret(0,dens,j);
       poly[0]=apoly[0];
       poly[1]=apoly[1];
      fprintf(file_ptr,"PU%6.2f,%6.2f,",apoly[0],apoly[1]);
        fprintf(file_ptr,"\nPD");
         for(i=1;i<dens;i++){
         cardret(i,dens,j);
       if(i%5==0)fprintf(file_ptr,"%6.2f,%6.2f;\nPD",apoly[0],apoly[1]);
       else fprintf(file_ptr,"%6.2f,%6.2f,",apoly[0],apoly[1]);
        }
/* here to plot mirror image */
     fprintf(file_ptr,"\nSP2;");
     fprintf(file_ptr,"\nPD;");
       if(mflg==1){
        for(i=dens;i>=0;i--){
        cardret(i,dens,j);
       if(i%5==0)fprintf(file_ptr,"%6.2f,%6.2f;\nPD",-apoly[0],apoly[1]);
       else fprintf(file_ptr,"%6.2f,%6.2f,",-apoly[0],apoly[1]);
```

```
            }
            }
          fprintf(file_ptr,"%6.2f,%6.2f;\n",poly[0],poly[1]);
            fprintf(file_ptr,"PU;PG;\n");
          }
         fprintf(file_ptr,"PU;");
         fclose(file_ptr);
 }
 /* here to draw logo */

 logo(page,maxpage,title,issue)
        int page,maxpage;
        char title[],issue[];
 {
        fprintf(file_ptr,"SP1;PU-280,-182;EA280,182;\n");
        fprintf(file_ptr,"SP1;PU-280,-182;EA-81,-130;\n");
        fprintf(file_ptr,"DT\3;LO11;PU-275,-145;LBLoughborough
 Anthropometric Shadow Scanner\3 \n");
        fprintf(file_ptr,"DT\3;LO11;PU-275,-160;LB%s\3 \n",title);
        fprintf(file_ptr,"DT\3;LO11;PU-275,-175;LBPage %d of %d. Issue
 %s \3 \n",page,maxpage,issue);
 }
 /* here to draw ref axes on the plotter */

 refax(file_ptr)
        FILE *file_ptr;
 {
        fprintf(file_ptr,"PU-180,0;\n");
        fprintf(file_ptr,"PD-180,0,180,0;\n");
        fprintf(file_ptr,"PU0,-50;\n");
        fprintf(file_ptr,"PD0,-50,0,50;\n");
 }

 /* here to get line into s, return length */

 int getline(s,cnt,lim)
       char s[];
```

```
        int cnt,lim;
{
        Device val;
        long dev;
        int c,flg,i;

        flg=1; i=cnt;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
        case KEYBD:
        if(val=='\b'){
        i--;
        s[i]= '\0';
        c3s(whitevec);
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        s[i++]=val;
        }
        if(val=='\n')flg=0;
        putline(s,1.0,5.0);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
```

```
    } /* end of while qtest */
   } /* end of while flg */
   return(i);
}
/* here to write a line to a window */

putline(s,x,y)
        char s[];
        Coord x,y;
{
        cmov2(x,y);
        charstr(s);
}
/* here to ask a question & get an answer */

quest(infile,outfile)
        char infile[],outfile[];
{
        Device val;
        long dev;
        int c,flg,i;

        winset(textwin);
        c3s(whitevec);
        clear();
        c3s(blackvec);
        cmov2(1.0,7.0);
        charstr(infile);
        cmov2(1.0,4.0);
        flg=1; i=0;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
        case KEYBD:
        if(val=='\b'){
        i--;
        outfile[i]= '\0';
```

```
        c3s(whitevec);
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        outfile[i++]=val;
        }
        if(val=='\n')flg=0;
        winset(textwin);
        c3s(whitevec);
        clear();
        c3s(blackvec);
        cmov2(1.0,7.0);
        charstr(infile);
        cmov2(1.0,4.0);
        charstr(outfile);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
   return(i);
}
/* here to return a point from a Cardinal curve */

cardret(point,maxpoints,sliceno)
        int point,maxpoints,sliceno;
```

```
{
    int s,u;
    float frangle[4],temp[4],npoints,fracpnt,interval;


    apoly[0]=0;
    apoly[1]=0;
    apoly[2]=0;

    interval=NPTS-1;

    npoints = (float) maxpoints/interval;

    s= interval*point/maxpoints;
    fracpnt = (point-s*npoints)/npoints;

    s=s-1;

    frangle[3]=1.0;
    frangle[2]=fracpnt;
    frangle[1]=frangle[2]*frangle[2];
    frangle[0]=frangle[1]*frangle[2];
    colmat(frangle,cardinal,temp);
      for(u=0;u<4;u++){
       if(s<0 && u==0){
       apoly[0]=apoly[0]-temp[0]*slices[sliceno][1][0];
       apoly[1]=apoly[1]+temp[0]*slices[sliceno][1][1];
       }
        else if ((u+s)>=0 && (u+s)<NPTS ){
       apoly[0]=apoly[0]+temp[u]*slices[sliceno][u+s][0];
       apoly[1]=apoly[1]+temp[u]*slices[sliceno][u+s][1];
       }
        else if((u+s)==NPTS){
       apoly[0]=apoly[0]-temp[3]*slices[sliceno][NPTS-2][0];
       apoly[1]=apoly[1]+temp[3]*slices[sliceno][NPTS-2][1];
       }
    } /* end of for u */
```

