Measurement of Corneal Topography and Transparency.

A Thesis

submitted by

Darryl Anthony de Cunha.

for the degree of

Doctor of Philosophy

Department of Optometry and Visual Science.

The City University, London.

March 1995

Contents

Acknowledgements

I gratefully acknowledge the support and encouragement given to me by Dr. David Thomson and Dr. John Saunders throughout this project. Also I would like to thank Prof. Geoff Woodward for his enthusiasm and knowledge of the cornea and for his use of the scar measurement system and photographs which I have used in chapter 7. I also thank Prof. John Barbur for his invaluable discussions on optical theory and guidance he has given me throughout my work at City University; Prof. R. Fletcher for discussions on topography measurement systems; Mr Roger Buckley of Moorfields Eye Hospital for allowing access to his patients and Mr. Ernie Caswell for construction of the projection system used in this project.

DECLARATION

I grant powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to me. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement.

Glossary

ADC	Analogue to Digital Converter.
A.F.	Apex factor (amount of flattening from the apex).
BF	Barrier filter.
CCD	Charge-coupled device (camera).
CMS	Corneal Modelling System.
C.S.	Contrast Sensitivity.
CTS	Corneal Topography System.
D	Dioptre.
EF	Excitation filter.
PAR	PAR Technology Corneal Topography System
PEK	Wesley-Jessen Photoelectronic Keratoscope.
PIM	Photogrammetric Index Method
Pixel	Picture Element
PRK	Photorefractive Keratectomy
R.of C.	Radius of Curvature.
R.I.	Refractive Index.
SAI	Surface Asymmetry Index.
s.d.	Standard deviation.
SDK	Software Development Kit (for Microsoft Windows)
Sim K.	Simulated Keratometry value.
SLRC	Single Lens Reflex Camera.
SRI	Surface Regularity Index.
V.A.	Visual Acuity.

Abstract

Measurement of Corneal Topography and Transparency.

A new method is described for measuring and modelling corneal topography. The method does not make assumptions about the underlying corneal shape and calculates radius of curvature at any point on the surface. This allows derivation of apex position, apex radius of curvature and rate of flattening away from the apex. The system reconstructs the surface shape independently of alignment of the eye with the instrument and can be applied to both regularly and irregularly shaped corneas.

To measure the corneal surface without any assumption about underlying shape, a system was set up to project vertical planes of light onto the cornea. An image of the diffuse reflection of the planes on the cornea was captured by a computer frame grabber. Intersection points of the light planes with the corneal surface were calculated to give a matrix of x,y,z points lying on the corneal surface. A polynomial surface equation was fitted to these points using a u,v co-ordinate system embedded in the surface. Differential geometry theory was applied to this equation to calculate radius of curvature across the surface. The results are output in the form of a contour map showing the apex position and other numeric parameters including rate of flattening from the apex.

The system has been applied to both normal and keratoconic eyes and has been used to measure keratoconic apex radius of curvature down to 5.1mm. The contour map output gives a good visual impression of the actual shape of the cornea. Further applications would include - 1. Assessment of irregular corneal shape prior to photoablative surgery. 2. Monitoring of progression of Keratoconus. 3. Post operative assessment of corneal grafts. 4. Accurate surface description of aspheric contact lenses.

A method is also developed to give a quantitative measure of corneal scarring. The system is attached to a modified slit lamp for observation of the eye and measurement of backscatter from corneal haze. The results are output as a corneal map showing the location of the scarring and area values for different scar densities. The map and density values are in a convenient form to be kept with patient records, so that improvement or regression in scarring can be accurately monitored.

Chapter 1.

Introduction

1.1 Background

The function of the eyes is to obtain information about the distribution of light in the environment and to convert this information into a suitable form for transmission to the visual areas of the brain. The eyes are therefore the receivers for the visual system and the quality of the final perception is limited by the fidelity of the information obtained from the eyes. This, in turn, is dependent on the characteristics of the optical system of the eye.

The main refracting surface of the eye is the cornea and the shape of the cornea plays a key role in determining the quality of the retinal image and ultimately the quality of visual perception. Small changes in either the shape or the transparency of the cornea can have a marked effect on visual performance. Knowledge of the exact topography of the cornea is therefore of importance both theoretically and clinically. To the clinician, knowledge of corneal topography is of particular interest in detecting and monitoring corneal disease (e.g. keratoconus), contact lens design and fitting, keratoplasty and refractive surgery.

The instrument in most widespread use today for routine assessment of corneal shape is the keratometer. Keratometers take advantage of the reflective properties of the cornea to form an image of an illuminated mire pattern. By measuring the magnification of the image, the curvature of the central portion of the cornea can be estimated. However, standard instruments are incapable of assessing the topography of other parts of the cornea and because the instruments assume sphericity between the mire images, they do not provide meaningful results with irregular corneas. Keratoscopes operate on

the same principle as keratometers, but by using a mire pattern consisting of a series of concentric rings, an estimate of the topography of a larger area of the cornea can be obtained. However, like keratometers, keratoscopes make a number of *a priori* assumptions about corneal shape and are incapable of providing meaningful results when the cornea is irregular.

The advent of laser refractive surgery has renewed interest in corneal shape and transparency measurement (Gartry (1991), Salz (1993), McCarey (1992)). Detection of subtle, but clinically significant, topographical detail are seen as important steps in the refractive surgical procedure (Wilson, 1994). However, at present there are no instruments available which are capable of providing precise topographical information about irregular corneas.

The aim of the research programme described in this thesis was to review current methods for assessing and describing corneal topography and to develop a system which would be capable of providing precise topographical information about any surface shape. A system for assessing the transparency of the cornea is also described.

1.2. The Optical System of the Eye.

The largest change of refractive index in the eye occurs at the front surface boundary between the front of the cornea and the air. The cornea therefore provides the greatest amount of optical refractive power for the eye and contributes approximately two-thirds of the eye's total refractive power (Boff, 1988).

The transparent cornea at the front of the eye is approximately 0.5 mm thick at the centre and 0.9mm thick at the limbus (cornea-scleral junction). The anterior corneal surface is covered by a thin layer of lacrimal fluid and it is at this air-lacrimal fluid boundary that the greatest change of refractive index occurs. The focusing power of

the eye which is usually attributed to the cornea, is in reality due to this air-lacrimal fluid boundary. The anterior and posterior corneal surfaces are often treated optically as being spherical, with radius of curvatures of 7.7mm (anterior) and 6.8mm (posterior). For schematic optical eyes (e.g. Gullstrand) the refractive index of the tear-layer/cornea is taken as 1.375.

Behind the cornea is the anterior chamber which has, on average, an axial length of about 3.1 mm (Longhurst, 1973) and is filled with a watery fluid called the aqueous humour. The refractive index of the aqueous humour is usually taken as 1.336. At the back of the anterior chamber is the pupil which acts as a variable aperture in front of the crystalline lens. The crystalline lens is a biconvex lens which can change its anterior and posterior surface curvatures through action of the ciliary muscle. This change of shape is used to focus images sharply on the retina and this action is called accommodation. When the ciliary muscles are relaxed, the radius of curvature of the anterior lens surface is approximately 10mm and that of the posterior surface is 6mm. The central thickness is of the order of 3.6mm. The internal structure of the lens shows layers of fibres forming a radial pattern with a central biconvex nucleus surrounded by a region called the cortex. The refractive index inside the lens displays a gradient, with a maximum of 1.40 at the centre of the nucleus, to 1.375 at the lens equator. This gradient index gives the lens a greater focusing power than if it were constructed of a uniform refractive index. Peripheral flattening of the cornea and crystalline lens help to reduce the spherical aberration in the eye. This flattening makes the shape of the cornea and lens difficult to measure and also implies that schematic eyes containing spherical optics only approximate the eyes real optical characteristics.

Behind the lens is the vitreous humor, which is a transparent gel with a refractive index of 1.336. At the back of the eye, images are focused on the retina where the light is absorbed by photorecepters.

There are considered to be 3 optically important axes of the eye (Bennett & Rabbetts, 1984(a)). The first axis, called the Optical axis, is the axis through which the 4 Purkinje images are aligned. The Purkinje images are the images formed of an object reflected from the 4 reflecting surfaces in the eye. The images are designated as Purkinje I, II, III and IV corresponding to the four reflection surfaces. These surfaces are the anterior and posterior corneal surfaces and the anterior and posterior lens surfaces. The Visual axis is the axis passing through the centre of the entrance and exit pupils. This represents the best axis along which rays enter the eye for vision. One further axis, called the Pupillary axis, is defined as the line from the centre of the entrance pupil which meets the anterior corneal surface normally. This line also passes through the centre of curvature of the corneal surface intersected by the axis. The angle between the Optical axis and the Visual axis is donated as angle α (alpha) and the angle between the Visual axis and Pupillary axis is angle κ (kappa). These axes and angles are shown below in fig. 1.1.

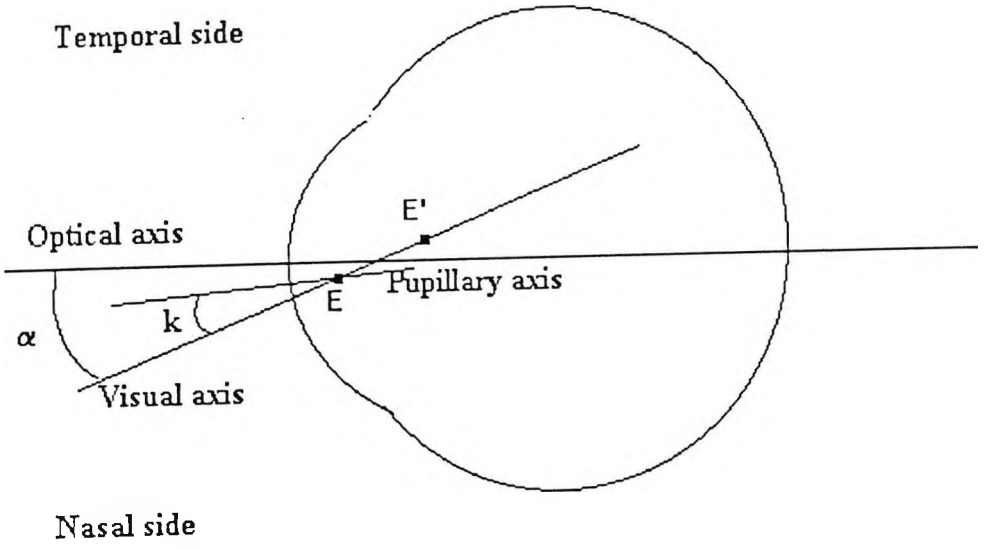


fig. 1.1. View of the eye showing the relationship of the different axes.

Points E and E' are the centres of the entrance and exit pupils respectively.

The value of angle α can be measured using an instrument called Tscherning's Ophthalmophakometer (Bennett & Rabbetts, 1984(b)). This uses a telescope placed at the centre of a circular arc with the eye placed at the arc's centre of curvature. Above and below the telescope, two lamps are placed to produce two sets of Purkinje I, III and IV images which are viewed through the telescope. A fixation point is then moved until the sets of Purkinje images are seen to be in best vertical alignment along the axis of the telescope. The telescope axis is then coincident with the eye's optical axis and the angle α can be found from the position of the fixation point . In practice, the optical axes of the refracting surfaces in the eye are not coincident so there is no position where the Purkinje images are in perfect alignment.

1.3. Anatomy of the eye.

1.3.1. Tears.

The lacrimal fluid or tears are produced mainly by the lacrimal gland (Crouch, 1978), located in the upper lateral side of the eye orbit which opens into 6 to 12 ducts leading to the upper part of the palpebral conjunctiva. The tears move over the anterior corneal surface at each blink and form a precorneal film. Following each blink, the tear fluid flows in a medial direction and empties into two small lacrimal ducts leading to the lacrimal sac. The lacrimal sac drains via the large nasolacrimal duct into the inferior nasal meatus. The functions of the tears are to help regenerate corneal epithelium cells, remove discarded cells and provide a smooth surface for refraction of light entering the eye. The structure of the tear film shows three distinct layers; an outer lipid layer 100nm thick above an 8 micron aqueous layer and below which is a mucous layer approximately 1 micron thick in contact with the corneal epithelium's microvilli surface. The lipid layer prevents evaporation of the aqueous layer and therefore stabilises the tear film. The condition of the lipid layer has no effect on the corneal curvature value as measured by keratometric methods (Lamberts, 1983), but instillation of flourescein into

the tear film does affect the stability of the lipid layer and so also affects the rest of the tear film. Many non-keratometric corneal measurement systems such as the one developed in this project involve the introduction of fluorescein into the tear layer, so possible consequences on tear film stability must be recognised as important. Below the lipid layer is the aqueous layer which contains the necessary salts, sugars, proteins and vitamins to maintain a healthy cornea. Anti-microbial substances are also contained in tears which help protect the cornea from bacterial infection.

Motion of tears remove cellular debris, circulate oxygen required for healthy epithelial cell development and remove waste carbon dioxide. Adjacent to the aqueous layer is a semi-solid mucus layer which adheres to the anterior surface of the corneal epithelium. This layer gradually merges into the aqueous layer and helps protect the epithelium from infection.

1.3.2. Epithelium.

The anterior cellular surface of the cornea, which is separated from the air by the precorneal tear film, has a uniform thickness of 50 to 60 microns and constitutes 10% of the total corneal thickness (Bergmanson ,1991). The internal limit of the epithelium is a membrane known as Bowman's layer or membrane. At the perimeter or limbus of the cornea, the corneal epithelium is continous with the conjunctival epithelium. The epithelium layer consists of 5 to 6 layers of cells, whose function is to resist damage to the central delicate layers of the cornea. The epithelium must also repair rapidly when it is damaged. This is achieved by moving existing cells to an area of damage and regeneration to replace lost cells. The surface of the epithelium shows a fine structure of irregularities (microvilli), providing a large surface area for bonding to the tear film. Corneas with epithelial damage show a reduced ability to maintain an intact tear layer. The cell structure of the epithelium layer shows cell migration from the basement membrane (Bowman's layer) to the anterior surface. At the basement membrane, cells are young with a circular appearance and are called basal cells. As the cells develop,

they move through the cornea towards the anterior surface becoming flatter in appearance and are described as 'wing' cells. Finally the cells reach the corneal anterior surface as very flat 'squamous' cells and are removed by tears having reached the end of their life cycle.

The epithelium also contains a large number of nerves to provide a response to physical stimuli. Between 70 and 80 nerves enter the cornea radially, passing through the cornea and ending in the basal epithelium layers. The energy requirements for rapid regeneration of cells necessitate oxygen intake from tears and glucose uptake from the aqueous humor via posterior corneal layers.

1.3.3. Stroma.

The stroma lies below the corneal epithelium and forms 90% of the thickness of the cornea. It comprises of layers of lamellae lying on each other (Davson,1972), with collagen fibrils running parallel within each lamella. The lamellae are composed of 78% water, 15% collagen and 5% other proteins. Between the lamellae are scattered cells called Keratocytes and also nerve fibres which weave around the lamellae. The lamellae constitute a solid framework which resists disruption to fibrils but allows diffusion of nutrients through the structure. The stroma also exhibits negative 'imbibition' pressure of 60mmHg which has the effect of sucking fluids into the stroma which helps preserve the regular structure of the stroma. To keep the stroma in equilibrium, the outer corneal layers pump out any excess fluid from the stroma. The transparency of the stroma is thought to depend on the parallel nature and spacing of the collagen fibrils in the lamella (Maurice, 1957). The configuration of the fibrils is such that incident light is either transmitted, or reflected with destructive interference producing no reflection.

1.3.4. The Corneal Endothelium and Descemet's Membrane.

The corneal endothelium is a single cell layer comprising around 400,000 cells, each of 25 micron diameter on average (Hirst ,1991). Each cell has a regular hexagonal shape in the normal cornea, but a less regular structure in abnormal corneas. If the

endothelium layer suffers damage, such as surgical trauma, the wound healing will not produce normal shaped endothelium calls. To maintain corneal clarity, excess fluid must be removed from the cornea and this is the principle function of the endothelium layer. The layer has a passive pump action in the form of a permeable membrane, removing excess fluid from the stroma to the aqueous humor. If the stroma fluid content is not in equilibrium but is much higher than normal, an active pump mechanism comes into operation. This mechanism pumps ions into the aqueous humor reducing stromal swelling. At the base of the endothelium on the endothelium-stroma boundary is Descemet's Membrane. Descemet's Membrane is secreted by the endothelium and forms a 10 micron thick elastic membrane which can recover quickly after deformation.

1.3.5. Loss of corneal transparency.

The most common cause of transparency loss in the cornea is disruption of the stromal lamellae. This may be caused by Oedema, stretching of the cornea or trauma (Benjamin, 1991). The regular lattice structure of the collagen fibrils is disrupted and light scatter occurs from the irregular lattice structure. In some cases scarring within the stroma can be severe, leading to complete corneal opacity. Loss of transparency can also occur in the corneal epithelium layer. Oedema in the epithelium caused by trauma may disrupt the close adhesion of the epithelium to Bowman's layer. The subsequent spaces become filled with fluid causing light scatter. Certain chemical deposits such as calcium, may also occur in the epithelium due to corneal pathologies. Even a small loss of corneal transparency, whether localised or covering the whole of the cornea, can cause a significant reduction in visual performance.

1.4. Corneal Refractive Surgery.

1.4.1 Penetrating Keratoplasty.

In severe cases of corneal disease, surgical correction in the form of a corneal transplant (Caroline, 1991) may be the only method of improving vision. This is true in cases of advanced Keratoconus, where scarring is severe over the visual axis and the cornea is significantly distorted. In these cases the central area of the cornea (up to 8mm diameter) will be totally removed and replaced by a donor cornea. Success rates for this type of operation are better than 90% for the graft to be accepted by the host cornea. One reason for the low rejection rate is the total absence of blood vessels in the normal cornea. The two main post-operative sequelae are irregular astigmatism, which is sometimes severe and tilting of the graft cornea relative to the host cornea. Both conditions are difficult to measure accurately using current topographical instruments, but general areas of steepening and irregularity are visible with photokeratoscopes.

1.4.2. Epikeratophakia.

Epikeratophakia is used to treat severe myopia (Maguire, 1987) and central corneal thinning found in Keratoconus (Kaufman, 1982) when the condition has not reached an advanced stage with severe scarring. A circular explant is sutured on top of the central part of the cornea after removal of the epithelium to reinforce and flatten the conical cornea. This has the advantage over penetrating keratoplasty of less risk during surgery and lower probability of explant rejection relative to graft rejection. The technique cannot be used if the cornea has developed scar tissue because this would not be improved by the explant. As in penetrating keratoplasty, one of the main post-operative problems is residual astigmatism which is minimised by careful adjustment and selective removal of sutures.

1.4.3. Radial Keratotomy.

Radial keratotomy is a technique to reduce myopia (McDonnell, 1989) where incisions are made on the corneal surface from the centre of the cornea to the limbus.

The incisions avoid the central 3mm of the cornea and are equally spaced and radially outward from the visual axis. The technique depends on the incisions weakening the corneal structure which results in a flattening of the corneal surface and a reduction in myopia. The amount of flattening is dependent on the length and depth of the incisions, therefore sucess of the procedure is largely determined by the skill and experience of the surgeon. After the operation, the patient is left with scars where the incisions were made which may cause different amounts of glare depending on pupil diameter.

1.4.4. Photorefractive Keratectomy (PRK)

The surgical procedure for correction of myopia using excimer laser is of particular relevance to this project. In this technique, the surface of the central cornea is cut (photoablated) with high precision, producing a new smooth surface contour. The curvature of the cornea after ablation should be that required to fully correct myopia. (Gartry, 1991).

In PRK, photoablation of the cornea is achieved using an argon fluoride excimer laser, emitting ultraviolet radiation at 193nm. At this wavelength, radiation is absorbed within a few microns of entering the cornea. Each photon has an energy of 6.4 electron volts, which exceeds the binding energy of carbon-carbon bonds in the cornea. Therefore with each laser pulse a layer of only a fraction of a micron will be ablated from the corneal surface. A circular aperture in front of the laser beam produces photoablation of a circular zone on the cornea and damage to unexposed tissue is limited to 300nm from the boundary of the ablation zone. For the photoablated area to result in a smooth clear cornea, certain properties of the cornea must function correctly after healing. The properties which may be disrupted after surgery include epithelial adhesion and cell migration, transparency loss of the stroma caused by disruption of the lamella lattice structure and cell loss in the corneal endothelium.

Prior to photoablation, the epithelium is softened and mechanically removed to expose the stromal layers. When correct fixation is achieved, the laser is pulsed at 10 Hz photoablating the cornea. A flattening of the cornea is achieved by increasing the laser aperture size during the operation, ablating the central areas more than the outer areas. The overall diameter of the ablation zone is typically 4-6mm with a total of 400 pulses applied to the cornea. The depth of tissue removed per pulse has been estimated at 0.24 microns on average, giving a final ablation depth of the order of 0.1mm. Immediately after the ablation is completed, a series of ridges are visible indicating discrete changes in the laser aperture, producing distinct ablation zones. Thirty six hours after treatment, the anterior surface of the cornea has re-epithelialised and shows a smooth appearance but is distinct from the non-ablated area.

Post-operative examination of treated eyes includes measurement of refraction and acuity, corneal thickness (pachymetry), eye pressure (tonometry), fundus examination, visual field examination and keratometry (usually with a computerised system such as EyeSys). Slit lamp examination is also made with subjective grading of corneal haze. This haze does not occur immediately after treatment but seems to appear between 1 and 3 months following surgery and is often accompanied by extensive regression in the corneal refractive state (Seiler, 1994). The corneal haze has been graded (Salz, 1993) as follows: Grade 4 - opacity prevents view of anterior chamber details; Grade 3 - opacity easily visible and markedly interferes with refraction; Grade 2 - haze easily visible and interferes with refraction; Grade 1 - haze easily visible but does not interfere with refraction; Grade 0.5 - barely visible; Grade 0 - cornea clear and ablation zone not apparent. Although it is reported (Salz, 1993) that 99% of treated corneas eventually achieve Grade 1 haze of better, this can still leave the patient with irritating light scatter and ghost images. The current opinion (Taylor, 1994) on the efficacy of treatment is that PRK is acceptable when the result is a clear cornea with an accurate and stable dioptric result.

In addition to corneal haze, further post-operative complications of PRK include regression to myopia, increased corneal astigmatism and loss of best corrected visual acuity. These problems are thought to be due to the reformation and abnormal growth of the corneal epithelium after surgery (hyperplasia), over which there is no control and no method of predicting the final outcome at present. Although most studies of PRK claim that over 90% of patients achieve a refractive correction within ±1.0 D of the required correction for emmetropia and an uncorrected V.A. of 6/12 or better, this usually means that patients still require a correction for acceptable vision. Success rates for results of ±0.5 D or better which would not require further correction are not generally quoted.

Monitoring and prediction of epithelium growth seem not to have been attempted to date, probably due to the irregular nature of post-operative corneal changes. The major topographical change appears to be the formation of central islands on the cornea (Krueger, 1994) covering the central 3mm and producing up to 3 diopters of regression in addition to astigmatism and scarring. Efforts are being made to objectively quantify scarring by measuring backscatter (Braunstein, 1994) but little information on the methods or use are available at present.

In this study, a method to measure surface topography was developed which can be used successfully on irregular corneas such as in post-operative PRK patients as described above. The system could also assist post-operative planning in penetrating keratoplasty to improve astigmatism by adjustment of sutures. It has already been shown in cases of high astigmatism that reference to corneal topography measurements can produce significant improvements (Karabatsas, 1994). In addition to topography measurement, a system was developed to give objective measurement of corneal haze by backscatter measurements. This system was designed to be easy to use in a clinical environment and to provide quantitative data which could be used in further statistical studies for correlation of the effects of haze.

Chapter 2.

Review of Corneal Topography Measurement Systems

2.1. Keratometry.

The instrument in most widespread use today for routine assessment of corneal shape is the Keratometer (sometimes referred to by the older term Ophthalmometer). Keratometers take advantage of the reflecting properties of the cornea to form an image of an illuminated mire pattern of known dimensions. The size of the mire image formed by the cornea is viewed and measured through the Keratometer. If the cornea is treated as a spherical convex reflecting surface, standard paraxial ray equations can be applied to the mire images and the corneal radius of curvature can be calculated. Keratometers are usually designed to make radius measurements in one meridian or in two orthogonal meridians, with the viewing system able to rotate for measurements along the maximum and minimum curvature axes.