```
}
/*matrix multiply 4 X 1 array times 4 X 4 array, A * B result in R */

colmat(mata,matb,matr)
        float mata[4],matb[4][4],matr[4];
{
        int i,j;
        float temp[4];

        for(i=0;i<4;i++){
        temp[i]=0;
          for(j=0;j<4;j++){
          temp[i]=temp[i]+mata[j]*matb[j][i];
          }
        }
        for(i=0;i<4;i++)matr[i]=temp[i];
}
/* here to print slice array */

pslice()
{
       int i,j;

       for(i=0;i<dcount;i++){
         for(j=0;j<11;j++){
          printf("%7.2f %7.2f  ",slices[i][j][0],slices[i][j][1]);
         if(j==3 || j==7 || j==10)printf("\n");
         }
       }
}
```

/* vslice.c */

```
/* Creates a plotfile to give vertical slices
              on H.P.Plotter from 16 spline points */
#include <stdio.h>
#include "device.h"
```

```
#include <gl.h>
#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>
#define NPTS 16 /* number of points in curve fit polygon */
#define NFRAME 420 /* no. of vertical intervals in 2.1 metres
2100/VSTEP */
#define PATCHMAX 1600 /* size of patches array */
#define PI 3.141592654


Coord geomxfr[4][4];
Coord geomyfr[4][4];
Coord geomzfr[4][4];
Coord geomxfl[4][4];
Coord geomyfl[4][4];
Coord geomzfl[4][4];
Coord tempxf[4][4];
Coord tempyf[4][4];
Coord tempzf[4][4];

float trans[4][4] = {
        { 1,  0,  0, 0 },
        { 0,  1,  0, 0 },
        { 0,  0,  1, 0 },
        { 0,  0,  0, 1 },
        };
Matrix cardinal = {
        {-0.5, 1.5,-1.5, 0.5},
        { 1.0,-2.5, 2.0,-0.5},
        {-0.5, 0.0, 0.5, 0.0},
        { 0.0, 1.0, 0.0, 0.0},
        };


        short blackvec[3] = { 0, 0, 0};
        short redvec[3] = { 255, 0, 0};
        short greenvec[3] = { 73, 206, 130};
```

```
        short bluevec[3] = { 0, 0, 255};
        short whitevec[3] = {255,255,255};
        long textwin;
        float  MINIHEAD[20];
        float patches[PATCHMAX][3][4][4];
        float DPOLY[NPTS+1][2];
        float slices[50][NPTS+1][2];
        float vslice[NFRAME][3];
        float apoly[3];
        float secangle;
        float set1,set2;
        int dcount,bytecount,mflg,vcnt,spangcnt;
        int maxhite, minhite;
        int patchcount;
        char *name;


main()
        {
        inittextwin();
  qdevice(ESCKEY);
  qdevice(KEYBD);
  qdevice(RETKEY);
        slicelod();
        anglein();
        bslcptch(dcount);
        getvslice(secangle);
/*
vslprnt();
*/
        sliceplot();
        }


inittextwin()
{
  prefposition(2*
XMAXSCREEN/5,4*XMAXSCREEN/5,4*YMAXSCREEN/10,5*YMAXS
CREEN/10);
  textwin = winopen("Slice plot.");
```

```
        RGBmode();
        shademodel(FLAT);
        ortho2(0,15,0,10);
        foreground();
        gconfig();
}
/* here to load slice data */

slicelod()
{
 FILE *fopen(),*file_ptr;
 char *string1,string2[50],*string3,string4[50],buffer[20];
  int count,i,j,k;
 float xx,yy;

  string1 = "Enter file name ";
 string3 = "r";

      for(i=0;i<50;i++){
      string2[i]='\0';
      string4[i]='\0';
      }
      i=0;
      while(string1[i] !='\0'){
      string2[i]=string1[i];
      i++;
      }

 name = &string4[0];
      winset(textwin);
      winpop();
      c3s(whitevec);
      clear();
      c3s(blackvec);
 putline(string2,1.0,5.0);
 count = getline(string2,i,50);
 sscanf(&string2[i],"%s",name);
 winpush();
```

```
  file_ptr=fopen(name,string3);
        fscanf(file_ptr,"%d",&dcount);
        fscanf(file_ptr,"%d",&spangcnt);
          for(i=0;i<dcount;i++){
            for(j=0;j<=NPTS;j++){
             for(k=0;k<2;k++){
        fscanf(file_ptr,"%f",&xx);
        slices[i][j][k]=xx;
          }
         }
        }
        fclose(file_ptr);
        set1=slices[0][NPTS][0];
        set2=slices[dcount-1][NPTS][0];
}

/* here to create a plot file */

sliceplot()
{
  FILE *fopen(),*file_ptr;
  char *string1,string2[50],*string3,string4[50];
   int count,i,j,height,vstep;
  float poly[2];


  string1 = "Enter plot file name ";
  string3 = "w";
        vstep=5;
        for(i=0;i<50;i++){
        string2[i]='\0';
        string4[i]='\0';
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }
```