The results are usually assumed to represent the shape of the central 3mm of the cornea and can be used as an initial "shape value" when fitting contact lenses. One problem when making direct measurements of the reflected images is image motion caused by eye movements. To overcome this, keratometers typically have an optical system which doubles the image from the cornea (Ruben, 1975). This is achieved by placing a prism system behind half of the Keratometer aperture, producing a second shifted mire image next to the direct image from the cornea. The size and relative displacement of both images depends on the corneal radius of curvature. This displacement is then measured by shifting the images to a fixed separation indirectly giving the radius of curvature value. Image motion caused by eye movements is eliminated because both images move together.

2.2.1. Photokeratoscopes and the PEK.

Keratoscopes have been developed which extend the keratometer principle to measure the entire corneal surface, replacing the keratometer mires with a set of concentric black and white circles. The image of the circles formed by the cornea is viewed through the keratoscope and the shape and size of the circles give an indication of the corneal topography. The simplest form of keratoscope is the hand held Plácido disc, where the ring images are viewed by eye through the disc centre. To record and analyse these images, Gullstrand used photographic recording techniques thus producing the first photokeratoscope (Gullstrand ,1924). A further important refinement of the photokeratoscope was introduced by Knoll (1957) when he placed the concentric circular object rings on a hemispherical surface. The centre of curvature of the hemispherical surface was designed to coincide with the centre of curvature of the cornea under study. If this condition is met, which in general it is not, the image of all object rings will be simultaneously in focus on the photographic plane.

Photokeratoscopes came into relative widespread use with the development of the Wesley-Jessen PEK (Townsley ,1967 ; Bibby ,1976). This is a robust and simple to use photokeratoscope, designed to give enough information about corneal shape to allow contact lens design and fitting from the shape measurements. Seven concentric object rings are internally illuminated in the instrument and their reflected images are photographed on Polaroid film. Quantitative analysis of the corneal shape was carried out by the manufacturers on receipt of the PEK pictures. Before the introduction of computer based photokeratoscopes, this instrument was widely used in the general study of corneal shape in addition to contact lens fitting requirements.

Accuracy measurements of instruments based on the keratometry principle have been made by Hannush (1990) using a Bausch & Lomb keratometer on 4 calibrated steel

spheres showed results within ±0.25 D of the true equivalent power value, (assuming a refractive index of 1.3375 for conversion from radius of curvature to power). These results are typical of these types of instruments, but only after careful alignment of the test spheres.

2.2.2. Keratometry measurement principles.

If the cornea is treated as a convex mirror, topographical information can be derived by measuring the image size of an object reflected from the corneal surface. Applying standard paraxial ray equations to the image and object sizes,the radius of curvature of the equivalent spherical mirror is calculated (Stone ,1975). Fig 2.1 shows the ray diagram for a convex spherical mirror.

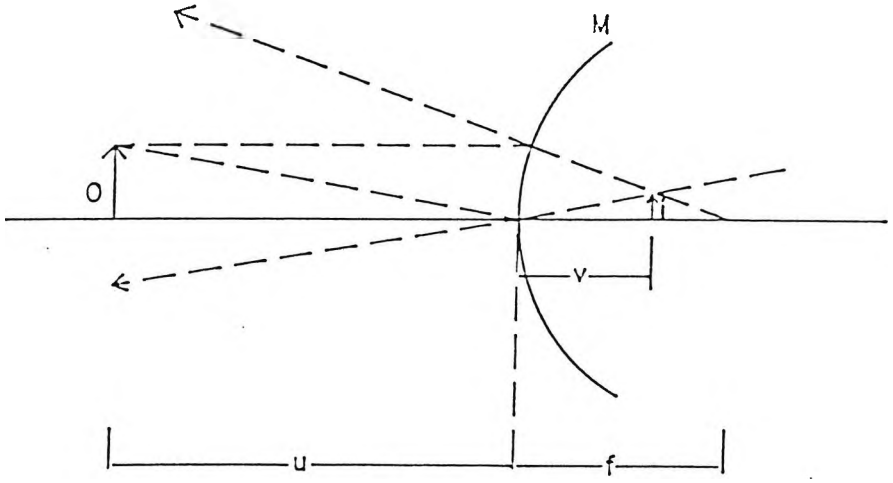


fig. 2.1 Ray from an object O is reflected from a spherical convex mirror M, forming an image at v of height i. The object distance is u and the focal length of the mirror is f.

Considering the image magnification we have

(2.1) $\frac{i}{o} = \frac{v}{u}$

Further, the image distance is assumed to coincide with the focal point of the mirror. Then, if r is the mirror radius of curvature

(2.2) $v = \frac{r}{2}$

combining equation (2.1) with (2.2) and solving for r

(2.3) $r = \frac{2ui}{o}$

Equation (2.3) is the basic equation of keratometry and produces one radius of curvature value for an image height along a given meridian.

In an attempt to build a corneal profile, multi-ring keratoscopes have been developed. The number of rings vary between different instruments, from 7 in the case of the PEK (Bibby,1976) to 32 for the Corneal Modelling System (CMS) (Gormley ,1988). To generate a profile for a given meridian, a reflection point on the surface must be calculated for each object ring. This is achieved using the geometry shown in fig. 2.2 (Townsley,1967).

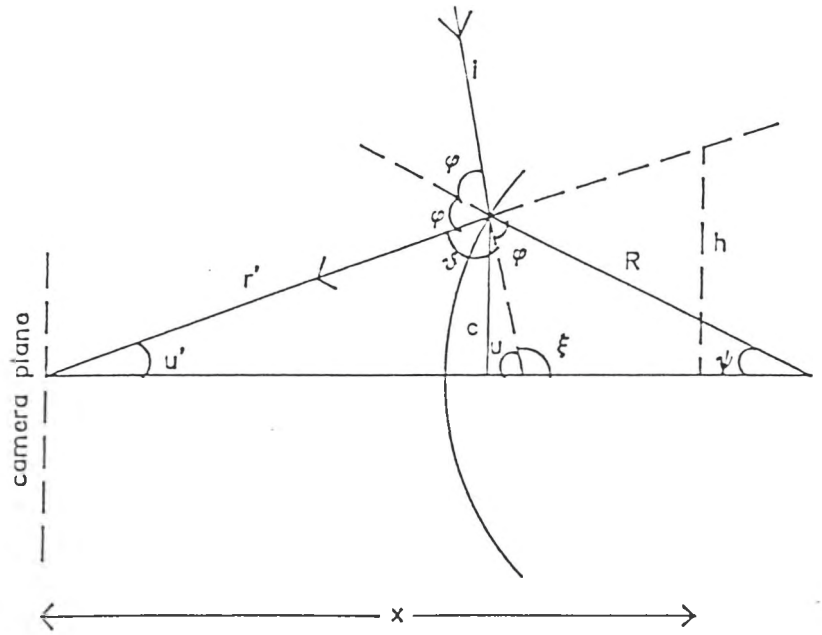


fig 2.2 Geometry for reflection from spherical surface.

i = incident ray from target, r' = reflected ray to camera,

h = image height of ring, R = Radius of curvature of reflecting surface,

x = focussing distance of camera,c = semichord.

To find the incident ray intersection point with the surface, R and ψ must be calculated. For a known image size, R can be found from equation (2.3) and ψ is derived from fig. 2.2 as follows

(2.4) $u' + u + \vartheta = 180°$

(2.5) $\vartheta + 2\varphi = 180°$

(2.6) $\therefore u' + u = 2\varphi$

and

(2.7) $\varphi + \xi + \psi = 180°$

(2.8) $u + \xi = 180°$

(2.9) $\therefore \varphi + \psi = u$

substitute φ into equation (2.6)

(2.10) $u' + u = 2\left(\frac{u - \psi}{\quad}\right)$

(2.11) $\psi = (u - u')/2$

where $\tan u' = h/\ \text{ɔc}$

and $\tan u \equiv o/x$ where o = object ring height.

To build up the profile, the cornea is assumed to have a smooth shape with no abrupt changes in radius of curvature. Consecutive arc elements can then be joined together at the reflection points.

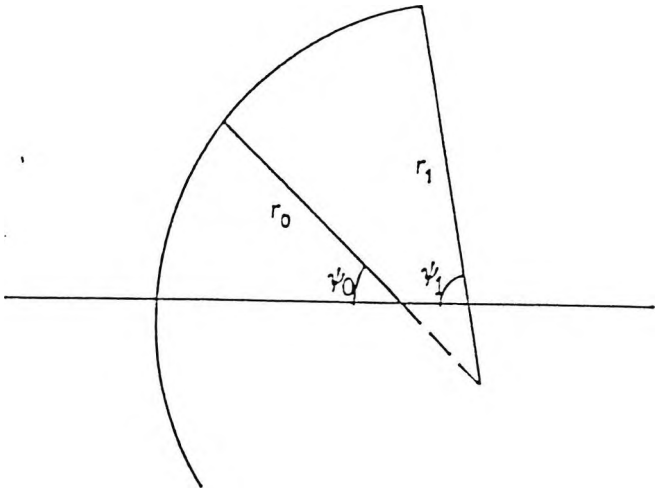


fig. 2.3 Two consecutive arc elements joined at a reflection point.

Fig 2.3 shows the situation for 2 circular sections and also shows the movement of the centre of curvature. Points on the surface in x,y must be found and an ellipse fitted to them to complete the description of the profile. The semichord c in fig.2.2 gives the y value and is found at each reflection point by

(2.12) $y = R \sin\psi$

The x value of each reflection point is calculated by summing the sagittal distance of successive curve elements from the vertex x_0. This is shown in fig. 2.4.

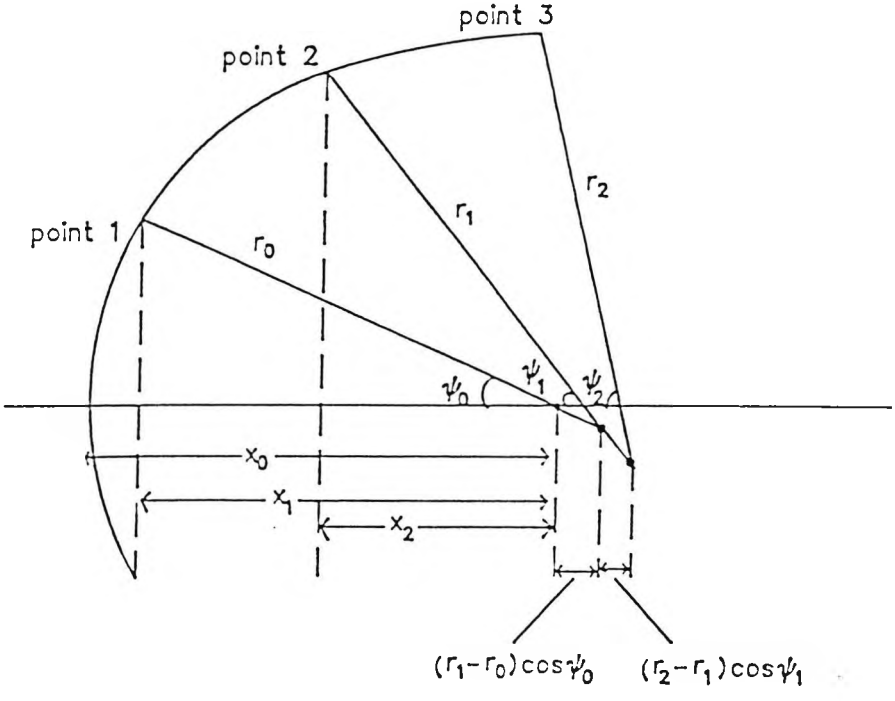


fig. 2.4. Calculating x values of reflection points from curve elements.

In fig. 2.4 x_0 = radius of curvature of central arc r_0.

For the first point

(2.13) $x_1 = r_0 \cos\psi_0$

The first sag value is then

(2.14) sag0 = x0 - x1

For the next point, x2 is given by

(2.15) $x2 = r1\cos\psi 1 - [(r1-r0)\cos\psi 0]$

and the sag value (sag1) is

(2.16) $sag1 = x1 - x2$

For the third point, x3 is given by

(2.17) $x3 = r2\cos\psi 2 - [(r2-r1)\cos\psi 1 + (r1-r0)\cos\psi 0]$

and the sag value (sag2) is

(2.18) $sag2 = x2 - x3$

Hence for successive points the sag distances are calculated and summed from the vertex position to give successive x,y points on the cornea relative to the vertex.

After points on the surface have been calculated, an ellipse or other conic section must be fitted to them to give a description of the surface. Unfortunately the usual ellipse or conic section equations are unworkable when applied to a least squares method of data fitting (Townsley ,1970). The equations must therefore be fitted using an iterative method giving results within some error bound chosen by the operator (Bookstein ,1979 ; Sampson ,1980 ; Porrill ,1990). Because points on the ellipse are only known around the central curvature area, a large number of ellipses posessing identical central curvatures but different shape factors can be fitted to the data (Bibby,1976). This is shown in fig. 2.5.

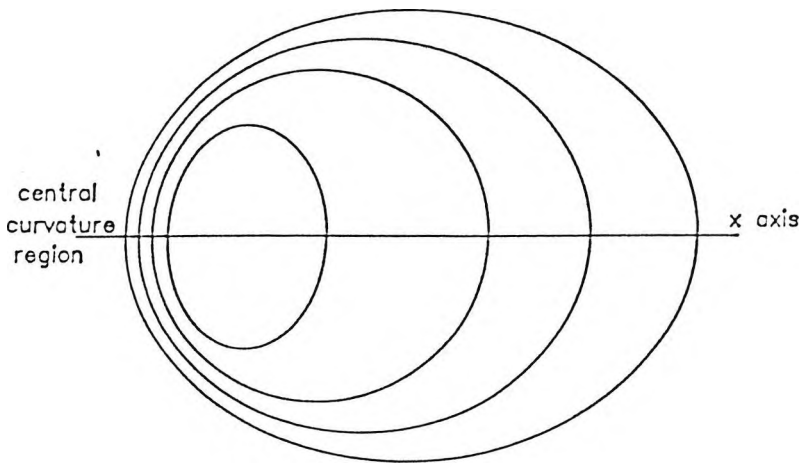


fig.2.5 Different ellipses with the same central curvature.

Therefore in fitting an ellipse, the establishment of the ellipse centre is probably the most important step in achieving acceptable results (Yuen,1989). The first assumption to be made is to place the major axis of the fitted ellipse along the axis of the instrument. Further, the locus of the centres of curvature (evolute) of the curve elements has the minor axis as its asymptote. If the position of the minor axis can be estimated by extrapolating the evolute, then the semi-major axis of the ellipse can be established. This fixes the position of the origin along the x axis.

The equation of the ellipse centered about the origin is given by

(2.19) $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$

a = semi-major axis

b = semi-minor axis

with shape factor given by

(2.20) $s=1-\frac{b^2}{a^2}$

2.3. Autocollimation systems.

A variation on the keratoscope method outlined above, is the measurement of radius of curvature using Drysdales' method (Drysdale ,1900). This method measures the distance between the two positions of keratoscope autocollimation on the corneal surface. The principle is shown in fig 2.6 where rays from the keratoscope are reflected back along their path to form images.

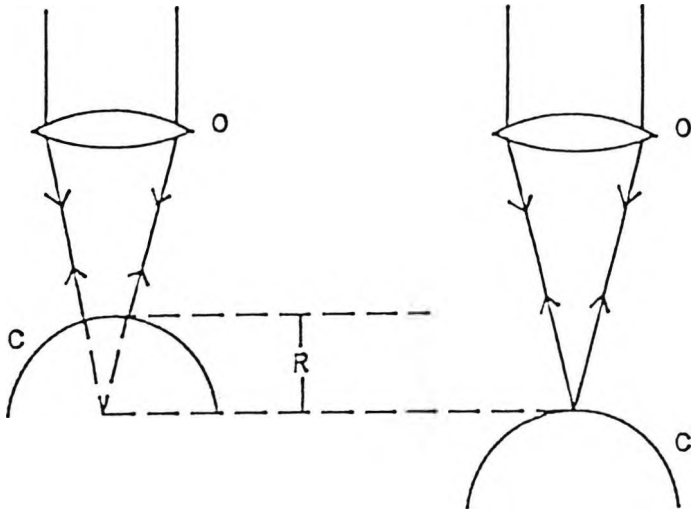


fig 2.6. The keratoscope objective O forms an image from the cornea c at both autocollimation positions. The radius of curvature R of the cornea is the distance between these two positions.

Bennett (1964) described an autocollimating keratometer which allows simultaneous observation of both autocollimation positions. Independent adjustment of both imaging systems, shown in fig. 2.7, allow measurement of the radius of curvature R.

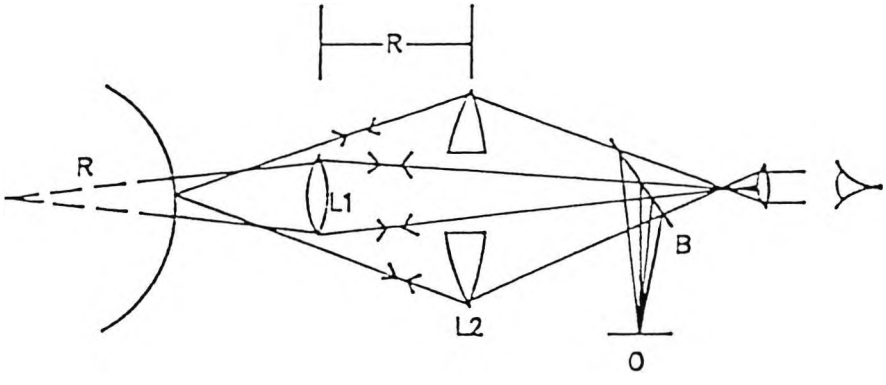


fig. 2.7 Adjustment of both imaging systems L1 and L2 allow measurement of the radius of curvature R. The object O is projected onto the cornea with the aid of a beamsplitter B.

2.4. Collimated keratoscopes.

One theoretical objection to keratoscope measurements in modelling corneal shape, is that the image is constructed from light reflected from different parts of the corneal surface. If the whole of the surface is spherical then no problem arises. However, if the surface varies in curvature, the area of the reflecting surface creating a given image is uncertain. To overcome this problem and also measure over a wide corneal diameter, Fujii (1972) developed a collimated keratoscope. This limits the light forming an image to a narrow bundle of rays reflected from the cornea. The principle is shown in fig. 2.8, where a collimator L projects light onto the cornea from a source at a fixed angle ϑ to the corneal optical axis. The cornea is then viewed by an imaging system with objective O. Behind the objective O is a pinhole P which only allows a narrow bundle of rays to be imaged on the focal plane F.

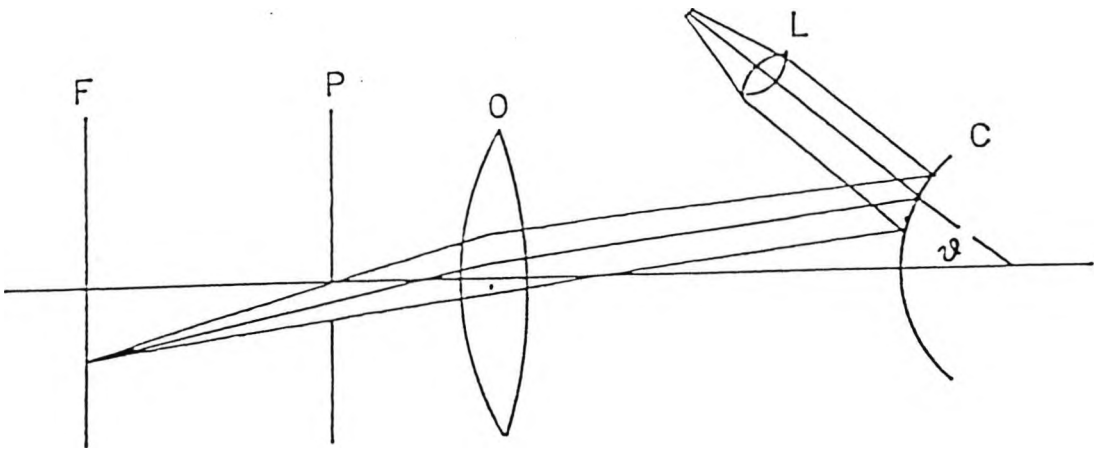


fig. 2.8 Principle of collimated photokeratoscopes. Light from a collimator L is reflected from the cornea C. After passing through a pinhole P the light is focused at the focal plane F.

The distance of the image from the optical axis is used to calculate the angle of reflection from the corneal surface. In practice, Fujii (1972) used 18 collimated light sources arranged in a semi-circle in a plane in front of the cornea. A camera was used to record an image of the sources covering a width of about 11mm on the cornea, from which the corneal curve could be calculated for a given meridian.

2.5 Computerised photokeratoscopes.

The Corneal Modeling System (CMS) (Gormley ,1988 , Wilson ,1991) developed by Dennis Gormley and Computer Anatomy Inc. (New York NY) is a photokeratoscope with an integrated computer imaging system to process the ring images. The computer imaging system replaces the photographic camera with a video camera of 500 lines/frame attached to a computer frame store. The CMS can be operated with a maximum of 32 concentric object rings with each ring evaluated at 256 equally spaced points. The results are output graphically on a computer screen as a colour coded map. Radius of curvature 'r' values have been converted to power 'p' values using the formula p=(n-1)/r where n is the corneal refractive index taken as 1.3375 (Hannush ,1989). Each colour on the colour coded map represents a specific power interval and can be displayed in either an absolute scale of 2.5 dioptre interval or a relative scale down to 0.2 dioptre interval. In addition to the colour coded map, general shape descriptors of the cornea are also calculated. These include -

1. Simulated Keratometry Value (Sim K) which averages results from rings 7,8,9 approximating a zone of 2.5mm radius on the cornea. The power and location of the steepest and flattest meridians are calculated in either spherocylindrical mode with meridians at 90°, or non-spherocylindrical mode independent of angle.

The Sim K parameter represents probably the most useful measure produced by the system, as it allows direct comparison with other keratometric instruments. In studies using the CMS, it is the parameter most often quoted as a measure showing significant changes in topography.

2. Surface Asymmetry Index (SAI) measures the power difference of points 180° apart for 128 equally spaced meridians on the 4 inner mires.

3. Surface Regularity Index (SRI) sums 'local' power fluctuations around 256 equally spaced hemimeridians over the 10 central mires.

Hannush (1989) measured the accuracy of the CMS using 4 calibrated steel test spheres of radius 8.73mm, 7.85mm, 7.84mm, 6.73mm. A refractive index of 1.3375 was used to convert between radius of curvature and dioptric power. The accuracy using rings 2 - 26 showed power measurements with a range of ±0.26 D and a mean error of +0.10 D and s.d. = 0.07 D. Ring 1 was found to be very variable and was not included in the assessment. This should have been included to give a more accurate reflection of the overall system performance. Ring 8 corresponds to the usual measurement ring of keratometers (at 1.5mm radius from the centre).

Another computerised photokeratoscope similar to the CMS and now commercially available is the EyeSys Corneal Analysis System (EyeSys Laboratories, Houston, Tex) (Tsilimbaris ,1991). This system also displays the radius of curvature results as a colour coded power map of similar form to the CMS, but with additional information of average power, location and power difference of the two main astigmatic meridians for 3mm, 5mm and 7mm diameter zones.

McCarey

The accuracy of the EyeSys Corneal Analysis System has been measured by (1992) using 4 calibrated steel spheres. Using reflected mire images at 3mm, 5mm and 7mm radius from the centre of the instrument axis, the average spherical equivalents for the spheres showed an error ranging from +0.29 D to +0.46 D. These results assume accurate alignment of the test spheres with the instrument axis.

Accuracy measurements on a radially aspheric surface was made by Roberts (1994). Results from an ellipsoid of 7.5mm apical radius of curvature and 0.5 eccentricity showed errors of up to 3.00 dioptres at 4mm radius from the apex. This shows the order of magnitude of the inherent error in photokeratoscopes, when measuring surfaces which are non-spherical.