```
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);

        putline(string2,1.0,5.0);
        count = getline(string2,i,50);
        sscanf(&string2[i],"%s",string4);
        winpush();
        file_ptr=fopen(string4,string3);
       fprintf(file_ptr,"IN;SP1;PA;\n");
      fprintf(file_ptr,"IP-15100,-10640,15100,10640;\n");
       fprintf(file_ptr,"PA-15100,-10640;EA15100,10640;\n");
       fprintf(file_ptr,"SC%f,40,-281,40,2;\n",set1-30.0);
/* here to plot axis */
       fprintf(file_ptr,"DI0,-1;DT#;LO14;PU%8.0f,0;LBFile %s  Height =
%8.0f#\n",set1,string4,set1);
       fprintf(file_ptr,"PA%8.2f,%8.2f;",set1,0.0);
       fprintf(file_ptr,"PD%8.2f,%8.2f;",set1,0.0);
       fprintf(file_ptr,"PD%8.2f,%8.2f;",set2,0.0);
       fprintf(file_ptr,"\nDI0,-1;DT#;LO14;PU%8.0f,0;LBFile %s  Height
= %8.0f#\n",set2,string4,set2);
       fprintf(file_ptr,"PU;");
/* here to plot front side */
       fprintf(file_ptr,"PU%8.2f,%8.2f;",set1,vslice[0][0]);
       for(i=0;i<vcnt;i++){
       fprintf(file_ptr,"PD%8.2f,%8.2f;",vslice[i][2],vslice[i][0]);
       if(i%5==0)fprintf(file_ptr,"\n");
       }
       fprintf(file_ptr,"\n");
/* here to plot reverse side */
       fprintf(file_ptr,"PU%8.2f,%8.2f;",set1,-vslice[0][1]);
       for(i=0;i<vcnt;i++){
       fprintf(file_ptr,"PD%8.2f,%8.2f;",vslice[i][2],-vslice[i][1]);
       if(i%5==0)fprintf(file_ptr,"\n");
       }
       fprintf(file_ptr,"PU;SP0;");
```

```
        fclose(file_ptr);
}
/* here to get line into s, return length */

int getline(s,cnt,lim)
        char s[];
        int cnt,lim;
{
        Device val;
        long dev;
        int c,flg,i;

        flg=1; i=cnt;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
        case KEYBD:
        if(val=='\b'){
        i--;
        s[i]= '\0';
        c3s(whitevec);
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        s[i++]=val;
        }
        if(val=='\n')flg=0;
        putline(s,1.0,5.0);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
```

```
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
  return(i);
}
/* here to write a line to a window */

putline(s,x,y)
        char s[];
        Coord x,y;
{
        cmov2(x,y);
        charstr(s);
}
/* here to return a point from a Cardinal curve */

cardret(point,maxpoints)
        int point,maxpoints;
{
        int s,u;
        float frangle[4],temp[4],npoints,fracpnt,interval;


        apoly[0]=0;
        apoly[1]=0;
        apoly[2]=0;

        interval=NPTS-1;
        npoints = (float) maxpoints/interval;
        s= (int)point/npoints;
        fracpnt = (point-s*npoints)/npoints;
        s=s-1;
```

```
        frangle[3]=1.0;
        frangle[2]=fracpnt;
        frangle[1]=frangle[2]*frangle[2];
        frangle[0]=frangle[1]*frangle[2];

        colmat(frangle,cardinal,temp);

          for(u=0;u<4;u++){
           if((u+s)<0){
          apoly[0]=apoly[0]-temp[0]*DPOLY[1][0];
          apoly[1]=apoly[1]+temp[0]*DPOLY[1][1];
          }
           else if ((u+s)>=0 && (u+s)<NPTS ){
          apoly[0]=apoly[0]+temp[u]*DPOLY[u+s][0];
          apoly[1]=apoly[1]+temp[u]*DPOLY[u+s][1];
          }
           else if((u+s)>=NPTS){
          apoly[0]=apoly[0]-temp[3]*DPOLY[NPTS-2][0];
          apoly[1]=apoly[1]+temp[3]*DPOLY[NPTS-2][1];
          }
    . } /* end of for u */
       apoly[2]=DPOLY[NPTS][0];
}

/*matrix multiply 4 X 1 array times 4 X 4 array, A * B result in R */

colmat(mata,matb,matr)
       float mata[4],matb[4][4],matr[4];
{
       int i,j;
       float temp[4];

       for(i=0;i<4;i++){
       temp[i]=0;
         for(j=0;j<4;j++){
        temp[i]=temp[i]+mata[j]*matb[j][i];
        }
       }
```

```
        for(i=0;i<4;i++)matr[i]=temp[i];
}
/* here to print slice array */

pslice()
{
        int i,j;

        for(i=0;i<dcount;i++){
          for(j=0;j<11;j++){
           printf("%7.2f %7.2f  ",slices[i][j][0],slices[i][j][1]);
          if(j==3 || j==7 || j==10)printf("\n");
          }
        }
}
/* here to create a vertical slice file from patch data */

getvslice(angle)
        float angle;
{
        int i,j,vstep,resltn,step,vcount;
        float xx,yy,radius;
        float phi,pi2;

        vstep=5;


        pi2=2*PI;
        resltn=750;
        vcnt=0;
        vcount=0;
        if(angle>(PI/2))angle=angle-(PI/2);

        for(i=(int)set1+1;i<(int)set2;i=i+vstep){
        vcount=i-set1-1;
        getpoly((float)i);
        step=0;
          for(j=0;j<resltn;j++){
```

```
          cardret(j,resltn);
          xx=apoly[0];
          yy=apoly[1];
          if(xx>0)phi=fatan(yy/xx);
          else if(xx<0)phi=PI+fatan(yy/xx);
          else if(xx==0 && yy>0)phi= PI/2;
          else if(xx==0 && yy<0)phi= 3*PI/2;
          else if(xx==0 && yy==0)phi=0;
          phi=3*PI/2-phi;
          if(phi<0)phi=pi2+phi;
          if(phi>pi2)phi=phi-pi2;
           if(phi>=angle-0.005 && phi<angle+0.005 && step<2){
          radius=fhypot(xx,yy);
          if(step==0 ){
          vslice[vcnt][0]=radius;
          step++;
          }
          else {
          vslice[vcnt][1]=radius;
          vslice[vcnt][2]=apoly[2];
           vcnt++;
          step++;
          } /* end of else */
          angle=PI-angle;
          } /* end of if phi */
        } /* end of for j */

      } /* end of for i */
}
/* here to get slice polygon from patch data */

getpoly(height)
      float height;
{
      int i,j,k,count;
      float htmin,htmax,w,ww,www,maxmin;

      count=0;
```

```
        for(i=0;i<patchcount;i++){
        htmin = patches[i][1][1][1];
        htmax = patches[i][1][2][1];

        if(height>=htmin && height<=htmax)break;
        } /* end of for i */
        count=i;
          maxmin=htmax-htmin;
         if(maxmin!=0) w =(height-htmin)/maxmin;
         else w=0;
         ww = w*w;
        www = ww*w;

        for(i=1;i<NPTS;i++){
          for(j=0;j<4;j++){
           for(k=0;k<4;k++){
          geomxfr[k][j] = patches[count][0][k][j];
          geomyfr[k][j] = patches[count][1][k][j];
          geomzfr[k][j] = -patches[count][2][k][j];
          }
         }
        matmult(cardinal,geomxfr,tempxf);
        matmult(cardinal,geomyfr,tempyf);
        matmult(cardinal,geomzfr,tempzf);

        matmult(tempxf,trans,tempxf);
        matmult(tempyf,trans,tempyf);
        matmult(tempzf,trans,tempzf);

      DPOLY[i][0]=www*tempxf[0][1]+ww*tempxf[1][1]+w*tempxf[2][1]+
tempxf[3][1];
      DPOLY[i][1]=www*tempzf[0][1]+ww*tempzf[1][1]+w*tempzf[2][1]+
tempzf[3][1];

      if(i==1){
      DPOLY[0][0]=www*tempxf[0][2]+ww*tempxf[1][2]+w*tempxf[2][2]
+tempxf[3][2];
```

```
        DPOLY[0][1]=www*tempzf[0][2]+ww*tempzf[1][2]+w*tempzf[2][2]
 +tempzf[3][2];
        }
        count++;
        } /* end of for i */
        DPOLY[NPTS][0]=www*tempyf[0][1]+ww*tempyf[1][1]+w*tempyf[
2][1]+tempyf[3][1];
}
/* here to input angle */

anglein()
{
        char *string1,string2[50],string3[50];
        int i;

        string1 = "Enter angle of section.";
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
        for(i=0;i<50;i++){
        string2[i]='\0';
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }
  putline(string2,1.0,5.0);
  getline(string2,i,50);
  sscanf(&string2[i],"%s",string3);
  secangle=atof(string3);
  secangle=secangle*PI/180;
  winpush();
}
/* here to print vslice */
```

```
vslprnt()
{
        int i;

        for(i=0;i<vcnt;i++)printf("%f          %f
%f\n",vslice[i][0],vslice[i][1],vslice[i][2]);
}
/* matrix multiply of 4 X 4 array , A * B result in R*/

matmult(mata,matb,matr)
        float mata[4][4],matb[4][4],matr[4][4];
{
        int i,j,k;
        float temp[4][4];

        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
         temp[i][j]=0;
           for(k=0;k<4;k++){
          temp[i][j]=temp[i][j]+mata[i][k]*matb[k][j];
           }
         }
        }
        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
         matr[i][j]=temp[i][j];
         }
        }
}
/* here to make large patches (total horiz. body section ) from 'n'
bspline slices */

bslcptch(count)
        int count;
{
        int i,j,k,l,m;
```

```
        for(j=0;j<(count-3);j=j+1){
          for(k= -1;k<NPTS-2;k++){
           for(i=0;i<4;i++){
           getslice(i+j);
           for(l=0;l<4;l++){
           m=k+l;
           if(m>=NPTS){
                tempxf[i][3-l]= -DPOLY[m-2][0];
                tempzf[i][3-l]= -DPOLY[m-2][1];
                }
           else if(m<0){
                tempxf[i][3-l]= -DPOLY[-m][0];
                tempzf[i][3-l]= -DPOLY[-m][1];
                }
           else {
                tempxf[i][3-l]= DPOLY[m][0];
                tempzf[i][3-l]= -DPOLY[m][1];
                }
          tempyf[i][3-l]=DPOLY[NPTS][0];
          } /* end of for l */
        } /* end of for i */
       pstore();
       } /* end of for k */
       } /* end of for j */
/* now to make patch of mirror image */
 if(mflg==1){
       for(j=0;j<(count-3);j=j+1){
         for(k= -1;k<(NPTS-2);k++){
          for(i=0;i<4;i++){
          getslice(i+j);
          for(l=0;l<4;l++){
          m=k+l;
          if(m>=NPTS){
               tempxf[i][l]= DPOLY[m-2][0];
               tempzf[i][l]= -DPOLY[m-2][1];
               }
          else if(m<0){
               tempxf[i][l]= DPOLY[-m][0];
```