2.6. Corneal profile measurements.

Direct photography of the corneal profile has been successfully carried out by McMonnies(1971). The cornea under study was positioned in front of a brightly illuminated background and light passing tangentially across the cornea was viewed by a photographic system as shown in fig. 2.9

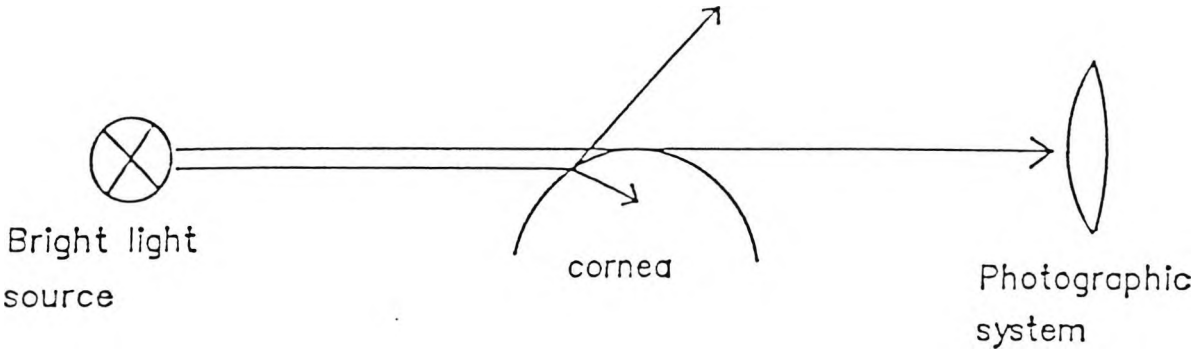


fig. 2.9 Recording the corneal profile.

Light incident on the cornea from the background is either reflected away from the photographic system, or refracted into the eye therefore the cornea always appears opaque with a high contrast and sharp profile. The profile was recorded on very fine grain film (5 - 10 ASA) and enlarged onto Kodalith orthoplates. The profiles were measured using a travelling microscope designed for stereophotogrammetric plotting.

Two different methods of analysis of results were proposed by McMonnies(1971). The first method fitted a circular curve segment between two points on the profile. The results on test spheres indicated acceptable accuracy for curve segments with chord

length greater than 0.86mm. This method gave a computed radius value of 8.046mm for a true radius of 8.001mm. A second method of analysis fitted a polynomial of the fourth degree to all points. If the fitted function y is a polynomial in x, then the radius of curvature at any point on the curve can be found from the formula

$$\text{(2.21)}\quad \text{Radius of curvature} = \frac{\left[1+\left(\frac{dy}{dx}\right)^2\right]^{\frac{3}{2}}}{\frac{d^2y}{dx^2}}$$

These methods give topography along a single meridian.

2.7. Moiré fringe topography measurement.

The technique of projection type Moiré fringe analysis is well known in engineering and has been sucessfully applied to the human cornea by Kawara (1979). It is fundamentally different to keratometry in that the eye is not used to create an image of an object, but instead an interference pattern is created by two grating images (Idesawa ,1977 , Mandel ,1966). The first grating image is created by projecting a grating pattern onto the corneal surface. The second grating image is formed by placing a reference grating in the imaging system used to view the cornea. The grating pattern on the cornea is superimposed on the reference grating generating moiré contour fringes. A simplified schematic diagram of the system used by Kawara (1976) is shown in fig.2.10

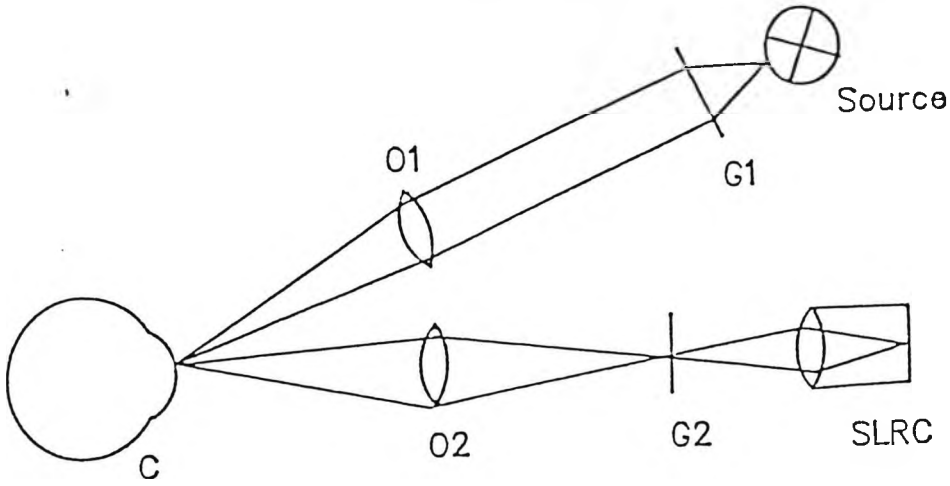


fig. 2.10 A grating G1 is imaged on the cornea by O2. Objective O2 of the viewing system focuses the corneal image on the reference grating G2 and the resulting interference pattern is recorded by camera SLRC.

To achieve a visible grating pattern image on the cornea, 2% sodium fluorescein is instilled onto the subject's cornea to act as a diffusing agent. This has the additional advantage of emitting yellow-green light when illuminated by blue light of the correct wavelength. The iris image could then be removed by chromatic filtering. From the photograph of the interference pattern, the location of each fringe was measured using a microdensitometer. The fringe separation gives the depth interval on the cornea and depends on the period of the projected grating and the angle of projection.

Typical results show a grating pattern visible out to the corneal periphery, with the fringe density increasing radially outwards from the centre of the cornea. The system used by Kawara (1979) achieved a depth resolution of ±0.005mm on a reference sphere.

2.8. Moiré fringe analysis theory.

The general principle of the method is shown in fig. 2.11 where a grating is placed in front of a curved surface 's' under examination and is illuminated at angle i.

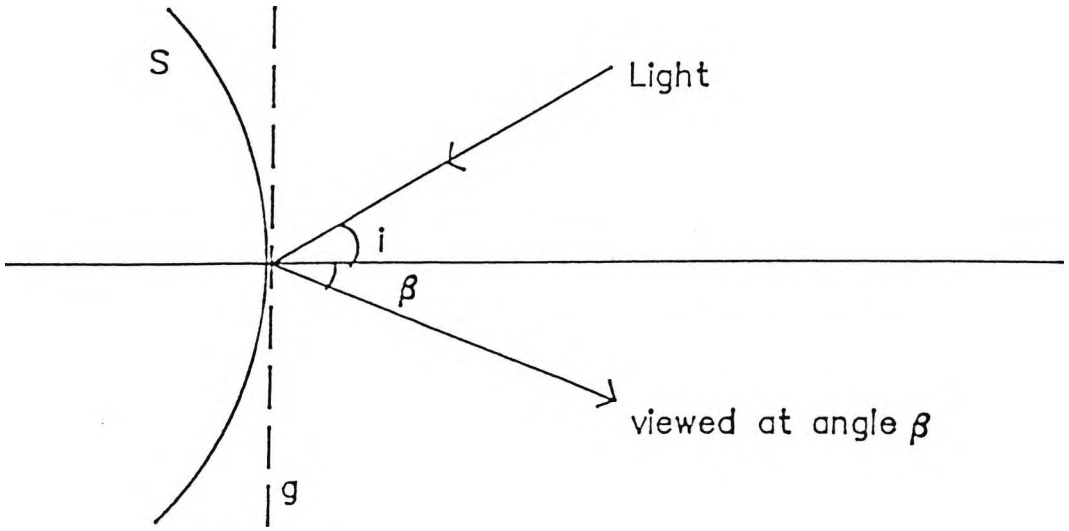


fig 2.11 A matt surface s is illuminated through a grating g. The angle of illumination is i and the image is viewed at angle β

A shadow of the grating falls on the surface and is viewed at angle β by an observer looking through the grating. The observer sees an interference pattern between the grating and the grating image on the surface. This pattern takes the form of light and dark bands or 'fringes', with the dark fringes occuring where the dark band of the image can be seen through the light bands of the grating. This is shown in fig. 2.12.

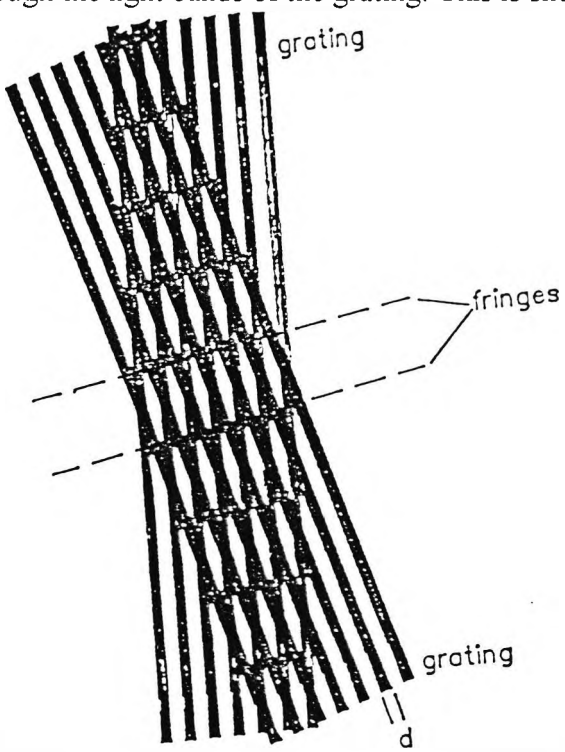


fig. 2.12. Grating image is viewed through grating of width d, producing a series of interference fringes.

The method of calculating the surface depth (Theocaris ,1969) at a fringe position is illustrated in fig. 2.13.

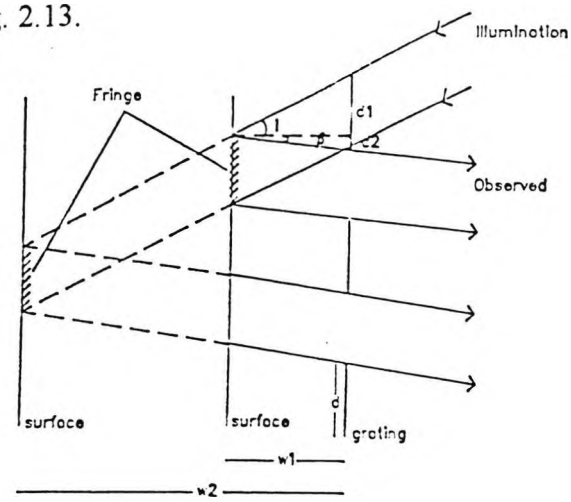


fig. 2.13 The two positions of a surface at distance w1 and w2 from the grating produce successive fringes.

For a grating illumination angle i and viewing angle β, the first shadow can be seen creating a fringe when the surface is a distance w1 behind the grating.

For a grating band width of d, the first fringe occurs at

(2.22) $\tan i + \tan \beta = d1/w1 + d2/w1 = (d1+d2)/w1 = d/w1$

giving a depth at the first fringe of

(2.23) $w1 = d / (\tan i + \tan \beta)$

When the surface distance to the grating is w2, another fringe is observed. For this second position

(2.24) $\tan i + \tan \beta = 3d/w2$

and

(2.25) $w2 = 3d/(\tan i + \tan \beta)$

The surface height difference between successive fringes is

(2.26) $\Delta w = w2 - w1 = 2d/(\tan i + \tan \beta)$

Depths on the surface can then be measured by marking the centre of the first fringe and simply counting subsequent fringes, taking the distances to the centre of each fringe.

2.9. Stereophotogrammetry.

The basic stereophotogrammetric method of calculating 3-dimensional position uses 2 images of an object taken simultaneously from 2 different points of view, as shown in fig. 2.14

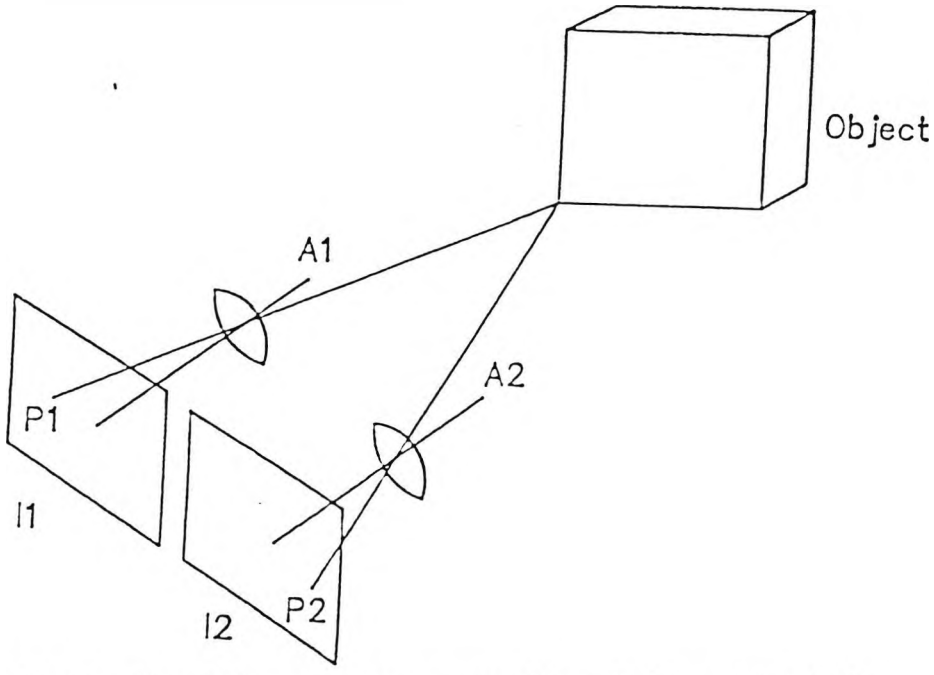


fig 2.14 I1 and I2 are camera systems with optical axes A1 and A2 respectively. The images P1 and P2 of a point on the object show a relative shift in the photographs.

The relative shift of the image position between the two photographs allows the 3-dimensional position and shape of the object to be reconstructed. Bertotto (1948) successfully studied the anterior corneal surface using this method after powdering an anaesthetised eye with lamp black to render the cornea opaque. Pictures of the eye were taken with a Zeiss stereoscopic camera and analysed using a

Wild A5 Autograph. Horizontal profiles were measured which corresponded to transverse sections through the eye, with vertical intervals of 1mm between each profile. Bertotto was able to make measurements out to the corneal limbus with a depth resolution of 0.017mm.

2.10. Stereophotogrammetry measurement principles.

Stererophotogrammetry techniques have been used in aerial land surveying for many years and were first applied sucessfully to corneal topography by Bertotto (1948). The imaging configuration is shown in fig 2.15 where O is a point on the object and p1,p2 are 2 independently recorded images of O. The position of O is measured in x,y,z cartesian coordinates aligned with P1.

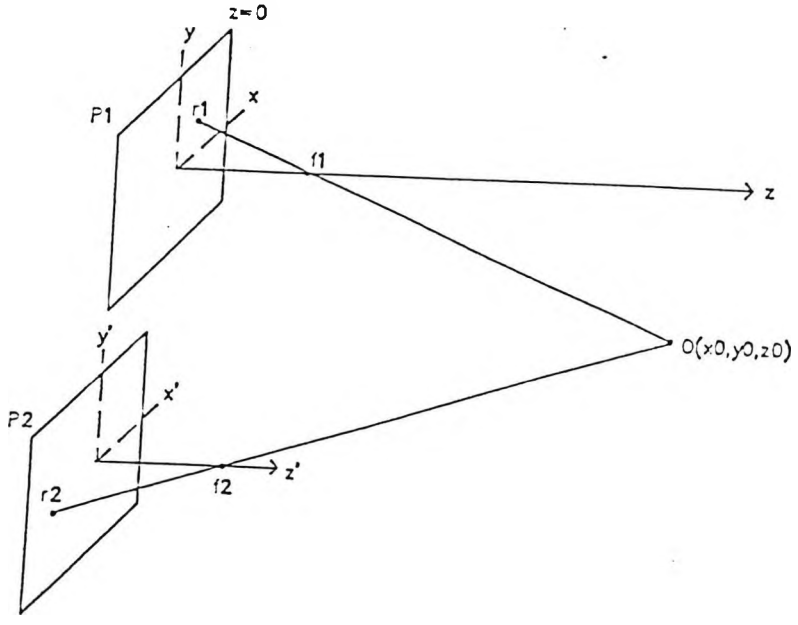


fig. 2.15. Stereogrammetry imaging of object O by 2 cameras P1 and P2, record images of O at r1 and r2 respectively

If P1 and P2 are camera image planes, then the focal lengths of each imaging system are f1 and f2 respectively from the centre of each image plane. The image of O is located at r1 on P1 and at r2 on P3. The image plane P1 has its own 'local' co-ordinate system with r1=(xi,yi,zi) and is coincident with the globally defined system. The image plane P2 also has its own local co-ordinate system with r2=(xi2,yi2,zi2), and the centre of the system shifted by (x2,y2,z2) with respect to P1. The object point O is at (x0,y0,z0) in the global co-ordinate system. The image point r1 is along a ray from O through the focal point at f1. In vector form this is given by

(2.27) $\bar{r}_1 = \bar{f}_1 + t(\bar{O} - \bar{f}_1)$

where t is a distance parameter

In matrix form (2.27) becomes

(2.28) $$\begin{bmatrix} x_i \\ y_i \\ 0 \end{bmatrix} = \begin{bmatrix} o \\ o \\ f_1 \end{bmatrix} + t \begin{bmatrix} x_0 \\ y_0 \\ z_0 - f_1 \end{bmatrix}$$

solving for t using the z component gives

(2.29) $0 = f_1 + t(z_0 - f_1)$

and

(2.30) $$t = \frac{f_1}{(f_1 - z_0)}$$

substituting t (2.30) back into equation (2.28) and solving for x_0 and y_0 gives

(2.31) $$x_i = \frac{f_1}{(f_1 - z_0)} x_0$$

(2.32) $$\therefore x_0 = \frac{x_i(f_1 - z_0)}{f_1}$$

and

(2.33) $$y_i = \frac{f_1}{(f_1 - z_0)} y_0$$

$$(2.34)\ \therefore y_0 = \frac{y_i(f_1 - z_0)}{f_1}$$

The second image of O in P2 is located at r2 with local co-ordinates

$$(2.35)\ \vec{r}_2 = \begin{bmatrix} x_{2i} \\ y_{2i} \\ 0 \end{bmatrix}$$

and with corresponding global co-ordinates

$$(2.36)\ \vec{r}_2 = \begin{bmatrix} x_{2i} + x_2 \\ y_{2i} + y_2 \\ z_2 \end{bmatrix}$$

The focal point of P2 in global co-ordinates is

$$(2.37)\ \vec{F}_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 + f_2 \end{bmatrix}$$

and the image point in P2 is located on a ray given by

$$(2.38)\ \vec{r}_2 = \vec{F}_2 + \beta(\vec{O} - \vec{F}_2)$$

which in matrix form is

$$(2.39)\ \begin{bmatrix} x_2 + x_{2i} \\ y_2 + y_{2i} \\ z_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} + \beta \begin{bmatrix} x_0 - x_2 \\ y_0 - y_2 \\ z_0 - z_2 - f_2 \end{bmatrix}$$

β can be solved using the z component equation

$$(2.40)\ \beta = \frac{f_2}{(f_2 + z_2 - z_0)}$$

substituting β back into (2.39) to solve for x_0

$$(2.41)\ x_2 + x_{2i} = x_2 + \beta(x_0 - x_2)$$

therefore

$$(2.42)\ x_0 = x_{2i}\frac{(f_2 + z_2 - z_0)}{f_2} + x_2$$

equating (2.42) with (2.32) gives the solution for the z component

$$(2.43)\frac{x_i(f_1 - z_0)}{f_1} = \frac{x_{2i}(f_2 + z_2 - z_0)}{f_2} + x_2$$

$$(2.44) z_0\left(\frac{x_{2i}}{f_2}-\frac{x_i}{f_1}\right)=x_2+x_{2i}-x_i+\frac{x_{2i}z_2}{f_2}$$

$$(2.45) z_0=\left(\frac{f_2 f_1}{x_{2i}f_1-x_2 f_2}\right)\left[x_2+x_{2i}-x_i+\frac{x_{2i}z_2}{f_2}\right]$$

Substituting $z_0$into equations (2.32) and (2.34) gives the x_0, y_0, z_0 position of O. The parameters f_1, f, x_2, z_2 are measured during the calibration process.

In the last 5 years, the stereogrammetric method has been modified to allow analysis using a computer imaging system. This form of photogrammetry has been termed Rasterstereography (Warnicki ,1988 , Arffa ,1989). The hardware configuration first developed by Warnicki (1988) replaces one of the cameras of the stereophotogrammetric pair in fig. 2.14 by a projection system which projects a grid or vertical line pattern onto the cornea. To increase the diffuse reflectance of the cornea, sodium fluorescein is instilled in the tear film. An image of the grid on the cornea is produced using a flash illumination system and recorded by a video camera through a sodium fluorescein filter. The image is then digitised by an image processor ready for computer analysis. The system utilises a modified Zeiss stereo photo slit lamp for ease of operation. A schematic diagram of the system is shown in fig. 2.16.

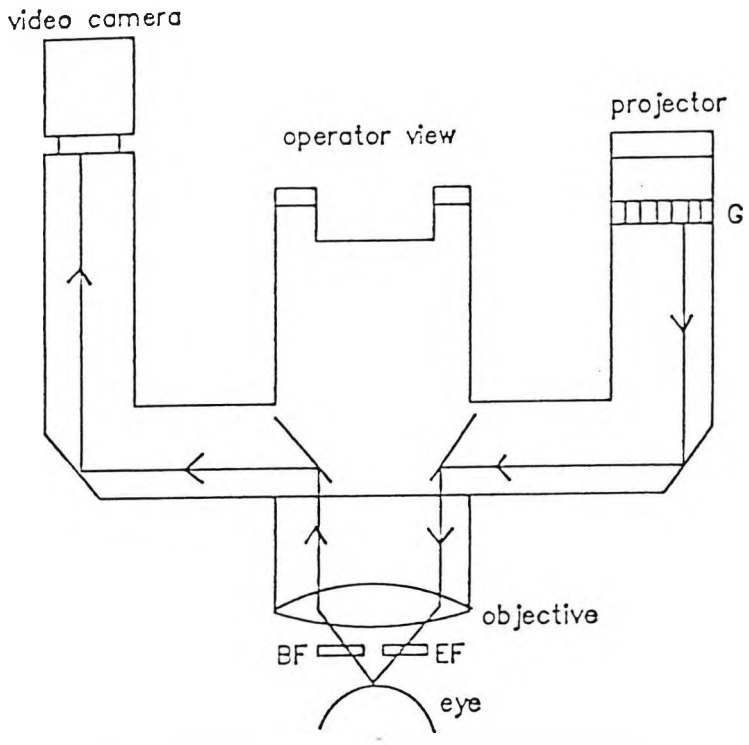


fig 2.16 Diagram showing modified slit lamp with grid G projected onto the cornea through excitation filter EF. An image of the grid on the cornea is viewed by a camera through a barrier filter BF.

The corneal elevation is calculated trignometrically by comparing the horizontal displacement of the grid on the cornea, relative to the grid position on a flat reference plane. A 2-dimensional matrix of elevation points is produced with approximately 1500 points in total. The radius of curvature of the cornea along any particular line is calculated by fitting the best arc to the elevation points along the given line. The results are displayed as a contour plot with each contour at a constant height on the cornea. The accuracy of Warnicki's Rasterstereography system is of the order of 0.10mm in calculating radius of curvature for calibrated steel balls with a depth resolution of 10 microns. The depth resolution may be increased to 4 microns by increasing the magnification (Arffa, 1989).

A version of the Rasterstereography system has now been developed, known as the PAR Technology Corneal Topography System (Belin, 1992), but is not commercially available at present. The PAR system utilises a modified Topcon Slit lamp and projects a grid onto a flourescein-stained tear layer to extract a maximum of

1700 points for analysis. The results are displayed as either an elevation colour coded map; a spherical subtraction map showing the corneal deviation from a best fitted sphere; or a profile display showing the actual points along any meridian relative to the best fit curve. This system probably represents the most advanced corneal measurement system available at present.

2.11. Laser holography

The systems described above may soon be superseded by laser interferometry systems designed specifically for measurement of corneal topography (Troutman ,1992). A laser system, already commercially available, is the KM-1000 CLAS Corneal Topography Unit from Kerametrics Inc. Details and results using laser systems are not well documented to date.