```
            tempzf[i][l]= -DPOLY[-m][1];
            }
          else {
            tempxf[i][l]= -DPOLY[m][0];
            tempzf[i][l]= -DPOLY[m][1];
            }
          tempyf[i][l]=DPOLY[NPTS][0];
          }
        }
      pstore();
      }
      }
 }
}
/* here to retreive horizontal patch slices */

getslice(count)
      int count;
{
      int i;

      for(i=0;i<NPTS;i++){
        DPOLY[i][0]= slices[count][i][0];
        DPOLY[i][1]= slices[count][i][1];
      }
      DPOLY[NPTS][0] = slices[count][NPTS][0];
}
/* here to store a patch in memory */

pstore()
{
      int i,j;

      for(i=0;i<4;i++){
        for(j=0;j<4;j++){
       patches[patchcount][0][i][j]=tempxf[i][j];
       patches[patchcount][1][i][j]=tempyf[i][j];
       patches[patchcount][2][i][j]=tempzf[i][j];
```

```
        }
      }
      patchcount++;
}
```

```
/* vslice2.c */
/* Creates a plotfile to give vertical slices parallel to z axis
            on H.P.Plotter from 16 spline points */
#include <stdio.h>
#include "device.h"
#include <gl.h>
#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>
#define NPTS 16 /* number of points in curve fit polygon */
#define NFRAME 420 /* no. of vertical intervals in 2.1 metres
2100/VSTEP */
#define PATCHMAX 1600 /* size of patches array */
#define PI 3.141592654

Coord geomxfr[4][4];
Coord geomyfr[4][4];
Coord geomzfr[4][4];
Coord geomxfl[4][4];
Coord geomyfl[4][4];
Coord geomzfl[4][4];
Coord tempxf[4][4];
Coord tempyf[4][4];
Coord tempzf[4][4];

float trans[4][4] = {
      { 1, 0, 0, 0 },
      { 0, 1, 0, 0 },
      { 0, 0, 1, 0 },
      { 0, 0, 0, 1 },
      };
```

```
Matrix cardinal = {
        {-0.5, 1.5,-1.5, 0.5},
        { 1.0,-2.5, 2.0,-0.5},
        {-0.5, 0.0, 0.5, 0.0},
        { 0.0, 1.0, 0.0, 0.0},
        };


        short blackvec[3] = { 0, 0, 0};
        short redvec[3] = { 255, 0, 0};
        short greenvec[3] = { 73, 206, 130};
        short bluevec[3] = { 0, 0, 255};
        short whitevec[3] = {255,255,255};
        long textwin;
        float  MINIHEAD[20];
        float patches[PATCHMAX][3][4][4];
        float DPOLY[NPTS+1][2];
        float slices[50][NPTS+1][2];
        float vslice[NFRAME][3];
        float apoly[3];
        float setx;
        float set1,set2;
        int dcount,bytecount,mflg,vcnt,vcntr,spangcnt;
        int maxhite, minhite;
        int patchcount;
        char *name;
        FILE *file_ptr;


main()
      {
      inittextwin();
  qdevice(ESCKEY);
  qdevice(KEYBD);
  qdevice(RETKEY);
      slicelod();
      xin();
      bslcptch(dcount);
      getvslice(setx);
```

```
/*
vslprnt();
*/
        sliceplot();
        }

inittextwin()
{
   prefposition(2*
XMAXSCREEN/5,4*XMAXSCREEN/5,4*YMAXSCREEN/10,5*YMAXS
CREEN/10);
   textwin = winopen("Slice plot.");
        RGBmode();
        shademodel(FLAT);
        ortho2(0,15,0,10);
        foreground();
        gconfig();
}
/* here to load slice data */

slicelod()
{
  FILE *fopen(),*file_ptr;
  char *string1,string2[50],*string3,string4[50];
   int count,i,j,k;
  float xx;

   string1 = "Enter file name ";
  string3 = "r";

        for(i=0;i<50;i++){
        string2[i]='\0';
        string4[i]='\0';
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
```

```
        }

    name = &string4[0];
         winset(textwin);
         winpop();
         c3s(whitevec);
         clear();
         c3s(blackvec);
    putline(string2,1.0,5.0);
    count = getline(string2,i,50);
    sscanf(&string2[i],"%s",name);
    winpush();
   file_ptr=fopen(name,string3);
         fscanf(file_ptr,"%d",&dcount);
         fscanf(file_ptr,"%d",&spangcnt);
           for(i=0;i<dcount;i++){
             for(j=0;j<=NPTS;j++){
              for(k=0;k<2;k++){
         fscanf(file_ptr,"%f",&xx);
         slices[i][j][k]=xx;
          }
         }
        }
        fclose(file_ptr);
        set1=slices[0][NPTS][0];
        set2=slices[dcount-1][NPTS][0];
}

/* here to create a plot file */

sliceplot()
{
 FILE *fopen();
  char *string1,string2[50],*string3,string4[50];
  char *string5,string6[50],*string7,string8[50];
  int count,i;
  float xx,yy,gridinc;
```

```
gridinc=50;
string1 = "Enter plot file name ";
string3 = "w";
string5 = "Enter title.";
string7 = "Enter issue No.";

      for(i=0;i<50;i++){
      string2[i]='\0';
      string4[i]='\0';
      string6[i]='\0';
      string8[i]='\0';
      }
      i=0;
      while(string1[i] !='\0'){
      string2[i]=string1[i];
      i++;
      }
      winset(textwin);
      winpop();
      c3s(whitevec);
      clear();
      c3s(blackvec);

      putline(string2,1.0,5.0);
      count = getline(string2,i,50);
      sscanf(&string2[i],"%s",string4);
      count=quest(string5,string6);
      count=quest(string7,string8);
      winpush();
      file_ptr=fopen(string4,string3);
     fprintf(file_ptr,"IN;SP1;PA;\n");
    fprintf(file_ptr,"IP-15100,-10640,15100,10640;\n");
      fprintf(file_ptr,"PA-15100,-10640;EA15100,10640;\n");
      fprintf(file_ptr,"SC%f,40,-281,40,2;\n",set1-30.0);
      logo(1,1,string6,string8);
/* here to plot axis */
```

```
        fprintf(file_ptr,"DI0,-1;DT#;LO14;PU%8.0f,0;LBFile
%s#\n",gridinc/2+set1,string4);
        fprintf(file_ptr,"PA%8.2f,%8.2f;",set1,0.0);
        fprintf(file_ptr,"PD%8.2f,%8.2f;",set1,0.0);
        fprintf(file_ptr,"PD%8.2f,%8.2f;",set2,0.0);
/*
        fprintf(file_ptr,"\nDI0,-1;DT#;LO14;PU%8.0f,0;LBFile %s  Height
= %8.0f#\n",set2,string4,set2);
*/
        fprintf(file_ptr,"PU;");
/* here to plot front side */
        fprintf(file_ptr,"PU%8.2f,%8.2f;",set1,vslice[0][1]);
        for(i=0;i<vcnt;i++){
        fprintf(file_ptr,"PD%8.2f,%8.2f;",vslice[i][2],vslice[i][1]);
        if(i%5==0)fprintf(file_ptr,"\n");
        }
        fprintf(file_ptr,"\n");
/* here to plot reverse side */
        fprintf(file_ptr,"PU%8.2f,%8.2f;",set1,-vslice[0][1]);
        for(i=vcnt;i<vcntr;i++){
        fprintf(file_ptr,"PD%8.2f,%8.2f;",vslice[i][2],-vslice[i][1]);
        if(i%5==0)fprintf(file_ptr,"\n");
        }
/* here to draw grid */
       fprintf(file_ptr,"SP2;\n");
        xx=266;
/* horizontal lines */
        for(i=0;i<(int)750/gridinc;i++){
        yy=(float)i*gridinc+set1;
        fprintf(file_ptr,"PU%8.2f,%8.2f;",yy,-xx-15);
        fprintf(file_ptr,"PD%8.2f,%8.2f;",yy,xx-15);
        if(i%2==0)fprintf(file_ptr,"\n");
        fprintf(file_ptr,"DI0,-1;DT#;LO14;PU%8.0f,0;LB%8.0f#\n",yy,yy);
        }
        fprintf(file_ptr,"\n");
/* vertical lines */
        for(i= -(int)250/gridinc;i<=(int)250/gridinc;i++){
        fprintf(file_ptr,"PU%8.2f,%8.2f;",set1-30.0,(float)i*gridinc);
```

```
        fprintf(file_ptr,"PD%8.2f,%8.2f;",set1+725.5,(float)i*gridinc);
        if(i%2==0)fprintf(file_ptr,"\n");
        }
        fprintf(file_ptr,"PU;PG;SP0;\n");
        fclose(file_ptr);
}
/* here to draw logo */

logo(page,maxpage,title,issue)
        int page,maxpage;
        char title[],issue[];
{
        float aa,bb;

        aa=set1-30;
        bb=set1+20;
        fprintf(file_ptr,"SP1;PU%f,-81;EA%f,-281;\n",aa,bb);
        fprintf(file_ptr,"DI0,-1;");
        aa=set1+10;
        fprintf(file_ptr,"DT\3;LO11;PU%f,-86;LBLoughborough
Anthropometric Shadow Scanner\3 \n",aa);
        aa=set1-5;
        fprintf(file_ptr,"DT\3;LO11;PU%f,-86;LB%s\3 \n",aa,title);
        aa=set1-20;
        fprintf(file_ptr,"DT\3;LO11;PU%f,-86;LBPage %d of %d. Issue %s
\3 \n",aa,page,maxpage,issue);
}
/* here to get line into s, return length */

int getline(s,cnt,lim)
        char s[];
        int cnt,lim;
{
        Device val;
        long dev;
        int c,flg,i;

        flg=1; i=cnt;
```

```
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
        case KEYBD:
        if(val=='\b'){
        i--;
        s[i]= '\0';
        c3s(whitevec);
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        s[i++]=val;
        }
        if(val=='\n')flg=0;
        putline(s,1.0,5.0);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
  return(i);
}
/* here to write a line to a window */
```