2.12. Laser holography principles.

The use of laser holography in eye research was first suggested by Wray (1970a)(1970b) and has now been successfully applied to the study of corneal astigmatism (Troutman ,1992). To understand how 3-dimensional measurements can be made using holograms, the principles of holography must be known(Vest ,1979).

Basic wave theory states that a travelling wave in one dimension can be described by the function

(2.46) $\Psi(x,t) = A\sin(\omega t - kx)$

A wave in three dimensions is given by

(2.47) $\Psi(\bar{r},t) = a(x,y,z)\sin(\omega t - \bar{k}\bullet\bar{r})$

or in complex form

(2.48) $\Psi(\bar{r},t) = a(x,y,z)e^{-i(\omega t-\bar{k}\bullet\bar{r})}$

If only interested in the spatial structure the time dependence when dealing with light can be ignored as the temporal frequency is of the order of 10^{15}Hz.

If the wave function is given by

(2.49) $u(x,y,z) = a(x,y,z)e^{-i(\bar{k}\cdot\bar{r})}$

where

(2.50) $\bar{r} = x\hat{x} + y\hat{y} + z\hat{z}$

and

(2.51) $\bar{k} = 2\pi\left(f_x\hat{x} + f_y\hat{y} + f_z\hat{z}\right)$

then equation (2.49) becomes

(2.52) $u(x,y,z) = a(x,y,z)e^{-i2\pi(f_x\hat{x}+f_y\hat{y}+f_z\hat{z})}$

Specifying the spatial frequencies for a wave given by (2.52) determines the direction of propagation. This is shown in fig 2.17

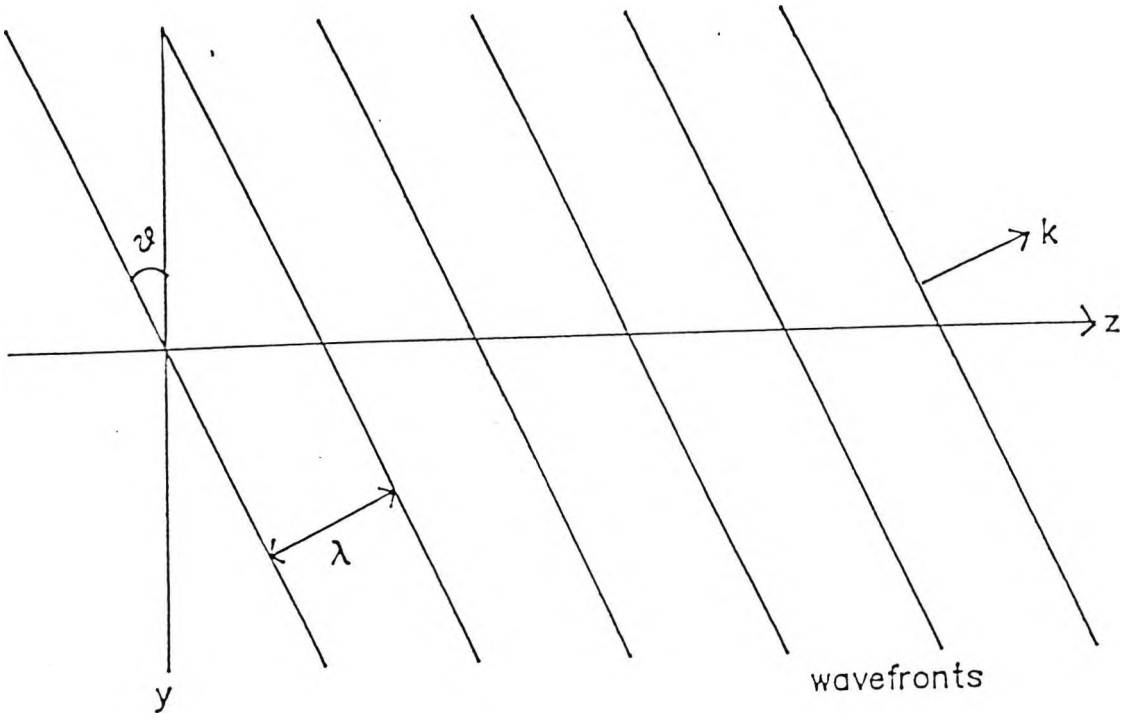


fig 2.17 Wave propagating in direction $\bar{k}$ with wavelength λ.

The wavefronts intersect each axis with an interval equal to the wavelength along that axis.

For example, a wave propagating in direction $\bar{k}$ as in fig 2.17 has a wavelength along the y axis of

(2.53) $\lambda_y = \dfrac{\lambda}{\sin \vartheta}$

the reciprocal $\dfrac{1}{\lambda_y}$ then gives the frequency f_y. If $\bar{k}$ changes w.r.t. the y axis then f_y also changes.

Examining the transmission of a wave through a sinusodial grating, if the grating is along the y-axis and has frequency F_y and the wave propagates in the z-direction as shown in fig 2.18, then the transmission through the grating is

(2.54) $t(x,y) = t_0 + t_1 \cos(2\pi F_y y)$

where

t_1 = amplitude of grating
t_0 = Bias level of grating

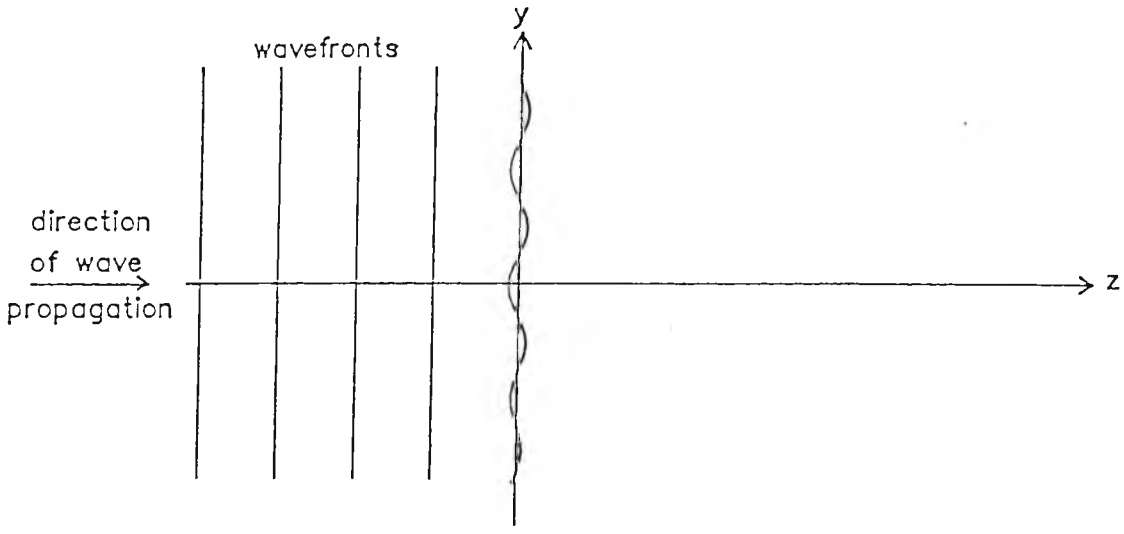


fig 2.18 Wave propagating in z-direction and incident on grating G.

In the z-direction the incident wave is

(2.55) $u_i = a_1 e^{-j\frac{2\pi}{\lambda}}$

and in the plane of the grating (z=0)

(2.56) $u_i = a_i$

therefore the transmitted wave is given by

(2.57) $u_0 = tu_i = a_1 t_0 + a_1 t_1 \cos(2\pi F_y y)$

expanding $\cos(2\pi F_y y)$ using

(2.58) $\cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2}$

and adding a propagation term,equation (2.57) then becomes

(2.59) $u_0 = a_1 t_0 e^{-i\frac{2\pi}{\lambda}} + \frac{a_1 t_1}{2} e^{i2\pi\left(F_y y - \frac{z}{\lambda_z}\right)} + \frac{a_1 t_1}{2} e^{-i2\pi\left(F_y y + \frac{z}{\lambda_z}\right)}$

The first term in equation (2.59) is a wave directly transmitted through the grating, while the second and third terms are waves diffracted in a direction determined by the frequency of the grating. This is illustrated in fig. 2.19.

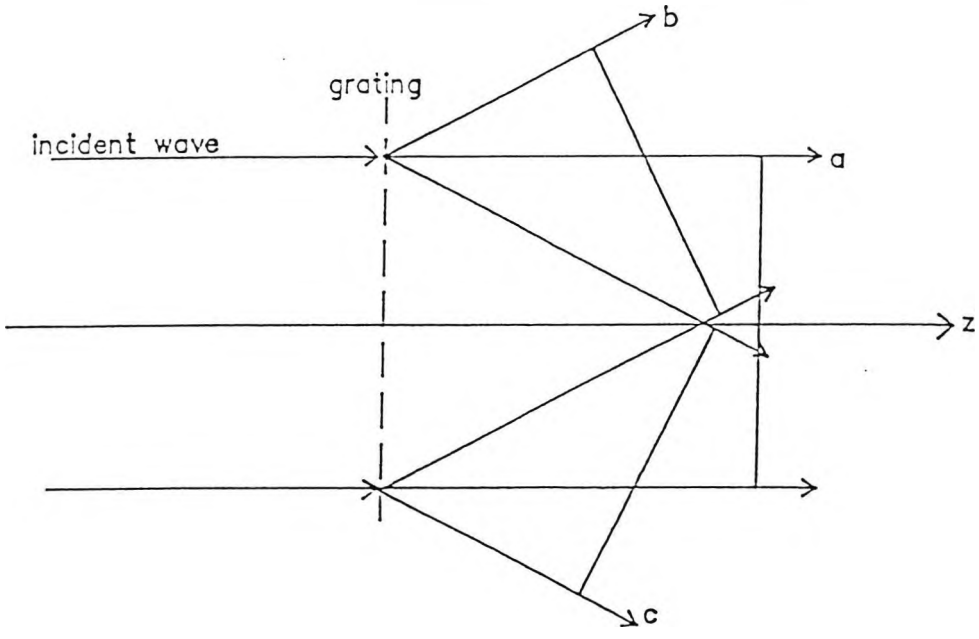


fig 2.19 Waves 'a', 'b' and 'c' are transmitted through grating G.

If the frequency varies across the grating, waves are diffracted in a variety of directions corresponding to the various frequencies. This is the situation in a hologram where a reference beam is diffracted in different directions to reconstruct the object wave. The method of holographic recording is shown in fig.2.20 where a photographic plate H (hologram) is illuminated by a coherent reference wave and an object wave from an object under study.

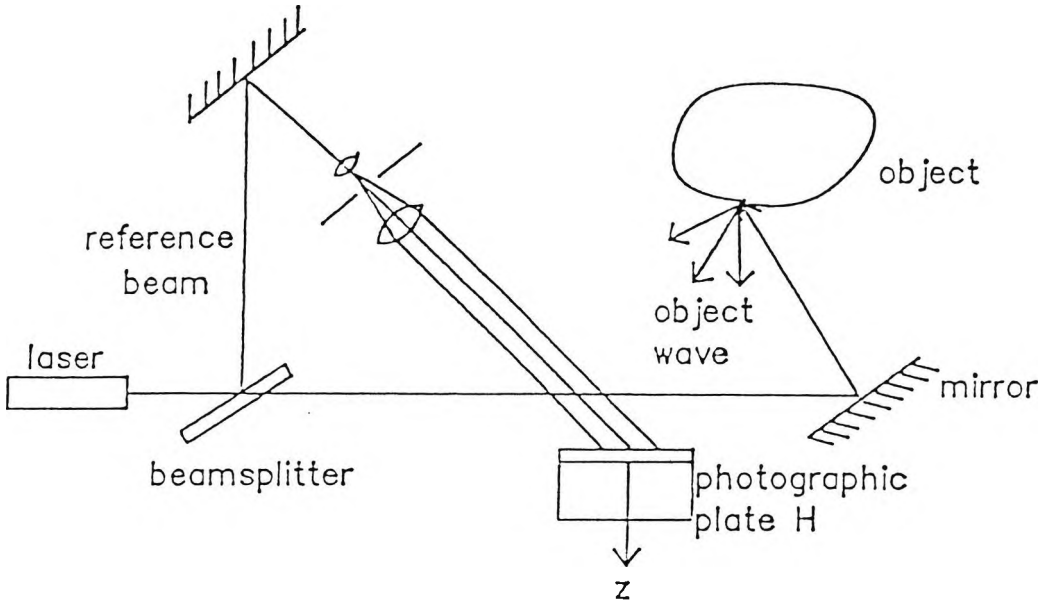


fig. 2.20. A photographic plate (hologram) is illuminated by a reference wave R and an object wave O.

The interference pattern of the reference and object waves are then recorded on the photographic plate H as shown in fig. 2.21

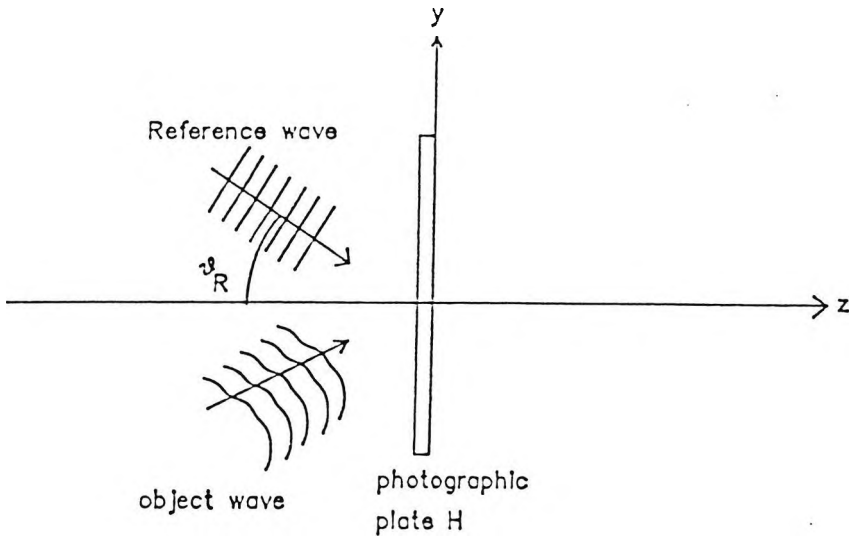


fig. 2.21 Photographic plate H at z=0 recording interference pattern between the reference and object waves.

At z=0 the object wave complex amplitude is

(2.60) $u_0(x,y) = a_0(x,y)e^{-i\Phi_0(x,y)}$

the reference is given by

(2.61) $u_R(x,y) = a_R e^{i2\pi f_y y}$

where $f_y = \dfrac{\sin\theta_R}{\lambda}$

The irradiance at the film plane z=0 is

(2.62) $I(x,y) = |u_0 + u_R|^2$ (film responds to $|u|^2$)

(2.63) $I(x,y) = \left(u_0 + a_R e^{i2\pi f_y y}\right)\left(u_0^* + a_R e^{-i2\pi f_y y}\right)$

(2.64) $= u_0 u_0^* + u_0 a_R e^{-i2\pi f_y y} + u_0^* a_R e^{i2\pi f_y y} + a_R^*$

=interference pattern on film

The transmission of the film is given by t where

(2.65) $t = t_b + \beta I$

t_b = constant (including a_R^2)

β = value depending on film

To reconstruct the object wave, the photographic plate H is illuminated by the reference wave u_R'

(2.66) $u_R' = a_R' e^{i2\pi f_y y}$

At z=0+, ie. just to the right of the hologram H

(2.67) $u_t = t u_R'$

therefore

(2.68) $u_t = \left(t_b + \beta |u_0|^2\right) a_R' e^{i2\pi f_y y} + \beta u_0 a_R e^{-i2\pi f_y y} a_R' e^{i2\pi f_y y} + \beta u_0^* a_R e^{i2\pi f_y y} a_R' e^{i2\pi f_y y}$

(2.69) $u_t = \left(t_b + \beta |u_0|^2\right) a_R' e^{i2\pi f_y y} + \beta u_0 a_R a_R' + \beta u_0^* a_R a_R' e^{i4\pi f_y y}$

The first term in equation (2.69) is a part of the reconstructed wave transmitted with attenuation and some modulation. The second term is a replica of the original object

wave and propagates as if it were from the original object. The third term is a conjugate of the original object wave.

When the hologram of an object has been taken, the object's 3-dimensional shape can be measured by making a second hologram of a reference object of known shape (eg. a sphere) on the same photographic plate. When the plate is illuminated with the reference beam, an interference pattern occurs between the two wavefronts of each hologram on the plate. Light fringes on the contour pattern correspond to a path difference of $2\pi N$ where N=integer. Dark fringes are contours of path difference πN.

For any distance Δx the change in optical phase is

$$(2.70)\ \frac{2\pi\Delta x}{\lambda}$$

For the Nth bright fringe therefore

$$(2.71)\ \frac{2\pi\Delta x}{\lambda} = 2\pi N$$

The distance Δx is then given by $N\lambda$, giving the variation in height between object and reference object at any bright fringe.

Chapter 3.

Shape models of the cornea.

3.1.. Descriptive models.

The minimum model representing the shape of the cornea has been presented by Ellerbrock (1961) and is shown in fig. 3.1. The sclera is considered to be circular with radius of curvature r1. The broken line in fig 3.1 is the scleral curvature in the region of the cornea. The cornea can be described as having a central circular optic cap of radius of curvature c1 surrounded by a peripheral annular zone which flattens toward the limbus. The limbus is the zone of transition from cornea to sclera. The corneal displacement from the circular contour of the sclera is called ectasia.

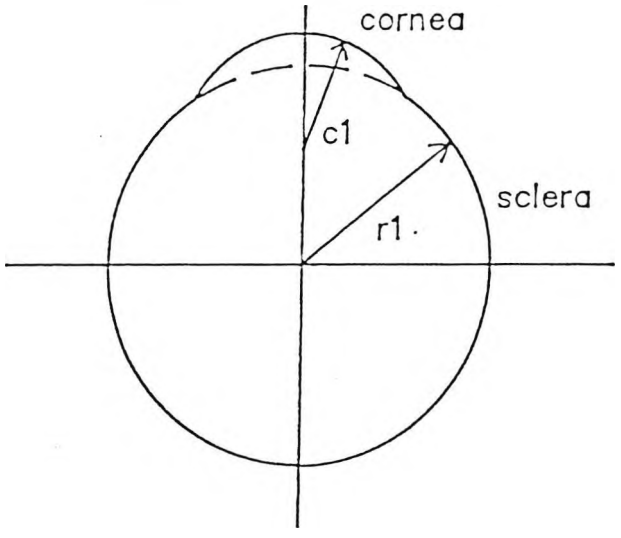


fig 3.1 Minimum model of the cornea. r1 is the radius of curvature of a circular sclera. The cornea has a central radius of curvature c1 and joins the sclera at the limbus.

Ellerbrock also made the following general observations -

a/. The corneal diameter is not constant between different corneas and in different meridians of the same cornea.

b/. The central curvature of the cornea varies between individuals and may be asymmetric or irregular.

c/. The cornea flattens towards the limbus with the amount of flattening varying between different corneas.

A more detailed description of the points and zones of interest on the cornea was presented by Sampson (1965)(cited by Clark ,1973b,) and is shown in fig. 3.2

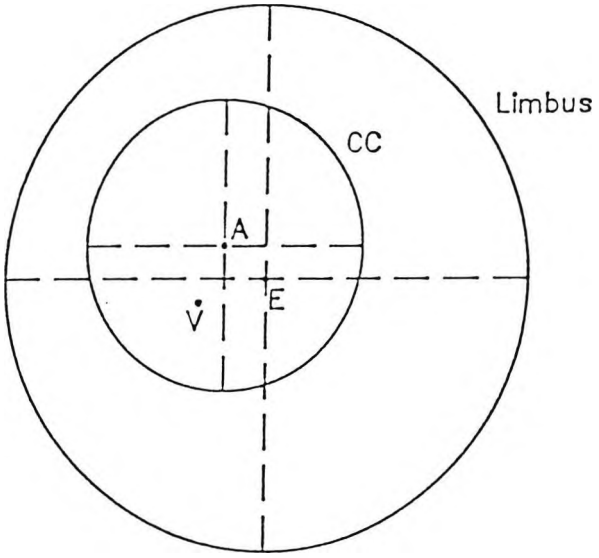


fig 3.2 Description of cornea showing its geometric centre E, circular optic cap CC, apex A (centre of cc) and visual centre V.

This model is an improvement on that of Ellerbrock in showing the potential non-alignment of the visual axis, corneal axis and apex position.

A more quantitative description of corneal shape was made by Knoll (1961) when defining 4 different corneal types based on 3 distinct zones on the cornea. The 3 corneal zones used were -

2. Central - 2mm in diameter.

2. Zone 1 - 4.5mm in diameter.

3. Zone 2 - 8.5mm in diameter.

Using these 3 zones, 4 different types of cornea were defined as follows -

Type A- Zone 1 symmetric; zone 2 flattening is less than 2.0mm of central zone value.

Type B- Zone 1 symmetric; zone 2 flattening is 2.0mm or more than central zone value.

Type C- Zone 1 asymmetric; zone 2 flattening is less than 2.0mm of central zone value.

Type D- Zone 1 asymmetric; zone 2 flattening is 2.0mm or more than central zone value.

From photokeratoscope pictures, Knoll was able to assign each cornea to one of the above types by measuring corneal radii at 5 points along the horizontal meridian.

3.2. Conic representation of the cornea

In an x-y Cartesian co-ordinate system the general equation of a conic section given by

(3.1) $Ax^2 + Bxy + Cy^2 + Dx + Ey + G = 0$

where A,B,C,D,E,G are coefficients to be determined for the particular ellipse.

This form of equation was used to describe the corneal shape by Townsley (1970) using data from images taken using the Wesley-Jessen PEK. The PEK images were analysed to find intersection points on the corneal surface with light rays from the PEK rings. For a given meridian a series of points on the corneal surface was produced and an ellipse fitted to them. A conic section described by equation (3.1) cannot be fitted

easily to a series of points using a 'least squares' method, therefore Townsley used an iterative method to evaluate the coefficients. The method of calculating the reflection points to construct the corneal profile is described in detail in chapter 2. Results from normal corneas showed a range of eccentricities for fitted ellipses from 0.4 - 0.9 with an average value of 0.55.

A conic section was also selected as an adequate description of the corneal profile by Mandell (1971), who fitted ellipses to photokeratoscope images. The results for normal corneas showed an eccentricity range of 0.2 - 0.85 with an average value of 0.48.

The Mandell method of fitting ellipses to the photokeratoscope images was different to the procedure used by Townsley. Instead of calculating reflection points along the corneal profile, the corneal reflecting surface was assumed to be an ellipse with unknown apical radius of curvature and eccentricity. For a given photokeratoscope object ring, theoretical image heights were calculated for elliptical reflecting surfaces of different apical radius of curvature and eccentricities. The basic equation of an ellipse in x-y co-ordinates, centred on a point (a,0) is given by

$$(3.2)\quad \frac{(x-a)^2}{a^2}+\frac{y^2}{b^2}=1$$

where 'a' is the semi-major axis and 'b' is the semi-minor axis.
This equation was used to generate the reflecting surface and only rays parallel to the x-axis were used to calculate the theoretical image heights as shown in fig 3.3.

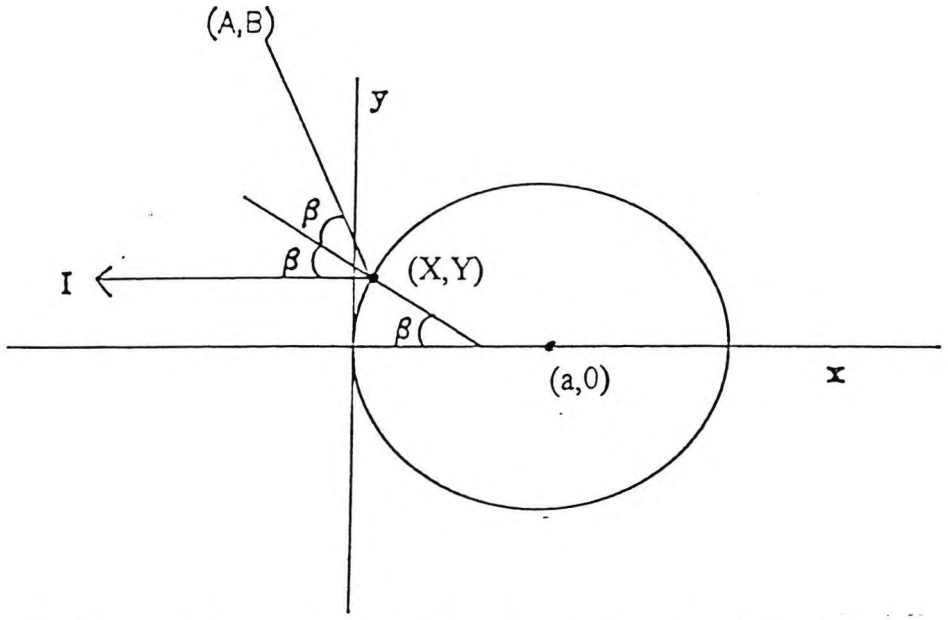


fig. 3.3 Shows is an ellipse centred at (a,0). Rays from an object ring at (A,B) strike the elliptical reflecting surface at (X,Y). The angle of incidence β is such that rays forming the image at I were reflected parallel to the x-axis.

A graph of different ellipse parameters (apex radius of curvature and eccentricity) generating a set of image heights is shown in fig 3.4

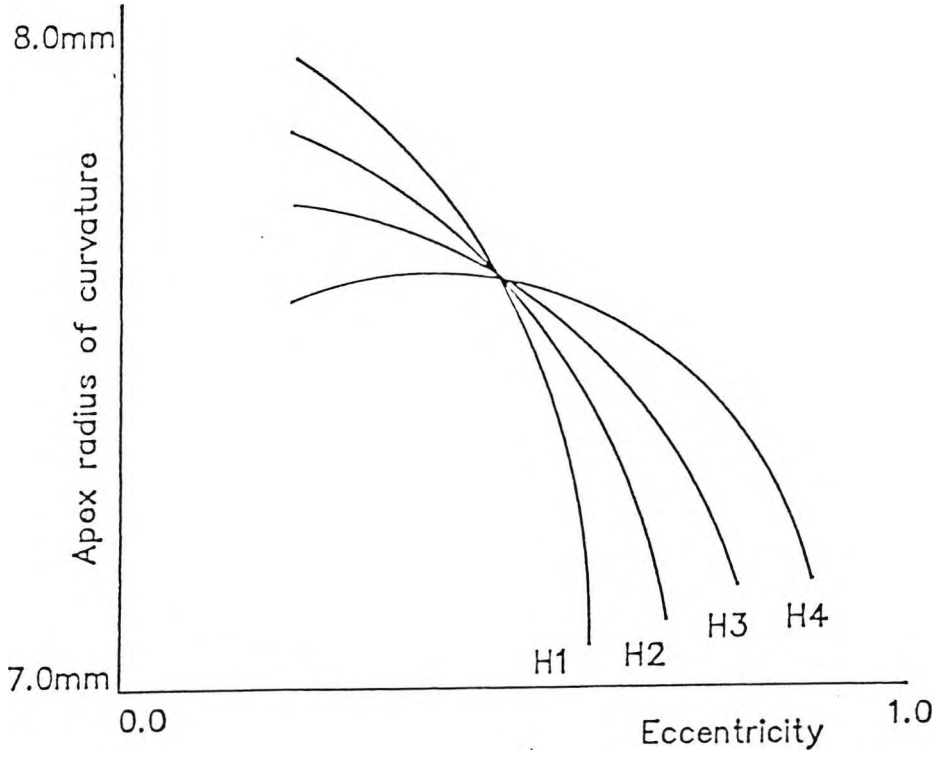


fig 3.4 For a given keratoscope ring, each curve is a locus of ellipses which will form an image of the ring at height H_n.

The ellipse given by the intersection point of the curves on the graph in fig 3.4 is the one ellipse satisfying the formation of all keratoscope ring images. The results from a typical cornea show a range of intersection points, the magnitude of which gives a measure of the 'goodness of fit' of an ellipse to the cornea.

Developing the idea of fitting a conic section to a corneal meridian, Kiely (1982) modelled the whole corneal surface using a conicoid. The basic measurements were obtained using the autocollimating photokeratoscope developed by Clark (1972), measuring along 4 equally-spaced meridians. A rotationally symmetric conicoid of the form

(3.3) $X^2 + Y^2 + (1+Q)Z^2 - 2ZR = 0$

was fitted to the meridian values using a least squares fit. Q in equation (3.3) specifies the type of conicoid as shown in fig 3.5 with R being the radius of curvature at Z=0. This equation is analogous to the Baker equation with P=1+Q.

Q is related to the eccentricity 'e' of the conicoid by

(3.4) $Q = -e^2$

and

(3.5) $e = \left(1 + \left(\frac{b}{a}\right)^2\right)^{\frac{1}{2}}$

where b=semiminor axis

a=semimajor axis.

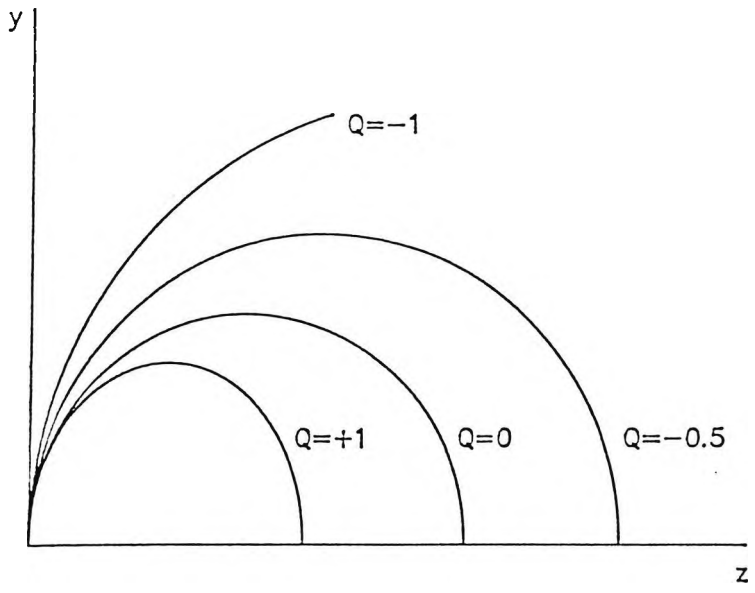


fig. 3.5 Conicoid cross-section described by equation (3.3).

The Q value specifies the type of conicoid where -

Q>0 ellipsoid with major axis in x-y plane.

Q=0 sphere

-1<Q<0 ellipsoid with major axis in z direction.

Q=-1 paraboloid with major axis along z axis.

Q<-1 hyperboloid

The results for normal corneas modelled by equation (3.3) show R=7.72±0.27mm and Q=-0.26±0.18. The absolute range of R was 7.06 to 8.64 and Q was -0.76 to +0.47.

In addition to the rotationally symmetric conicoid, a non-symmetric analysis was made by Kiely using equation (3.3), allowing R and Q to vary with angle θ from the horizontal. Q and R then take the form

(3.6) $Q(\theta) = Q_1 + Q_2 \cos^2(\theta - \alpha)$

(3.7) $R(\theta) = R_1 + R_2 \cos^2(\theta - \beta)$

α and β = angles containing either maximum or minimum values of Q and R assuming values are mutually orthogonal.

3.3. Equations of flattening

Equations expressing the peripheral flattening from a central spherical region have been applied to corneal meridians by Bonnet (1962) and Fujii (1972).

Bonnet applied a linear law of flattening to the cross-section of an axially symmetric cornea of the form

(3.8) $\log\beta = \kappa\gamma + b$

where b is the diameter of the central spherical zone and γ is the angle between the normal to the cornea at a point and the corneal axis (see fig. 3.6). κ is the coefficient of flattening. The deviation Δ of the cornea from the central zone sphere is shown in fig 3.6 and is given by

(3.9) $\Delta = -\frac{R_0\beta}{\kappa}$

where R_0 = radius of curvature of central zone of cornea.

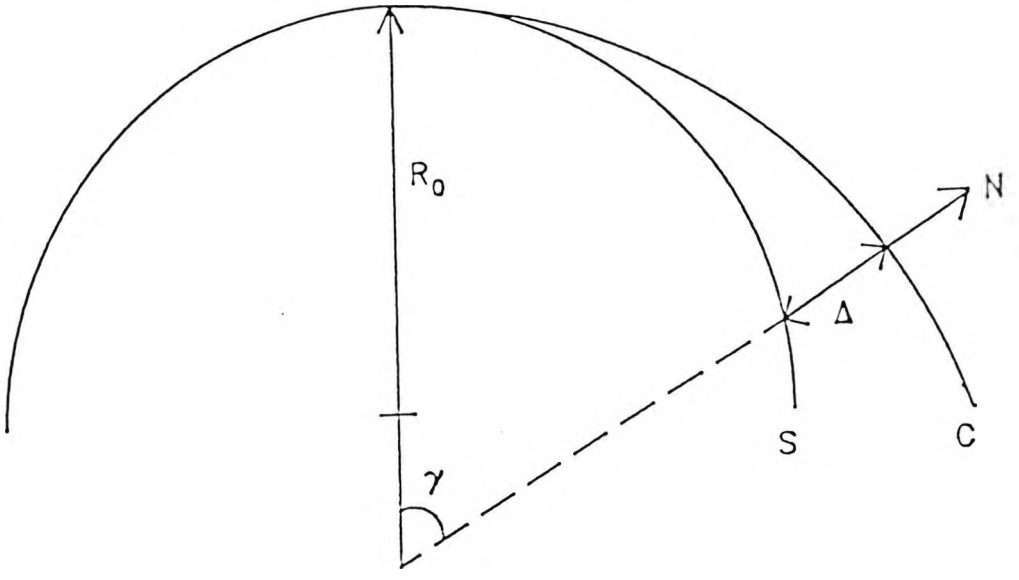


fig. 3.6 The cornea C has a central spherical zone described by sphere with radius of curvature R_0. Towards the periphery, the cornea deviates from the sphere by a distance Δ along the normal N to the cornea.

For each meridian R_0, κ and b is determined by measuring the radius of curvature at 3 different points across the cornea.

Fujii (1972) studied the cornea with a photokeratoscope and concluded that the corneal configuration could by represented by the formula

$$(3.10)\quad x = f(y) = \frac{1-\left(1-c^2y^2\right)^{\frac{1}{2}}}{c} + \begin{cases} A_1 y^{10} & (y \geq 0)(\text{temporal}) \\ A_2 y^{8} & (y < 0) \quad (\text{nasal}) \end{cases}$$

where x represents the corneal depth at a distance y from the central point of the cornea. The first term in (3.10) represents a sphere of curvature c and the second term represents the deviation of the corneal surface from this sphere. The coefficients c, A_1 and A_2 are determined for each cornea.

3.4 Corneal shape indices.

A quantitative method specifying corneal shape using shape indices was introduced by Cohen (1984) using results from a Corneascope. The Corneascope is a 9 ring photokeratoscope developed by Rowsey (1981) producing the usual images of concentric circles reflected from the cornea. Each of the outer 8 rings of the Corneascope image was analysed by Cohen, recording the maximum and minimum chord lengths for each ring. This is illustrated diagrammatically for 3 rings in fig 3.7

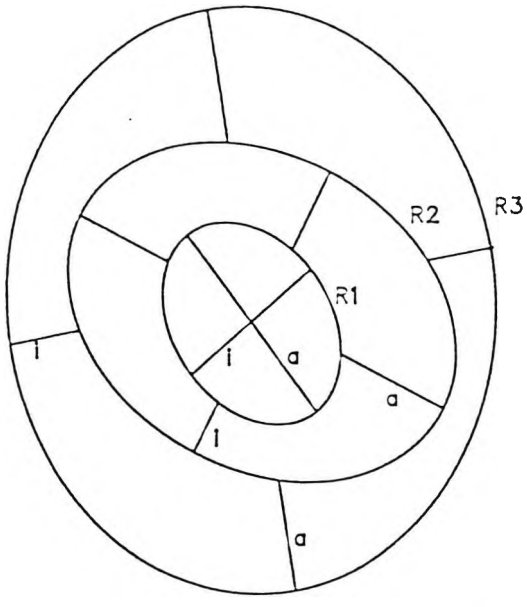


fig 3.7 Rings on Corneascope image showing maximum 'a' and minimum 'i' chords for 3 rings R1,R2,R3 reflected from the cornea.

For individual rings there are 4 indices:

1. Eccentricity (E) = minor chord length/ major chord length. For a circle E=1 and as a ring becomes elliptical, E→0. This is opposite to the usual ellipse eccentricity definition.

2. Angularity (A) measures the acute angle α between major and minor chords where $A = \alpha(\text{radians})/\pi/2$.

3. Major symmetry Sm_0. If e and f are major chord segments created by the intersection of the minor chord, then Sm_0=f/e where e is the longer chord section.

4. Minor symmetry Sm_a. This is analogous to Sm_0 but for the minor chord.

Combining ring values for one image produces 2 additional 'whole eye' indices -

5. Cluster index - which is further subdivided into

   a) angle cluster C_A = standard deviation of measurements of angle between major or minor chords and a reference line.

   b) distance cluster C_D describing the tendency of chords to pass through a common point. C_D = Standard deviation of the shortest distances from a specified point to each chord.

6. Trend index - which is also subdivided into 2 components

   a) eccentricity trend index measuring the change of eccentricity of the rings towards the periphery.

   b) angularity index measuring the orientation change of the major chord for consecutive rings from the centre to periphery.

This set of 6 indices for a given cornea are given the title Photogrammetric Index Method (PIM) by Cohen. PIM was then proposed as a method of distinguishing between 3 groups of corneas -

1. Symmetric corneas - defined as corneas with equal horizontal and vertical keratometric readings.

2. Regular astigmatic corneas - defined as corneas where the vertical K reading exceeds the horizontal by 0.5 Dioptres or more.

3. Keratoconic corneas - defined by various clinical criteria.

3.5 Corneal power maps

A feature of computerised photokeratoscopes is the colour coded display of power values on the cornea derived from ring image heights. This has led to a classification system devised by Bogan (1990) for the Corneal Modelling System (CMS) and using power contour patterns to differentiate between corneal types. The method analyses the basic contour pattern as shown in fig. 3.8.

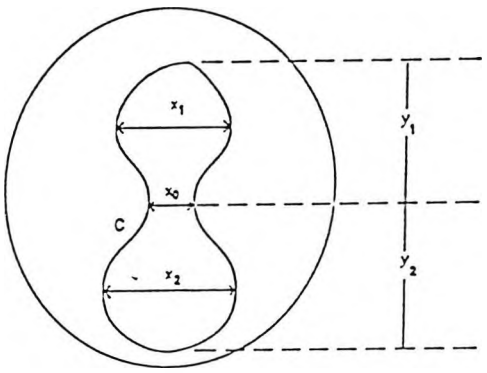


fig 3.8 The central contour pattern C on the corneal power map is divided into different regions as shown.

The basic pattern is used to define 5 classes of corneal shape as shown in fig. 3.9

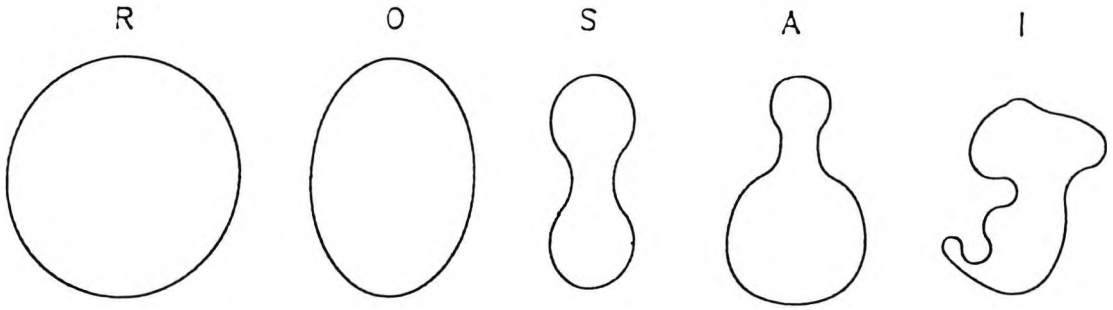


fig. 3.9. Contour patterns showing 5 different classes of corneal shape. R=round, O=oval, S=symmetric bowtie, A=asymmetric bowtie, I=irregular.

The shape classes are defined as follows (see fig 3.8)

Round - ratio of shortest to longest diameter 2/3 or more.

Oval - ratio of shortest to longest diameter less than 2/3

Symmetric bowtie - (a) central constriction exists.

(b) $\frac{x_0}{x_1}$ or $\frac{x_0}{x_2}$ is $\frac{1}{3}$ or less

(c) $\frac{x_1}{x_2}$ or $\frac{y_1}{y_2}$ are $\frac{2}{3}$ or more

Asymmetric bowtie - criteria (a) and (b) for symmetric bowtie are satisfied and

$\frac{x_1}{x_2}$ or $\frac{y_1}{y_2}$ are less than $\frac{2}{3}$

Irregular - no clear basic contour pattern could be identified.

Using this classification system, Bogan (1990) studied the distribution of normal corneas.

The results are given below.

Pattern	% of corneas
Round	22.6
Oval	20.8
Symmetric bowtie	17.5
Asymmetric bowtie	32.1
Irregular	7.1

In practice, power maps taken of a single cornea show a variety of patterns, with the shape most closely resembling the appearance of the cornea being chosen by the operator as the correct result. These variations are probably due to non-alignment of the corneal apex with the instrument axis. For most corneas, which flatten towards the periphery, a non-alignment will produce an asymmetric power distribution but is not indicative of an astigmatism. The results found by Bogan (1990) are more likely to indicate a variation in corneal alignment than a true classification of corneal types.

Chapter 4

Topography Measurement System Description

4.1 Introduction.

Different methods of measuring the topography and curvature of the anterior corneal surface have been developed over the last 50 years and have been reviewed in chapter 2. Although each method has its own unique features, two fundamentally different approaches have emerged in modelling the corneal surface.

The first approach uses the cornea as a reflecting convex mirror and measures the height of concentric ring images formed from the corneal surface. Standard paraxial ray equations are used to calculate the curvature at different points and build up a profile. Photokeratoscopes based on this principle (Townsley, 1967;Bibby 1976) have been widely used over the last 20 years and have lately been superseded by more sophisticated instruments (Gormley , 1988; McCarey, 1992) which photograph and analyse the ring images using a microcomputer system.

The second approach to mapping the cornea has been to measure actual points on the surface of the cornea and build a 3-dimensional map. Methods using this approach, which look directly at the corneal surface, have included profile measurements (McMonnies, 1971), moiré fringe analysis (Kawara, 1979) and stereophotogrammetry (Bertotto, 1948). Within the last few years, the stereophotogrammetry method has been developed into Rasterstereography (Warnicki , 1988; Arffa, 1989) using a modified slit lamp and computer image grabbing system.

Both approaches work well when the cornea under study is regular and near spherical. If the cornea is irregular as in Keratoconus for example, each

measurement system suffers from its own particular problems.

Keratoconus has been extensively studied and photographed using photokeratoscopes (Rowsey, 1981; Rabinowitz, 1990) but pictures typically show highly distorted images produced from non-spherical surfaces. Many of the rings are out of focus and there is additional uncertainty in the alignment of the cornea with the instrument axis. The resulting pictures are difficult to analyse and curvature values are probably invalid in these cases because they are derived from ray equations which incorrectly assume spherical surfaces. Stereographic techniques do not assume spherical surfaces, therefore they do not suffer from the problems outlined above when viewing distorted corneas. However, previous systems have calculated curvature by fitting the best arc to the corneal profile along any meridian. When the shape is far from spherical this method of measuring curvature is difficult and unreliable.

To measure the corneal topography and curvature of both normal and highly distorted eyes, the problems encountered by other instruments must be overcome. To achieve this it was necessary to develop a system which could -

1. Measure the corneal surface without any assumption about underlying shape.
2. Reconstruct the surface shape independent of alignment of the eye.
3. Extract parameters at any point on the surface and determine apex position and rate of flattening.

In addition to the above developments, specific system requirements include -

a. Measurement of corneal shape within the central 4mm. This is the area of particular interest with respect to vision.

b. Identification of apex position and measurement of apex curvature value.

c. Adequate depth-of-field of projection and camera system to measure irregular corneas.

d. Safety of the cornea during measurements.

e. Simple and accurate system calibration method.

f. Simple implementation of shape analysis method.

g. Analysis of surface to extract meaningful and concise parameters to describe the corneal surface.

h. Results output in a comprehensive and understandable form.

4.2. Non-mathematical description of the system developed for mapping and analysing the corneal shape (Corneal Topography System CTS).

4.2.1. Extraction of points on the surface of the cornea.

Points lying on the corneal surface in x,y,z space are extracted using a light plane and camera configuration as shown in fig 4.1.

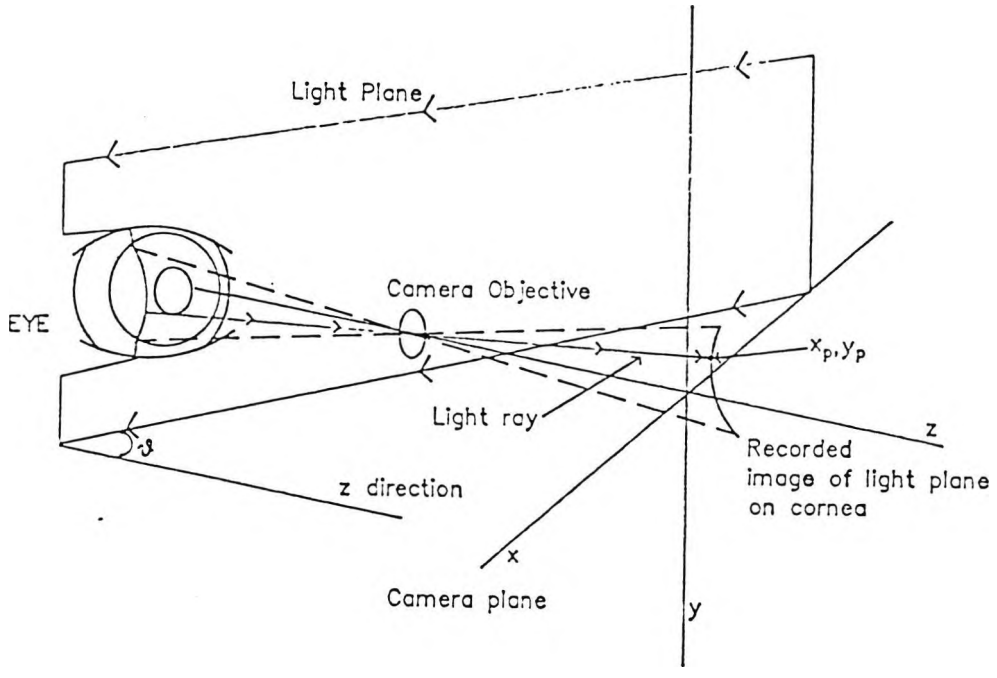


fig. 4.1. To extract points on the corneal surface in x,y,z space, a plane of light is projected onto the corneal surface. The image of the light plane is recorded by the camera system.

The light plane is projected onto the corneal surface where it is diffusely reflected and the image of the light plane on the cornea is recorded by a camera system. An arbitrary x,y,z co-ordinate system is defined coincident with the camera optical axes. The direction of the light plane in x,y,z is completely defined by measuring the angle ϑ and the distance the light plane cuts the z axis. Any point lying on the light plane image taken by the camera (e.g. at x_p,y_p) also lies on a ray from the corneal surface passing through the camera objective. When x_p and y_p are measured, the direction of the ray is known in x,y,z space. The intersection point of the ray and plane is unique and is calculated using the ray and plane equations in x,y,z space, giving a point which lies on the corneal surface.

4.2.2. Mapping the corneal surface.

To extract points over a large area of the corneal surface, 26 planes of light are projected onto the cornea from opposite sides as shown in fig. 4.2. Points on the cornea are then extracted from each plane using the method outlined above.