```
putline(s,x,y)
        char s[];
        Coord x,y;
{
        cmov2(x,y);
        charstr(s);
}
/* here to return a point from a Cardinal curve */

cardret(point,maxpoints)
        int point,maxpoints;
{
        int s,u;
        float frangle[4],temp[4],npoints,fracpnt,interval;


        apoly[0]=0;
        apoly[1]=0;
        apoly[2]=0;

        interval=NPTS-1;
        npoints = (float) maxpoints/interval;
        s= (int)point/npoints;
        fracpnt = (point-s*npoints)/npoints;
        s=s-1;

        frangle[3]=1.0;
        frangle[2]=fracpnt;
        frangle[1]=frangle[2]*frangle[2];
        frangle[0]=frangle[1]*frangle[2];

        colmat(frangle,cardinal,temp);

         for(u=0;u<4;u++){
          if((u+s)<0){
         apoly[0]=apoly[0]-temp[0]*DPOLY[1][0];
         apoly[1]=apoly[1]+temp[0]*DPOLY[1][1];
         }
```

```
            else if ((u+s)>=0 && (u+s)<NPTS ){
            apoly[0]=apoly[0]+temp[u]*DPOLY[u+s][0];
            apoly[1]=apoly[1]+temp[u]*DPOLY[u+s][1];
            }
            else if((u+s)>=NPTS){
            apoly[0]=apoly[0]-temp[3]*DPOLY[NPTS-2][0];
            apoly[1]=apoly[1]+temp[3]*DPOLY[NPTS-2][1];
            }
        } /* end of for u */
        apoly[2]=DPOLY[NPTS][0];
}

/*matrix multiply 4 X 1 array times 4 X 4 array, A * B result in R */

colmat(mata,matb,matr)
        float mata[4],matb[4][4],matr[4];
{
        int i,j;
        float temp[4];

        for(i=0;i<4;i++){
        temp[i]=0;
          for(j=0;j<4;j++){
         temp[i]=temp[i]+mata[j]*matb[j][i];
          }
        }
        for(i=0;i<4;i++)matr[i]=temp[i];
}
/* here to print slice array */

pslice()
{
        int i,j;

        for(i=0;i<dcount;i++){
          for(j=0;j<11;j++){
           printf("%7.2f %7.2f  ",slices[i][j][0],slices[i][j][1]);
          if(j==3 || j==7 || j==10)printf("\n");
```

```
        }
      }
}
/* here to create a vertical slice file from patch data */

getvslice(xxx)
      float xxx;
{
      int i,j,vstep,resltn;
      float tol;

      vstep=5;
      tol=1.0;
      resltn=750;
      vcnt=0;

/* here to plot front */

      for(i=(int)set1+1;i<(int)set2;i=i+vstep){
      getpoly((float)i);
        for(j=0;j<resltn;j++){
        cardret(j,resltn);
            if(-apoly[0]>xxx-tol && -apoly[0]<xxx+tol){
            vslice[vcnt][1]=apoly[1];
            vslice[vcnt][2]=apoly[2];
            vcnt++;
            break;
            } /* end of if */
        } /* end of for j */
      } /* end of for i */
/* here to plot back */
      vcntr=vcnt;
      for(i=(int)set1+1;i<(int)set2;i=i+vstep){
      getpoly((float)i);
        for(j=resltn;j>=0;j--){
        cardret(j,resltn);
            if(-apoly[0]>xxx-tol && -apoly[0]<xxx+tol){
            vslice[vcntr][1]= -apoly[1];
```

```
            vslice[vcntr][2]=apoly[2];
            vcntr++;
            break;
            } /* end of if */
        } /* end of for j */
    } /* end of for i */
}
/* here to get slice polygon from patch data */

getpoly(height)
    float height;
{
    int i,j,k,count;
    float htmin,htmax,w,ww,www,maxmin;

    count=0;
    for(i=0;i<patchcount;i++){
    htmin = patches[i][1][1][1];
    htmax = patches[i][1][2][1];

    if(height>=htmin && height<=htmax)break;
    } /* end of for i */
    count=i;
      maxmin=htmax-htmin;
     if(maxmin!=0) w =(height-htmin)/maxmin;
     else w=0;
     ww = w*w;
    www = ww*w;

    for(i=1;i<NPTS;i++){
      for(j=0;j<4;j++){
       for(k=0;k<4;k++){
      geomxfr[k][j] =  patches[count][0][k][j];
      geomyfr[k][j] =  patches[count][1][k][j];
      geomzfr[k][j] = -patches[count][2][k][j];
      }
     }
    matmult(cardinal,geomxfr,tempxf);
```

```
        matmult(cardinal,geomyfr,tempyf);
        matmult(cardinal,geomzfr,tempzf);

        matmult(tempxf,trans,tempxf);
        matmult(tempyf,trans,tempyf);
        matmult(tempzf,trans,tempzf);

        DPOLY[i][0]=www*tempxf[0][1]+ww*tempxf[1][1]+w*tempxf[2][1]+
tempxf[3][1];
        DPOLY[i][1]=www*tempzf[0][1]+ww*tempzf[1][1]+w*tempzf[2][1]+
tempzf[3][1];

        if(i==1){
        DPOLY[0][0]=www*tempxf[0][2]+ww*tempxf[1][2]+w*tempxf[2][2]
+tempxf[3][2];
        DPOLY[0][1]=www*tempzf[0][2]+ww*tempzf[1][2]+w*tempzf[2][2]
+tempzf[3][2];
        }
        count++;
        } /* end of for i */
        DPOLY[NPTS][0]=www*tempyf[0][1]+ww*tempyf[1][1]+w*tempyf[
2][1]+tempyf[3][1];
}
/* here to input x value */

xin()
{
        char *string1,string2[50],string3[50];
        int i;

        string1 = "Enter x value of section.";
        winset(textwin);
        winpop();
        c3s(whitevec);
        clear();
        c3s(blackvec);
        for(i=0;i<50;i++){
        string2[i]='\0';
```