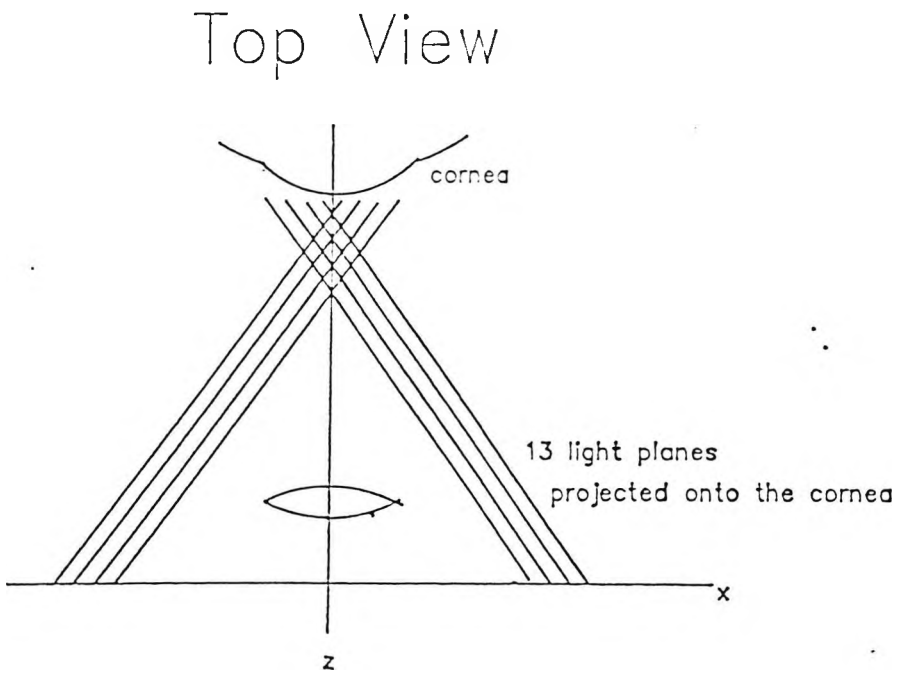


fig. 4.2. The central region of the cornea is covered by 2 sets of 13 light planes, projected from opposite sides.

The configuration yields maximum information over the central 4mm of the cornea which is the area of particular interest for vision. The depth resolution depends on the angle of the light planes with respect to the z axis, with a larger angle giving a higher resolution. If only one set of light planes is incident on the cornea, as is the case with the PAR system, the planes will appear close together on the side of the cornea towards the projection system, becoming wider as the corneal surface slants away on the opposite side to the projector. An example showing x,y,z co-ordinates at various points on the cornea is shown in fig 4.3.

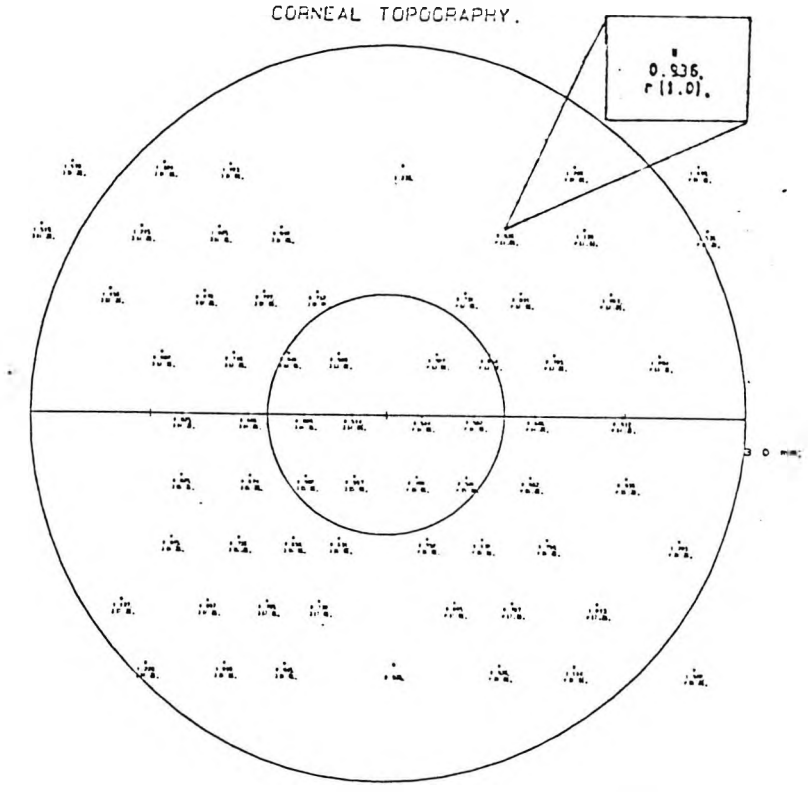


fig. 4.3. A set of extracted points on the cornea showing the distance of each point along the z axis and a point identification code (see inset).

4.2.3. Surface fit for calculation of curvature.

The method described above is able to map the surface irrespective of surface shape and irregularities. However, if the surface shape curvature is to be calculated, the curvature of interest is that of the general shape and avoiding any small localised

irregularities of high curvature. This is shown in fig 4.4. where the solid line is the actual profile and the dashed line represent the profile of a fitted surface for calculating curvature.

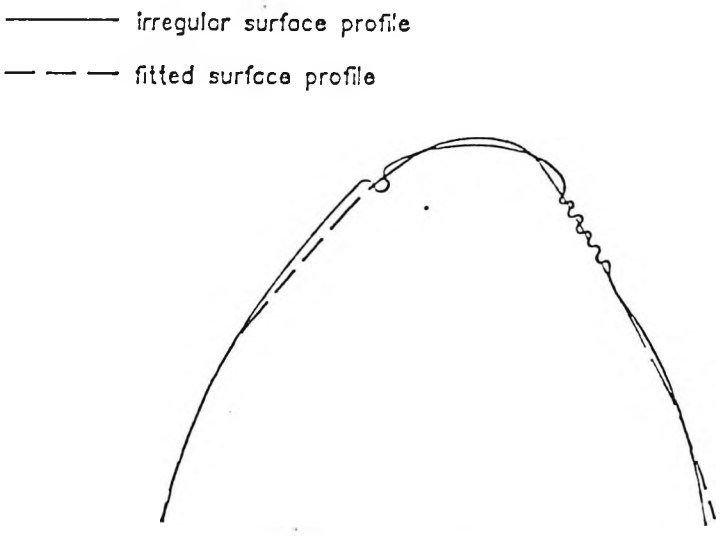


fig 4.4. For surface curvature calculations, a mathematical surface is fitted to the points which follows the general shape of the surface.

If it were not done in this way, a small irregularity which has a much larger curvature then the general corneal shape would be identified by the computer as the position of the corneal apex. A surface is therefore fitted to the points as shown in fig. 4.5. which smoothes over small irregularities and filters out system noise but is adjusted to pass within a pre-determined distance of the points. This distance is chosen by the operator and is usually of the order of the system resolution (10 microns).

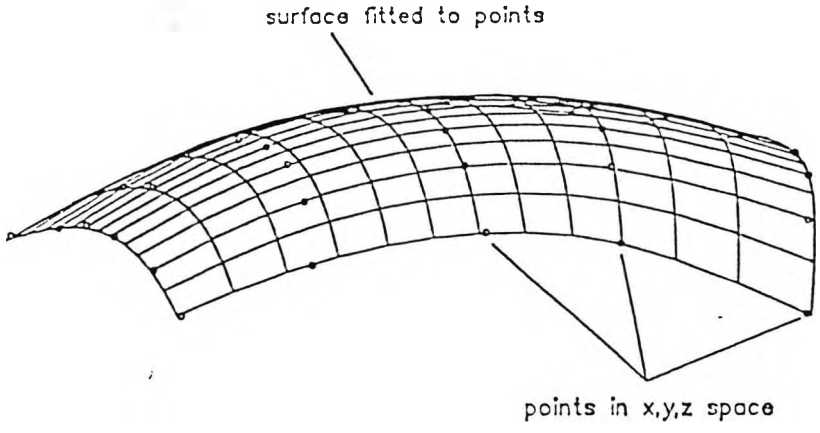


fig. 4.5. Fitted surface following points but smoothing noise and irregularities.

4.2.4. Calculation of surface curvature.

To calculate the curvature at any point on the surface, the direction of the centre of curvature must first be established. This is easily done by finding the plane which is tangental to the surface at the required point. This is shown in fig. 4.6.

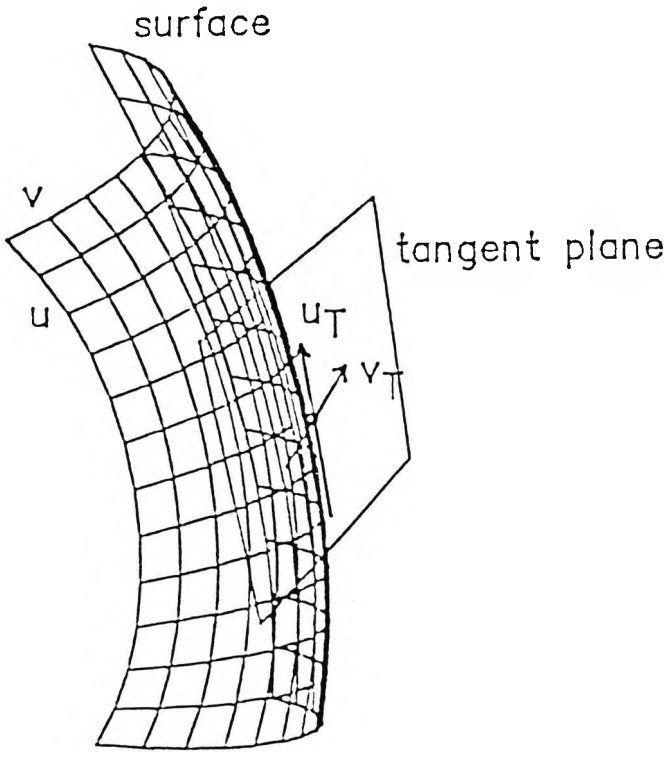


fig. 4.6. Tangent plane is calculated at any point on the surface.

This plane contains the two tangent vectors to the surface co-ordinate axes. By differentiating the surface equation along each axis the tangents are derived thus defining the tangent plane. When the tangent plane is known, the normal can be found and the centre of curvature of the surface at this point lies along the normal direction. This is shown in fig. 4.7.

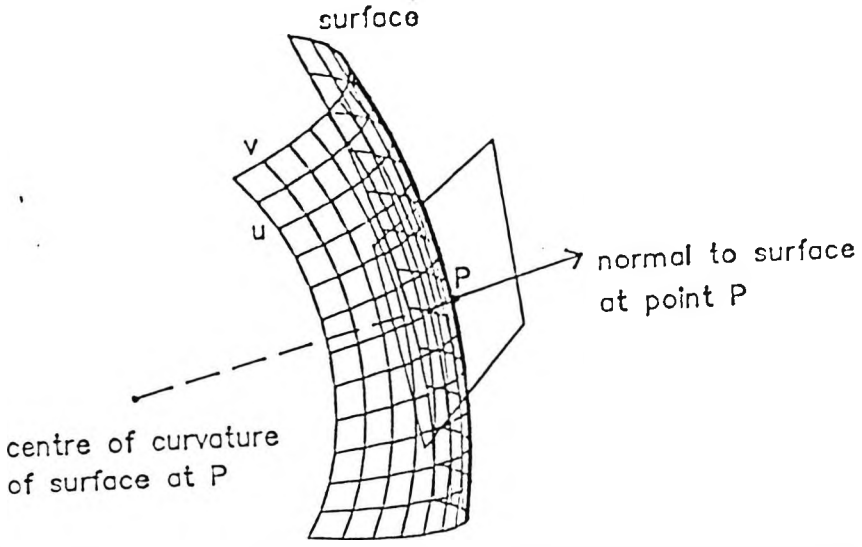


fig 4.7. The centre of curvature of the surface at P lies along the direction of the surface normal at that point.

To calculate the curvature at P, a plane containing the normal is constructed in a given direction with respect to the surface co-ordinate axes. The intersection of this plane with the surface defines a line whose curvature gives the curvature of the surface at that point. As the plane is rotated using the normal as the axis of rotation, different lines in the surface are defined giving different curvature values. This is shown in fig. 4.8.

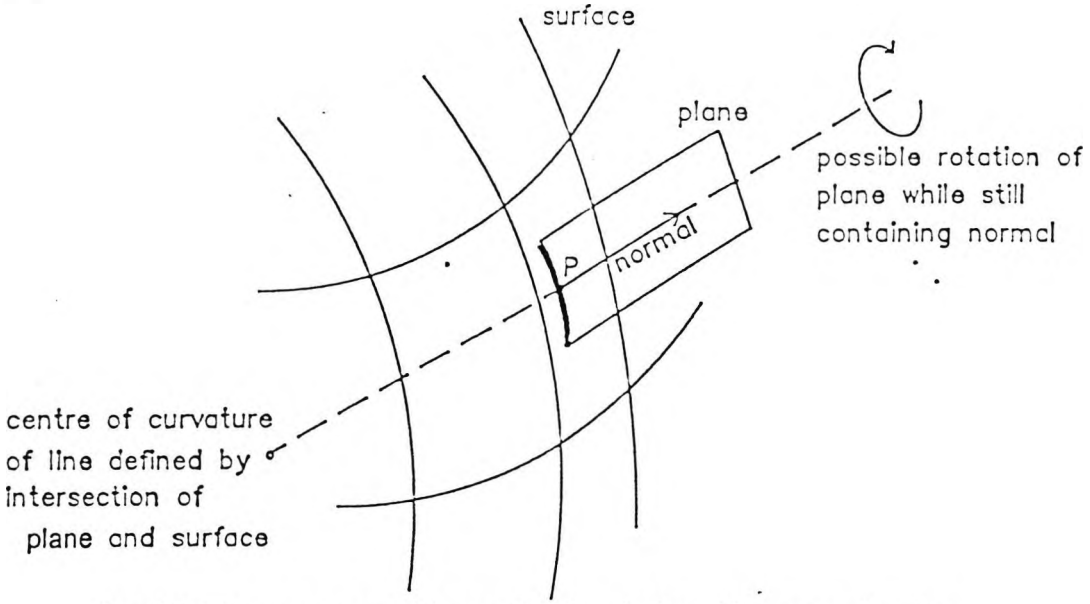


fig. 4.8. The curvature of the surface is calculated by measuring the curvature of a line defined by the intersection of the surface with a plane containing the normal to the surface.

Gauss showed that as the plane is rotated, positions of maximum and minimum curvature are found and are orthogonal.

In the calculation of curvature for the cornea, the corneal surface within a very small area of 0.05mm radius has been taken as spherical. Radius of curvature values within these small areas have been calculated for 100 different points within the central 4mm of the surface, to show the variation of curvature (astigmatism) over the cornea. Small area averaging has been used to eliminate noise and an average value of the flattening from the apex is calculated together with the apex position and radius of curvature. This gives 3 parameters classifying the surface shape (Edmund ,1987) and allowing concise comparisons between corneas.

4.3. System design.

The system consisted of a computerised imaging system and two custom made light plane projection systems. A block diagram showing all the system elements is shown in fig. 4.9.

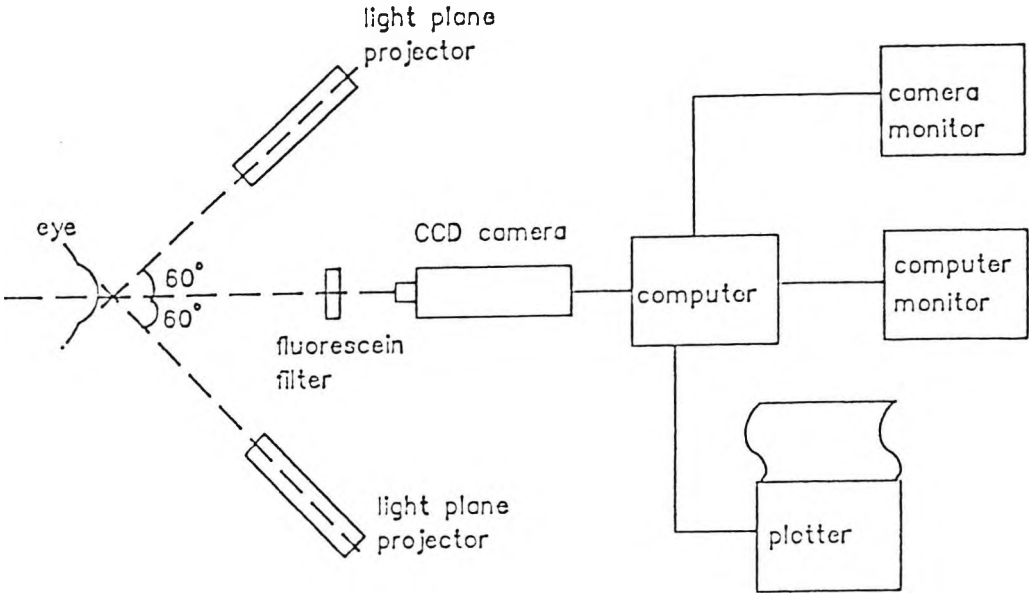


fig. 4.9. System block diagram

4.3.1. Light plane projector description.

Two projection systems were built to project 26 light planes over a 6mm width of the cornea. The 2 projectors were aligned at approx 60° on either side of the visual axis so that the projected planes covered opposite sides of the cornea. The projector was constructed with a 150 watt tungsten projector bulb behind a condensing lens and stop to give even illumination on a rectangular square wave grating of 12 lines/mm. Between the grating and the condensing lens, an I.R. blocking filter was positioned to protect the grating from heat. A 1mm wide vertical rectangular aperture positioned in front of the grating allows projection of 13 planes of light. A second I.R. blocking filter was positioned in front of the grating to give further protection to the cornea. An objective lens of 10cm focal length was positioned 22.5 cm from the grating. Behind the objective, an aperture stop of 4mm diameter was placed such that the image of the light source filament was in focus at the stop. This allowed maximum light through the aperture to

the objective. The image of the grating is then focused on the cornea at approximately 15cm from the objective lens. A diagram of the projection system is shown in fig. 4.10.

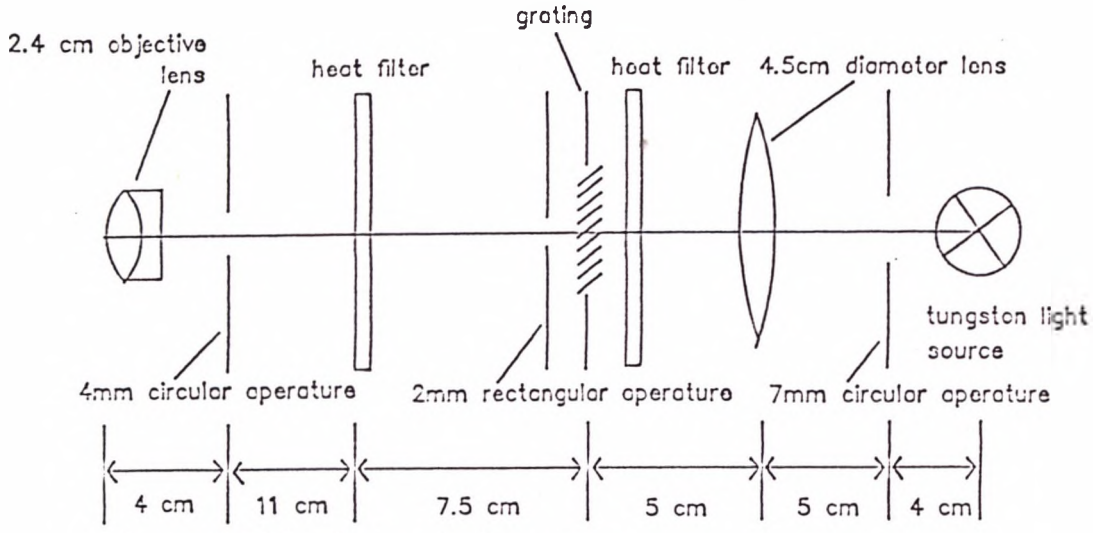


fig. 4.10. Light plane projection system

The position of the projection system and camera for corneal measurement is shown in fig 4.11.

fig. 4.11. showing the projection systems, CCD camera and position of subject.

4.4. Computer imaging system details.

The computer imaging system was based around the Imaging Technology Overlay Frame Grabber of 768 x 512 pixels. Each pixel was represented by 12 bits with the lower 8 bits holding a black and white image with a possible 256 different grey levels. The higher 4 bits hold text information which can be placed over the screen image without damaging the screen information. The frame grabber consisted of a single board and was fitted in an I.B.M. compatible PC with 4mb RAM and 100mb hard disk, running Microsoft Windows. All the software for the project was written in Microsoft C v6.0 and used the Microsoft 'Software Development Kit' (SDK) to run in Microsoft Windows. All the source code for the topography system is given in appendix A. The camera used was an iVC 800 BC CCD camera connected to the frame grabber through an analog to digital converter (ADC). The ADC could be programmed by the user to give real time enhancement of the camera image.

4.5. Real time image enhancement.

The faint image of the light planes on the cornea required an image enhancement to be programmed to the camera input, enabling the images to be clearly seen in real time. The enhancement used was a histogram equalisation method (Gonzalez, 1987) which measures existing grey levels in an image and remaps the levels, changing the relative brightness of features in the image. Large areas in the image which have a small difference in contrast are remapped to give a large variation in contrast. Areas which already have a large variation in contrast are left unchanged. The correct levels for remapping are first calculated for an average picture taken without enhancement and are then programmed to the ADC enhancing all subsequent levels. The resultant image of the cornea seen by the operator is shown in picture fig. 4.12. This picture shows the cornea (black) with the 2 sets of light planes imaged on the surface (curved stripy

appearance). The image of the light bands on the iris can be seen at either side of the fainter corneal bands.

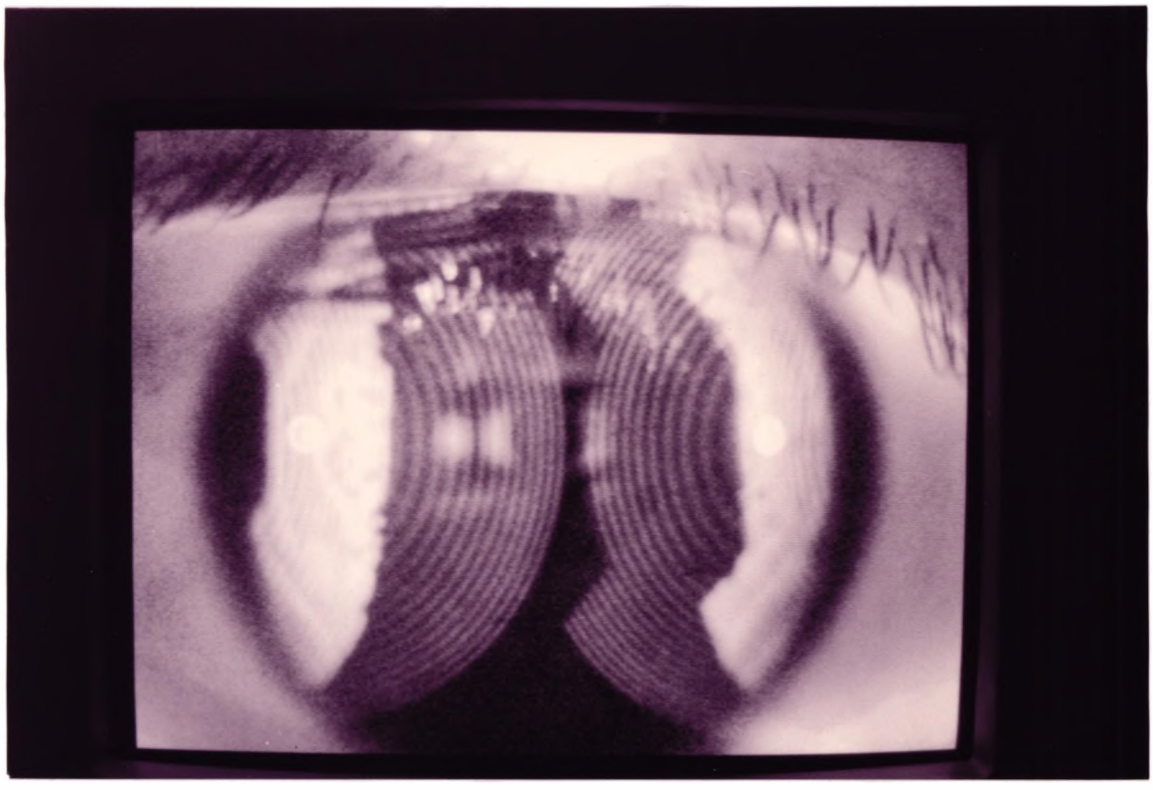


fig. 4.12. Real time corneal image showing projected light band pattern.

4.6. System calibration

The requirement of the system calibration is measurement of the angle and intersection distance along the camera axis for each light plane. To facilitate calibration, a simple method was designed using a flat white screen placed on a linear stage micrometer of 1 micron resolution. The linear stage is attached to a lightweight aluminium mounting that slots into a holder on the front of the camera system mounting. When switched on, the projected light planes fall on the flat screen and are imaged by the camera system. The configuration is shown in fig 4.13.

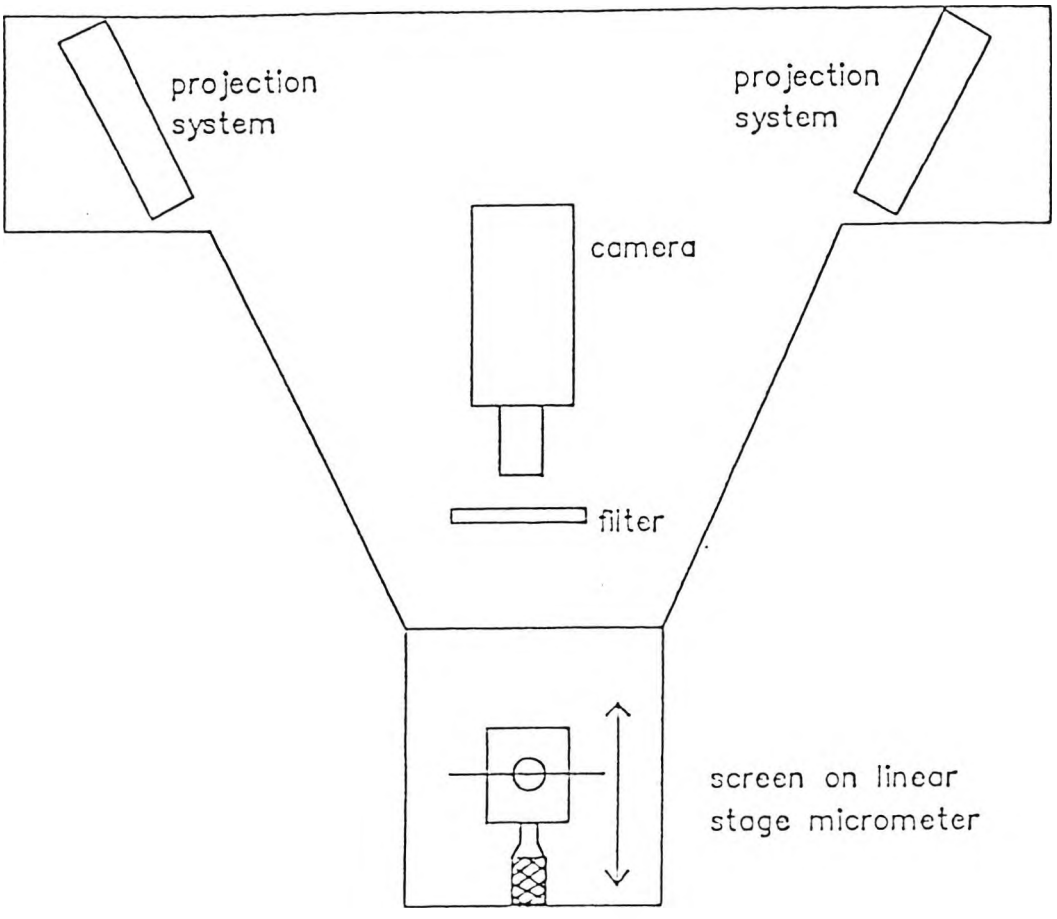


fig. 4.13. Mechanical configuration for calibration. Top view.

To correctly align the screen perpendicular to the camera axis, vertical lines are drawn by software at the centre of the camera screen and at equal distances either side of the centre. The camera and screen are adjusted such that linear motion of the screen brings the sets of planes simultaneously to either the centre line or the side lines and are vertical. The appearance on the camera screen for both positions is shown diagrammatically in fig 4.14.

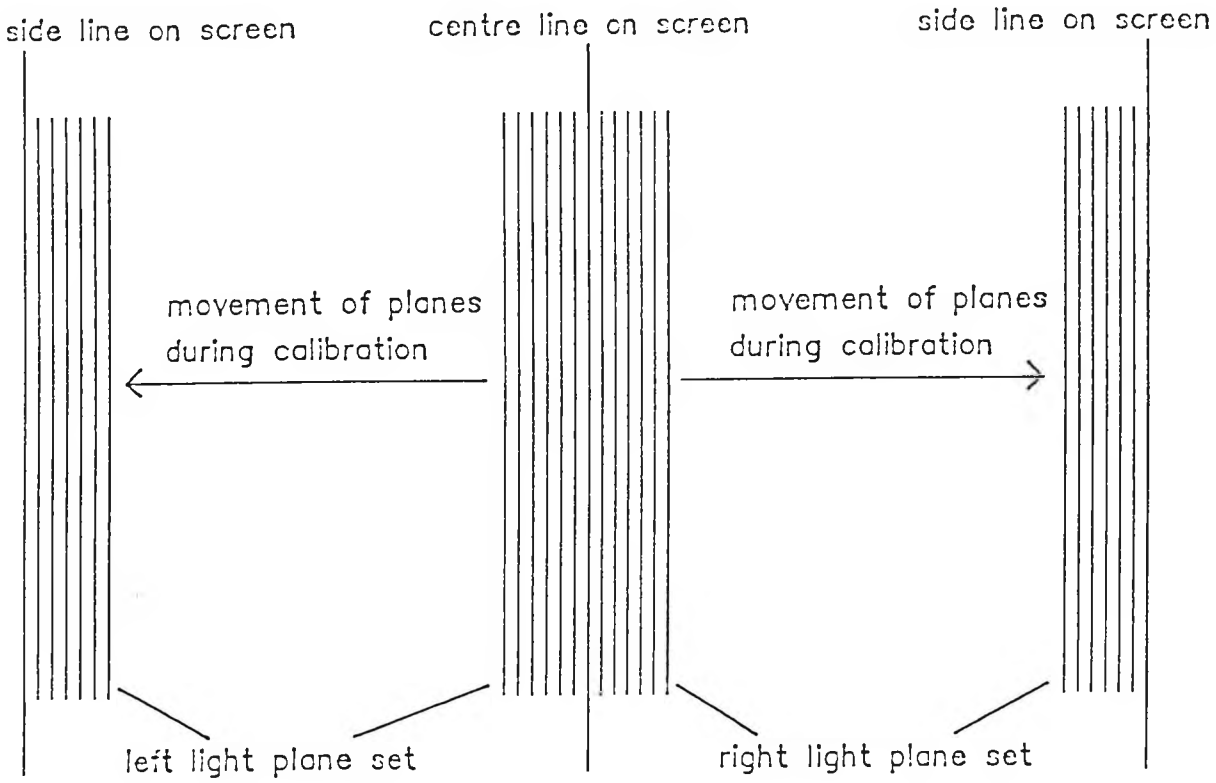


fig 4.14. Appearance of camera screen showing both positions for light plane calibration

For calibration, the screen is moved such that the light planes are positioned at the central line as shown in fig. 4.14. The position of each plane on the screen is identified and recorded by the computer and the micrometer reading is recorded by the operator. The screen is moved to position the light planes at the side lines and the plane positions are recorded by computer. The new micrometer reading is noted by the operator and the linear distance moved between screen positions is input to the computer. For a fixed camera magnification, the computer calculates the angle for each plane and the separation between planes along the z axis. This is shown in fig. 4.15

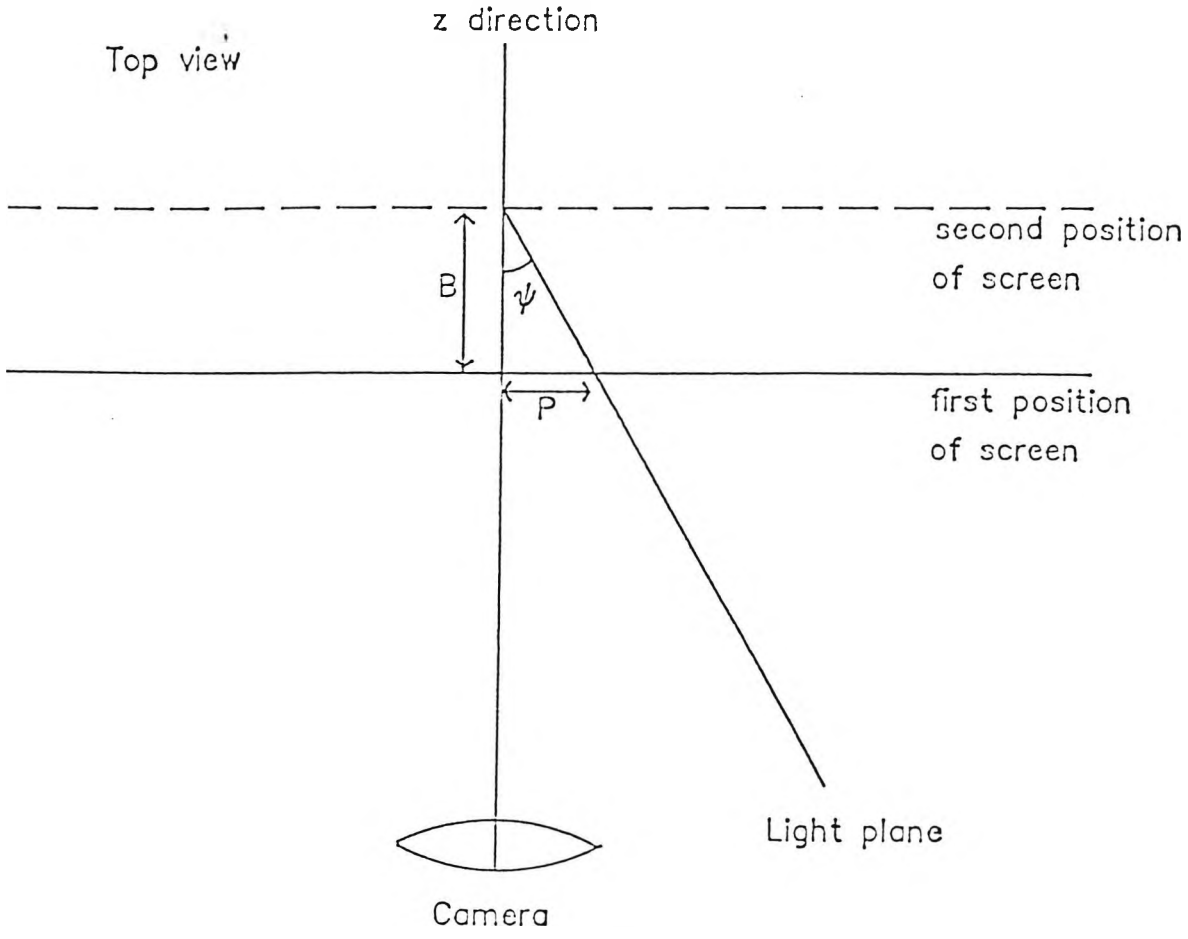


fig. 4.15 A given light plane image moves a distance P when the screen is moved a distance B.

If angle ϑ is the angle between the light plane and the z-axis then

(4.1) $\tan\vartheta = P/B$

where P is the horizontal distance moved by a plane when the screen is moved a distance B. With the angle ϑ known for each plane, the separation d between planes along the z axis is easily calculated as shown in fig. 4.16

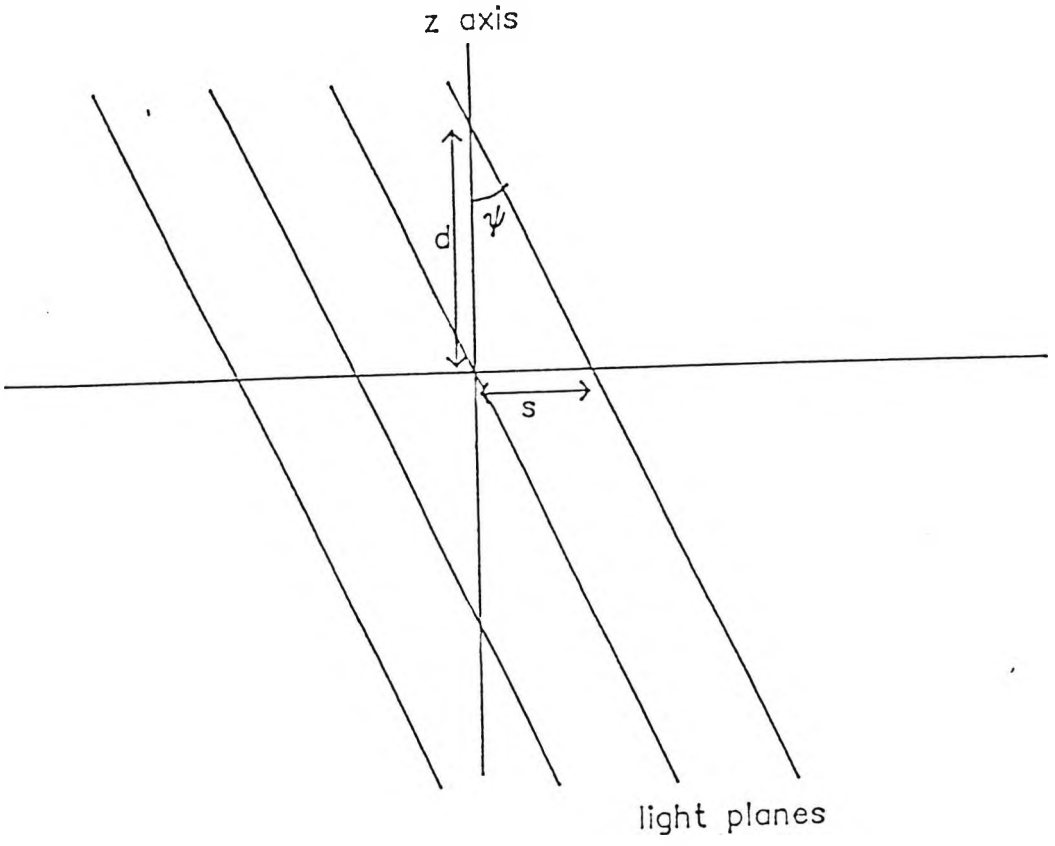


fig. 4.16 Calculation of light plane separation d along the z axis. s is the horizontal separation of the planes.

The horizontal separation s of the light planes is known from the calibration data, therefore d is given by

(4.2) $d = s/\tan\vartheta$

The intersection distance along the z-axis of the closest light plane to the camera of each set is given a fixed distance value = 150mm and all other plane distance values are known relative to this value. An incorrect estimation of the closest plane distance will lead to a linear shift in all the measurements but has no effect on the relative point positions, thus not affecting surface shape or curvature measurements.

4.7. Procedure for obtaining an image.

When the camera input ADC has been programmed for enhancement and the equipment calibrated, an image of the light planes on the cornea can be taken. To achieve a sharp image, 2% sodium fluorescein was instilled into the tear layer of the eye. The subject was then asked to place their head on the head rest and look directly into the camera objective, fixating on a reflected image of their eye in the camera objective. The camera and projection system were mounted on a smooth action horizontal and vertical stage and were moved until the two central light planes on the cornea were just touching. When this occurs they will also be at the centre of the camera plane and in focus. To assist this process, a vertical line was drawn at the centre of the monitor screen showing the required position of the two light planes. When the correct position was achieved, the image was grabbed and stored in the computer.

The next step is extraction of points on the cornea in x,y,z space from the image of the light planes. This step required 3 separate operations -

1. Identification of the geometric centre of the cornea to use as a reference point. This was achived by placing 2 vertical lines at the horizontal extremities of the corneal image (limbus). A horizontal line was moved down the screen to the position where the cornea just touches the vertical lines. The centre of the horizontal line was then taken as the geometric centre of the cornea. This is shown in fig 4.17. The direction from the centre of the camera axes to the corneal centre represents a best estimate for the visual axis direction.

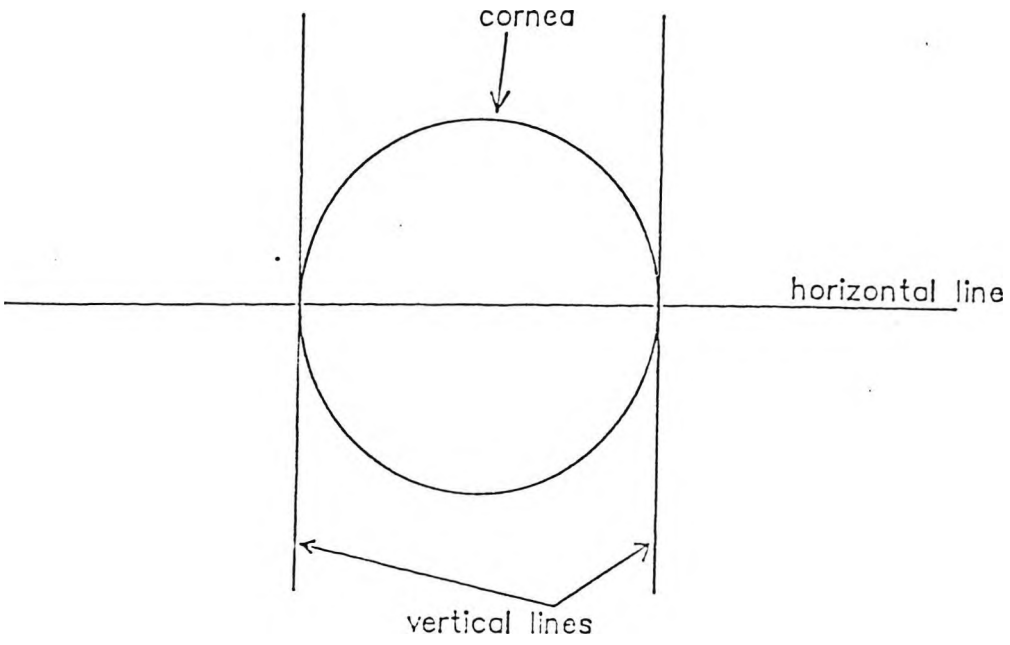


fig. 4.17 Identification of the centre of the cornea.

2. A vertical limit was set between which points were to be extracted. The criteria was that the light plane image bands must be in focus and well defined on the cornea.
3. A set of 9 equally spaced vertical positions were calculated on the image between the limits set in step 2. At these positions a cursor was moved using the mouse to identify the edge of each light plane. At these points the equations given in chapter 5 were solved to give discrete points on the cornea with x,y,z co-ordinate system values. 9 vertical positions were chosen as an optimum to give an adequate number of points on the cornea and also acceptable processing time for each cornea.

4.8. Implementation of analysis method.

To implement the surface equations described in chapter 5 section 2, the surface points must be moved to produce a set of points on a uniform matrix (equally spaced along a given co-ordinate axis) as is required by the surface theory. To

achieve this, 2 points were established on the central vertical line by horizontal interpolation of the corner points. A cubic interpolation scheme described in chapter 5 (eqn 5.26) was applied to all other points to construct an approximately uniform matrix. The points on this matrix were used to generate a surface equation for analysis of curvature over the area of interest. The equation is adjusted to smooth noise and irregularities while keeping within a set distance (error) of all the matrix points. For this project the error was set to the system resolution which was approximately 10 microns. The surface equation was analysed using the theory given in chapter 5 section 3. Radius of curvature was calculated at 100 different points over the surface. The apex position was identified and the apex radius of curvature and average flattening from the apex were calculated to give 3 parameters for concise description of the corneal shape. The description scheme based on these parameters was first devised by Carsten Edmund (1987) for clinical studies of corneal shape using the Wesley-Jessen PEK. Results printout of the corneal shape are in the form of a contour map with contours at depth intervals chosen by the operator (default = 50 microns). Direction and location of surface irregularities and astigmatism are indicated by the lines of contour and variations of the radius of curvature values. On the contour map, a letter 'A' indicates the position of the apex. Alongside the contour map are printed the apex radius of curvature, the average flattening from the apex (given the term Apex Factor A.F. by Edmund (1987)) and the apex distance from the centre of the cornea. The radius of curvature and flattening values are given in both mm and dioptres where the dioptre value is calculated from the radius of curvature value using

(4.3) $P = (n-1)/r$

where P = power in dioptres

r = radius of curvature in mm

n = refractive index taken as 1.3375 (the value most commonly used with conventional keratometers e.g. CMS)

The values in dioptres are given because of their widespread use in the literature, however their value is questionable in cases where the cornea is irregular, as for example in Keratoconus.

Chapter 5.

Corneal Topography and Curvature Measurement Theory.

This chapter expands the mathematical details of the measurement system described in chapter 4.

5.1 Method

5.1.1. Measurement of surface without previous assumption of surface shape.

A system was constructed to project vertical planes of light onto the eye; then a picture of the planes on the cornea was taken by a computer image grabber. The direction of each plane is known and used in place of a second camera in a stereographic system for calculating depth information (Sato, 1982). The projection system was arranged to give maximum information over the central 4mm of the cornea, which is the area of particular interest for vision. The configuration is shown in fig. 5.1.

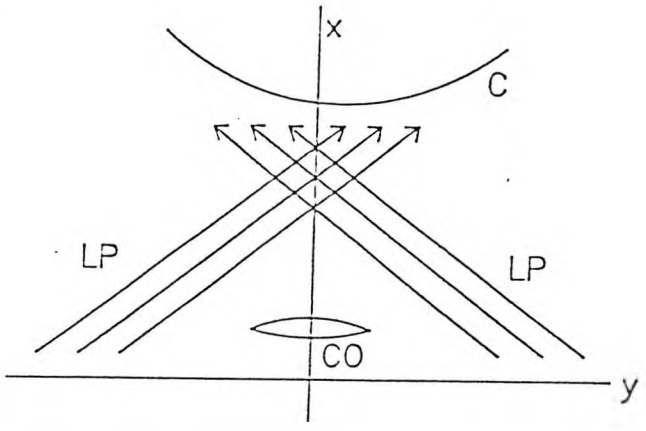


Fig. 5.1. Projector and camera set-up with arbitrary x-y axes aligned with camera axes. CO is the camera objective and LP are the light planes projected onto the cornea C.

The very faint images from the cornea were enhanced by instilling a drop of fluorescein in the eye; also a histogram equalisation image enhancement was applied to the camera input which increased the contrast between the light bands and the background.

A discrete point P is calculated on the cornea in x,y,z space by finding the

intersection of a given light plane with a ray from an image point P' on the camera image as shown in fig. 5.2.

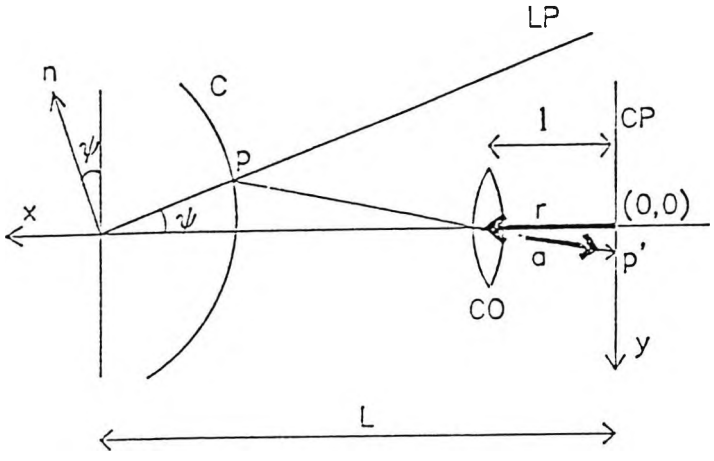


Fig. 5.2. Intersection of ray and plane LP gives point P in x,y,z space on the corneal surface C. The camera objective is CO and CP is the camera image plane. The normal to the plane is given by n.

The general equation of the plane LP is (5.1)

(5.1) $\hat{n}_p \bullet \left(\vec{r}_p - \vec{r}_{p0}\right) = 0$

where $\hat{n}_p$ is a unit vector normal to the plane

$\vec{r}_{p0}$ is a vector to a point on the plane

$\vec{r}_p$ is a general vector in x,y,z space

The unit vector normal to the plane is found by measuring the angle between the plane and the camera axis.