```
        }
        i=0;
        while(string1[i] !='\0'){
        string2[i]=string1[i];
        i++;
        }
  putline(string2,1.0,5.0);
  getline(string2,i,50);
  sscanf(&string2[i],"%s",string3);
  setx=atof(string3);
  winpush();
}
/* here to print vslice */

vslprnt()
{
        int i;

        for(i=0;i<vcnt;i++)printf("%f          %f
%f\n",vslice[i][0],vslice[i][1],vslice[i][2]);
}
/* matrix multiply of 4 X 4 array , A * B result in R*/

matmult(mata,matb,matr)
        float mata[4][4],matb[4][4],matr[4][4];
{
        int i,j,k;
        float temp[4][4];

        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
         temp[i][j]=0;
           for(k=0;k<4;k++){
          temp[i][j]=temp[i][j]+mata[i][k]*matb[k][j];
           }
         }
        }
        for(i=0;i<4;i++){
```

```
        for(j=0;j<4;j++){
        matr[i][j]=temp[i][j];
        }
      }
}
/* here to make large patches (total horiz. body section ) from 'n'
bspline slices */

bslcptch(count)
      int count;
{
      int i,j,k,l,m;


      for(j=0;j<(count-3);j=j+1){
        for(k= -1;k<NPTS-2;k++){
          for(i=0;i<4;i++){
          getslice(i+j);
          for(l=0;l<4;l++){
          m=k+l;
          if(m>=NPTS){
                tempxf[i][3-l]= -DPOLY[m-2][0];
                tempzf[i][3-l]= -DPOLY[m-2][1];
                }
          else if(m<0){
                tempxf[i][3-l]= -DPOLY[-m][0];
                tempzf[i][3-l]= -DPOLY[-m][1];
                }
          else {
                tempxf[i][3-l]= DPOLY[m][0];
                tempzf[i][3-l]= -DPOLY[m][1];
                }
          tempyf[i][3-l]=DPOLY[NPTS][0];
          } /* end of for l */
        } /* end of for i */
      pstore();
      } /* end of for k */
      } /* end of for j */
```

```
/* now to make patch of mirror image */
 if(mflg==1){
       for(j=0;j<(count-3);j=j+1){
         for(k= -1;k<(NPTS-2);k++){
          for(i=0;i<4;i++){
          getslice(i+j);
          for(l=0;l<4;l++){
          m=k+l;
          if(m>=NPTS){
               tempxf[i][l]= DPOLY[m-2][0];
               tempzf[i][l]= -DPOLY[m-2][1];
               }
          else if(m<0){
               tempxf[i][l]= DPOLY[-m][0];
               tempzf[i][l]= -DPOLY[-m][1];
               }
          else {
               tempxf[i][l]= -DPOLY[m][0];
               tempzf[i][l]= -DPOLY[m][1];
               }
          tempyf[i][l]=DPOLY[NPTS][0];
          }
         }
       pstore();
       }
       }
 }
}
/* here to retreive horizontal patch slices */

getslice(count)
       int count;
{
       int i;

       for(i=0;i<NPTS;i++){
         DPOLY[i][0]= slices[count][i][0];
         DPOLY[i][1]= slices[count][i][1];
```

```
        }
        DPOLY[NPTS][0] = slices[count][NPTS][0];
}
/* here to store a patch in memory */

pstore()
{
        int i,j;

        for(i=0;i<4;i++){
          for(j=0;j<4;j++){
         patches[patchcount][0][i][j]=tempxf[i][j];
         patches[patchcount][1][i][j]=tempyf[i][j];
         patches[patchcount][2][i][j]=tempzf[i][j];
         }
        }
        patchcount++;
}/* here to ask a question & get an answer */

quest(infile,outfile)
        char infile[],outfile[];
{
        Device val;
        long dev;
        int c,flg,i;

        winset(textwin);
        c3s(whitevec);
        clear();
        c3s(blackvec);
        cmov2(1.0,7.0);
        charstr(infile);
        cmov2(1.0,4.0);
        flg=1; i=0;
        while(flg==1){
        while(qtest()){
        dev=qread(&val);
        switch(dev){
```

```
        case KEYBD:
        if(val=='\b'){
        i--;
        outfile[i]= '\0';
        c3s(whitevec);
        clear();
        c3s(blackvec);
        } /* end of if val */
        else {
        outfile[i++]=val;
        }
        if(val=='\n')flg=0;
        winset(textwin);
        c3s(whitevec);
        clear();
        c3s(blackvec);
        cmov2(1.0,7.0);
        charstr(infile);
        cmov2(1.0,4.0);
        charstr(outfile);
        qreset();
        break;
        case ESCKEY:
        greset();
        gexit();
        exit(0);
        dglclose(-1);
        break;
        case RETKEY:
        flg=0;
        break;
        default:
        break;
        } /* end of switch */
    } /* end of while qtest */
  } /* end of while flg */
  return(i);
}
```