(5.2) $\hat{n}_p = \sin\psi\hat{x} - \cos\psi\hat{y}$

If the intersection point of the plane with the x axis is at distance L ,

then $\vec{r}_{p0}$ can be taken along the x axis to this point.

(5.3) $\vec{r}_{p0} = L\hat{x}$

When ψ and L are measured for each plane, the corresponding plane equation

can be found from (5.1)

(5.4) $(\sin\psi\hat{x} - \cos\psi\hat{y}) \bullet (x\hat{x} + y\hat{y} + z\hat{z} - L\hat{x}) = 0$

(5.5) $(\sin\psi\hat{x} - \cos\psi\hat{y}) \bullet ((x - L)\hat{x} + y\hat{y} + z\hat{z}) = 0$

(5.6) $\sin\psi(x - L) - y\cos\psi = 0$

Equation (5.6) is the equation for each plane in x,y,z space

The general equation of a light ray from P to P' is given by

(5.7) $\vec{r}_R = \vec{r}_{R0} + \vec{a}_R t$

where $\vec{r}_{R0}$ is a vector to the ray

$\vec{a}_R$ is a unit vector along the ray

$\vec{r}_R$ is a general vector in x,y,z space

t is a scalar distance parameter along the ray.

For each image point $P'(0, y'_p, z'_p)$, the ray equation (5.7) can be solved.

If the distance from the camera focal plane CP to the centre of the camera

objective CO = l, then $\vec{r}_{RO}$ can be taken along the x axis to CO.

(5.8) $\vec{r}_R = l\hat{x} + t(\hat{x}l - y'_p\hat{y} - z'_p\hat{z})$

(5.9) $x\hat{x} + y\hat{y} + z\hat{z} = (tl + l)\hat{x} - ty'_p\hat{y} - tz'_p\hat{z}$

equating components

(5.10) $x = l(t + 1)$

(5.11) $y = -y'_p t$

(5.12) $z = -z'_p t$

The value of t at the ray-plane intersection is found be inserting (5.10), (5.11) and

(5.12) into equation (5.6).

(5.13) $\sin\psi(l(t + 1) - L) + y'_p t\cos\psi = 0$

(5.14) $t\left(l\sin\psi + y'_p\cos\psi\right) = (L-l)\sin\psi$

(5.15) $t = \dfrac{(L-l)\sin\psi}{l\sin\psi + y'_p\cos\psi}$

The x,y,z point P is found by substituting t in (5.15) back into (5.10),(5.11) and (5.12). This gives a unique point on the corneal surface in 3-dimensional space, dependent only on image distance y'_p (for a given plane).

These equations are solved at chosen points on each plane; there are 26 light planes for which the data is collected producing a matrix of discrete points on the corneal surface.

5.2. Reconstruction of surface independent of x,y,z co-ordinate system (defined by the camera system).

After cubic interpolation (using eqn (5.26)) between the discrete points to produce a set of points on a uniform matrix, a surface equation can be developed using u,v co-ordinate axes defined in the surface as shown in fig. 5.3.

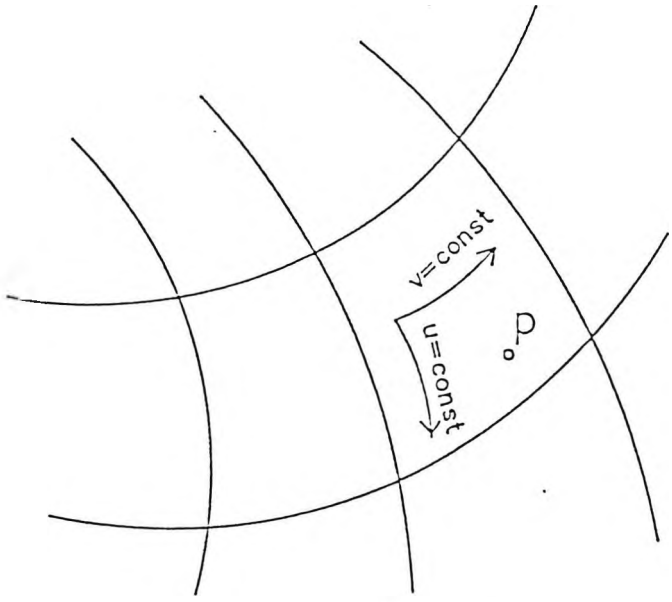


Fig. 5.3. Co-ordinate axes u,v on surface which uniquely define any point p on the surface.

The form of the surface equation and meaning of the u,v parameters can be illustrated by discussing the construction of a simple bilinear surface between 4 points (Rogers and Adams, 1976). This is a surface in which the boundaries between points are linear and the interpolation between boundaries are linear. Using 4 points in x,y,z space, a u,v co-ordinate system is constructed through the points with P_0, P_1,P_2,P_3 at u,v co-ordinates (0,0),(0,1),(1,0) and (1,1) respectively as shown in fig. 5.4.

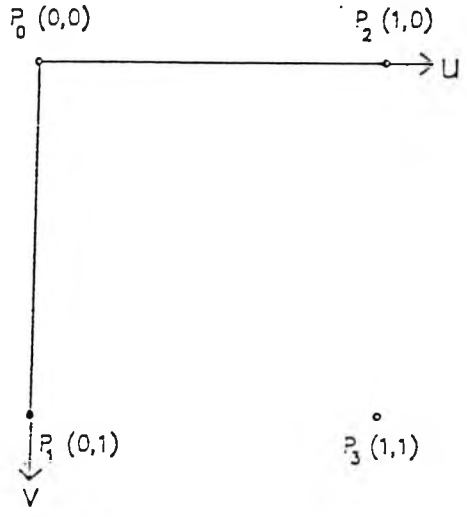


Fig. 5.4. Four points in u,v parameter space.

Linear interpolation between P_0 and P_1 gives (in vector notation)

(5.16) $\bar{P}_0 + v(\bar{P}_1 - \bar{P}_0) = \bar{Q}_{0v}$

Equation (5.16) represents 3 individual equations for the x,y,z components. Parameter v then represents the fractional distance along the line P_0 to P_1 . e.g. v=0.5 lies halfway between P_0 and P_1 . Similar interpolation between P_2 and P_3 gives

(5.17) $\bar{Q}_{1v} = \bar{P}_2 + v(\bar{P}_3 - \bar{P}_2)$

Equations (5.16) and (5.17) represent linear boundaries at constant u. Interpolation between the boundaries for any u at a given v is illustrated in fig. 5.5.

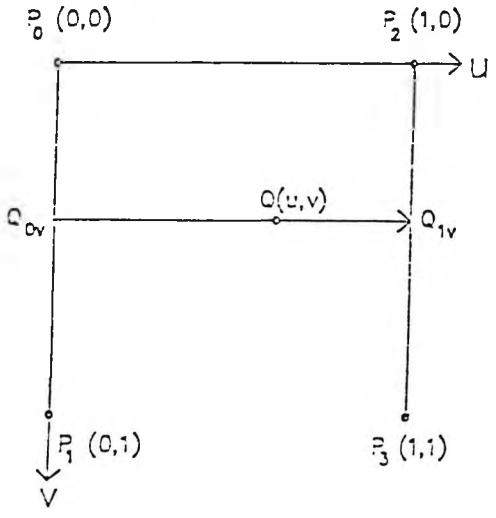


Fig. 5.5. Bilinear interpolation gives x,y,z values of a point $\bar{Q}$ for a given value of u and v.

For a given v, a linear interpolation between the boundaries gives

(5.18) $\bar{Q}(u,v) = \bar{Q}_{0v} + u\left(\bar{Q}_{1v} - \bar{Q}_{0v}\right)$

(5.19) $\bar{Q}(u,v) = (1-u)\bar{Q}_{0v} + u\bar{Q}_{1v}$

Substituting $\bar{Q}_{0v}$ and $\bar{Q}_{1v}$ from (5.16) and (5.17) gives

(5.20) $\bar{Q}(u,v) = (1-u)\left[\bar{P}_0 + v\left(\bar{P}_1 - \bar{P}_0\right)\right] + u\left[\bar{P}_2 + v\left(\bar{P}_3 - \bar{P}_2\right)\right]$

expanding (5.20) we obtain

(5.21) $\bar{Q}(u,v) = (1-u)(1-v)\bar{P}_0 + (1-u)v\bar{P}_1 + u(1-v)\bar{P}_2 + uv\bar{P}_3$

and in matrix notation (5.21) becomes

$$(5.22)\quad \bar{Q}(u,v)=\begin{bmatrix}(1-v)(1-u)\bar{P}_0 & v(1-u)\bar{P}_1\\ (1-v)u\bar{P}_2 & vu\bar{P}_3\end{bmatrix}$$

When the coefficients are separated from the point matrix, the surface equation takes the form

$$(5.23)\quad \bar{Q}(u,v)=\begin{bmatrix}(1-u) & u\end{bmatrix}\begin{bmatrix}\bar{P}_0 & \bar{P}_1\\ \bar{P}_2 & \bar{P}_3\end{bmatrix}\begin{bmatrix}1-v\\ v\end{bmatrix}$$

The coefficients (1-u),u,(1-v),v can be thought of as weighting functions for different points.

This method of development of a bilinear surface can be applied to a polynomial surface if the linear interpolation function is replaced by a polynomial function. A polynomial surface equation using a spline interpolation between 16 points has been used to construct the surface. The equation for each component x,y,z of the surface between 4 points is (Beach, 1990)

$$(5.24)\quad \bar{Q}(u,v)=\begin{bmatrix}\frac{1}{36}\end{bmatrix}\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}u^3\\ u^2\\ u\\ 1\end{bmatrix}^T\begin{bmatrix}P_{00} & P_{01} & P_{02} & P_{03}\\ P_{10} & P_{11} & P_{12} & P_{13}\\ P_{20} & P_{21} & P_{22} & P_{23}\\ P_{30} & P_{31} & P_{32} & P_{33}\end{bmatrix}\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}^T\begin{bmatrix}v^3\\ v^2\\ v\\ 1\end{bmatrix}$$

where $\begin{bmatrix}P_{00} & P_{01} & P_{02} & P_{03}\\ P_{10} & P_{11} & P_{12} & P_{13}\\ P_{20} & P_{21} & P_{22} & P_{23}\\ P_{30} & P_{31} & P_{32} & P_{33}\end{bmatrix}$ is a matrix of x,y, or z values.

Using an iterative process, the point matrix is adjusted until the surface passes through all the original points within some acceptable error and degree of smoothing.

Other interpolation functions giving the required degree of accuracy (see chapter 6.1.4) have been found to be a 9 point spline interpolation

$$(5.25)\bar{Q}(u,v)=\begin{bmatrix}\frac{1}{4}\end{bmatrix}\begin{bmatrix}1 & -2 & 1\\ -2 & 2 & 0\\ 1 & 1 & 0\end{bmatrix}\begin{bmatrix}u^2\\ u\\ 1\end{bmatrix}^T\begin{bmatrix}p_{00} & p_{01} & p_{11}\\ p_{10} & p_{11} & p_{12}\\ p_{20} & p_{21} & p_{22}\end{bmatrix}\begin{bmatrix}1 & -2 & 1\\ -2 & 2 & 0\\ 1 & 1 & 0\end{bmatrix}^T\begin{bmatrix}v^2\\ v\\ 1\end{bmatrix}$$

and a 9 point cubic interpolation

$$(5.26)\bar{Q}(u,v) = \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} u^2 \\ u \\ 1 \end{bmatrix}^T \begin{bmatrix} p_{00} & p_{01} & p_{11} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & p_{22} \end{bmatrix} \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix}^T \begin{bmatrix} v^2 \\ v \\ 1 \end{bmatrix}$$

The whole corneal surface is constructed by building patches between points in the point matrix using (5.24), (5.25) or (5.26).

5.3. Extracting values from the surface.

The most common parameter for describing the corneal surface is radius of curvature; radius of curvature can be calculated at any point on the surface using differential geometry theory (Stoker, 1969).

The problem of calculating surface curvature is approached by constructing a curve in the surface and then calculating the curvature of this curve. The radius of curvature is the reciprocal of the curvature. For a general plane curve the curvature is defined as

$$(5.27)\ \frac{d\bar{t}}{ds} = k\hat{N}_p$$

$\bar{t}$ = tangent to curve in plane of curve.

s = arc length along curve.

k = curvature.

$\hat{N}_p$ = normal in plane of curve.

For a plane curve defined by the intersection of the surface with any curve, the curvature can be separated into 2 components. One normal component - the curvature projection onto a plane in a given direction cutting the surface normally and one tangential component - the curvature projection onto the surface

tangent plane.

(5.28) $\frac{d\bar{t}}{ds} = k_n \hat{N} + k_g \hat{T}$

k_n = normal curvature.

$\hat{N}$ = unit surface normal vector.

k_g = tangential or geodesic curvature.

$\hat{T}$ = unit tangent vector in tangent plane.

The curvature in the normal plane is the surface curvature in the direction of the normal and has geodesic curvature = 0.

For geodesic curvature = 0

(5.29) $\frac{d\bar{t}}{ds} = k_n \hat{N}$

taking scalar product of (5.29) with the normal $\hat{N}$ gives

(5.30) $\left(\frac{d\bar{t}}{ds}\right) \bullet \hat{N} = k_n \hat{N} \bullet \hat{N} = k_n$

But for the tangent to any curve on the surface

(5.31) $\hat{N} \bullet \bar{t} = 0$

differentiating (5.31) with respect to arc length yealds

(5.32) $\frac{d}{ds}\left(\hat{N} \bullet \bar{t}\right) = \frac{d\bar{t}}{ds} \bullet \hat{N} + \bar{t} \bullet \frac{d\hat{N}}{ds} = 0$

(5.33) $\therefore \frac{d\bar{t}}{ds} \bullet \hat{N} = -\bar{t} \bullet \frac{d\hat{N}}{ds}$

with (5.30) the curvature k_n is simply the left hand side of the above equation,

therefore the curvature is given by

(5.34) $k_n = -\bar{t} \bullet \frac{d}{ds} \hat{N}$

and the tangent vector is the derivative with respect to arc length along the surface,

therefore

(5.35) $\bar{t} = \frac{d\bar{x}}{ds}$

where $\bar{x}$ is a general vector to the curve in x,y,z space.

The curvature in the normal plane is given by

(5.36) $k_n = -\frac{d\bar{x}}{ds} \bullet \frac{d\hat{N}}{ds} = -\frac{d\bar{x} \bullet d}{ds^2} \hat{N}$

and arc length squared ds^2 is by definition

(5.37) $ds^2 = d\bar{x} \bullet d\bar{x}$

(5.38) $\therefore k_n = -\frac{d\bar{x} \bullet d\hat{N}}{d\bar{x} \bullet d\bar{x}}$

k_n is the curvature of a curve C in the surface and also in a plane containing

the normal to the surface as shown in Fig. 5.6.

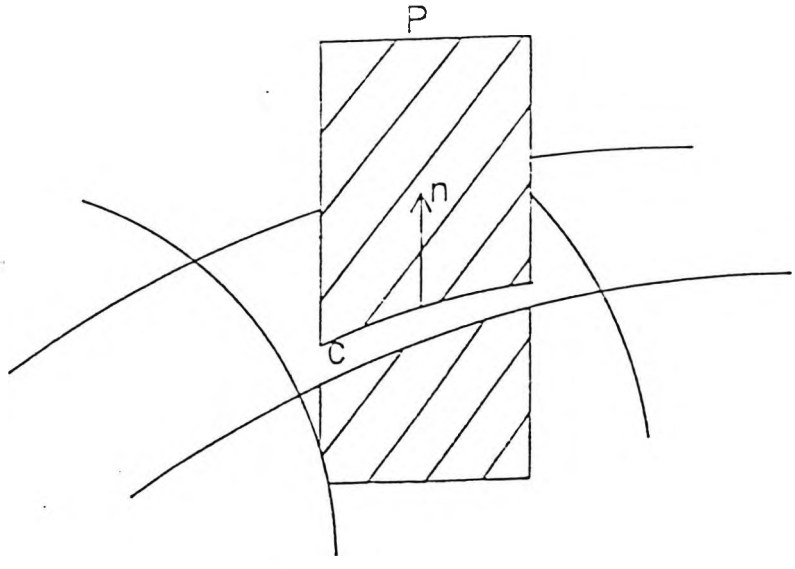


Fig. 5.6. Plane P containing normal n to surface and whose intersection

with surface defines a curve C.

5.3.1. Calculation of denominator $d\bar{x} \bullet d\bar{x}$ in equation (5.38).

For a surface given as a function of 2 parameters u,v

(5.39) $$d\bar{x} = \frac{\partial \bar{x}}{\partial u} du + \frac{\partial \bar{x}}{\partial v} dv$$

(5.40) $$d\bar{x} \bullet d\bar{x} = (\bar{x}_u du + \bar{x}_v dv) \bullet (\bar{x}_u du + \bar{x}_v dv)$$

where $\bar{x}_u = \frac{\partial \bar{x}}{\partial u}$ and $\bar{x}_v = \frac{\partial \bar{x}}{\partial v}$

(5.41) $$d\bar{x} \bullet d\bar{x} = \bar{x}_u \bullet \bar{x}_u du^2 + 2\bar{x}_u \bullet \bar{x}_v du dv + \bar{x}_v \bullet \bar{x}_v dv^2$$

(5.42) $$d\bar{x} \bullet d\bar{x} = E du^2 + 2F du dv + G dv^2$$

where

(5.43) $$E = \bar{x}_u \bullet \bar{x}_u \ , \ F = \bar{x}_u \bullet \bar{x}_v \ , \ G = \bar{x}_v \bullet \bar{x}_v$$

Equation (5.42) is called the First Fundamental Form of the surface and is usually donated by I; its square root ds is called the 'element of arc'.

The general vector to the surface $\bar{x}$ is given by

(5.44) $$\bar{x} = x(u,v)\hat{x} + y(u,v)\hat{y} + z(u,v)\hat{z}$$

and taking partial derivatives of $\bar{x}$ with respect to u and v gives

(5.45) $$\bar{x}_u = \frac{\partial x(u,v)}{\partial u}\hat{x} + \frac{\partial y(u,v)}{\partial u}\hat{y} + \frac{\partial z(u,v)}{\partial u}\hat{z}$$

and

$$(5.46)\quad \bar{x}_v = \frac{\partial x(u,v)}{\partial v}\hat{x} + \frac{\partial y(u,v)}{\partial v}\hat{y} + \frac{\partial z(u,v)}{\partial v}\hat{z}$$

$\therefore$ (5.43) becomes

$$(5.47)\quad E = \frac{\partial x}{\partial u}\frac{\partial x}{\partial u} + \frac{\partial y}{\partial u}\frac{\partial y}{\partial u} + \frac{\partial z}{\partial u}\frac{\partial z}{\partial u} = x_u^2 + y_u^2 + z_u^2$$

$$(5.48)\quad F = \frac{\partial x}{\partial u}\frac{\partial x}{\partial v} + \frac{\partial y}{\partial u}\frac{\partial y}{\partial v} + \frac{\partial z}{\partial u}\frac{\partial z}{\partial v} = x_u x_v + y_u y_v + z_u z_v$$

$$(5.49)\quad G = \frac{\partial x}{\partial v}\frac{\partial x}{\partial v} + \frac{\partial y}{\partial v}\frac{\partial y}{\partial v} + \frac{\partial z}{\partial v}\frac{\partial z}{\partial v} = x_v^2 + y_v^2 + z_v^2$$

5.3.2. Calculation of numerator $d\bar{x} \bullet d\hat{N}$ in equation (5.38).

$$(5.50)\quad d\bar{N} = \frac{\partial \bar{N}}{\partial u}du + \frac{\partial \bar{N}}{\partial v}dv$$

$$(5.51)\quad d\bar{N} = \hat{N}_u du + \hat{N}_v dv$$

where

$$\hat{N}_u = \frac{\partial \hat{N}}{\partial u} \text{ and } \hat{N}_v = \frac{\partial \hat{N}}{\partial v}$$

then

$$(5.52)\quad d\bar{x} \bullet d\hat{N} = \left(\bar{x}_u du + \bar{x}_v dv\right) \bullet \left(\hat{N}_u du + \hat{N}_v dv\right)$$

(5.53) $$= \bar{x}_u \bullet \hat{N}_u du^2 + \bar{x}_u \bullet \hat{N}_v du dv + \bar{x}_v \bullet \hat{N}_u dv du + \bar{x}_v \bullet \hat{N}_v dv^2$$

(5.54) $$\therefore k_n = -\frac{\bar{x}_u \bullet \hat{N}_u du^2 + \left(\bar{x}_u \bullet \bar{N}_v + \bar{x}_v \bullet \hat{N}_u\right) dv du + \bar{x}_v \bullet \hat{N}_v dv^2}{E du^2 + 2F du dv + G dv^2}$$

(5.55) $$k_n = \frac{e du^2 + 2f du dv + g dv^2}{E du^2 + 2F du dv + G dv^2}$$

where

$$e = -\bar{x}_u \bullet \hat{N}_u \ , \ 2f = -\left(\bar{x}_u \bullet \bar{N}_v + \bar{x}_v \bullet \hat{N}_u\right) \ , \ g = -\bar{x}_v \bullet \hat{N}_v$$

The numerator $e du^2 + 2f du dv + g dv^2$ is called the Second Fundamental Form of the surface.

At any point on the surface, 2 tangent vectors to the surface in the direction of the parameter axes are found by taking partial derivatives of the surface equation (5.24). The vector cross product of these tangent vectors gives a vector normal to the surface at that point, as shown in fig. 5.7.

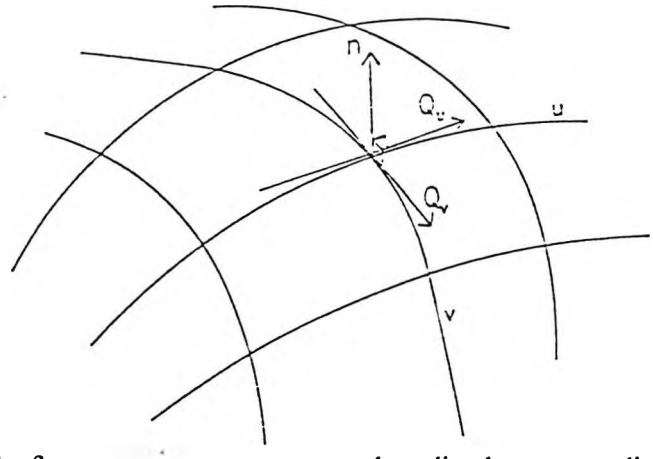


Fig. 5.7. Surface tangent vectors Q_u and Q_v lie along co-ordinate axes with surface normal vector n orthogonal to both Q_u and Q_v.

(5.56) $\therefore \vec{x}_u \bullet \hat{N} = 0$

(5.57) $\vec{x}_v \bullet \hat{N} = 0$

and the normal is given by

(5.58) $\hat{N} = \dfrac{\vec{x}_u \times \vec{x}_v}{|\vec{x}_u \times \vec{x}_v|}$

To remove the partial derivatives ($\hat{N}_v$ and $\hat{N}_u$) of the normal from the second fundamental form, we take partial derivatives of equations (5.56) and (5.57).

Taking the partial derivative of Eqn. (5.56) with respect to parameter u

(5.59) $\dfrac{\partial}{\partial u}\left(\vec{x}_u \bullet \hat{N}\right) = \vec{x}_{uu} \bullet \hat{N} + \vec{x}_u \bullet \hat{N}_u = 0$

(5.60) $\vec{x}_u \bullet \hat{N}_u = -\vec{x}_{uu} \bullet \hat{N}$

taking the partial derivative of Eqn. (5.57) with respect to parameter v

(5.61) $\dfrac{\partial}{\partial u}\left(\vec{x}_v \bullet \hat{N}\right) = \vec{x}_{vv} \bullet \hat{N} + \vec{x}_v \bullet \hat{N}_v = 0$

(5.62) $\vec{x}_v \bullet \hat{N}_v = -\vec{x}_{vv} \bullet \hat{N}$

and using (5.56) and (5.57) taking cross differentials with respect to $\vec{x}_v$ and $\vec{x}_u$

(5.63) $\dfrac{\partial}{\partial u}\left(\vec{x}_v \bullet \hat{N}\right) + \dfrac{\partial}{\partial v}\left(\vec{x}_u \bullet \hat{N}\right) = 0$

expanding equation (5.63)

(5.64) $\therefore \vec{x}_{uv} \bullet \hat{N} + \vec{x}_u \bullet \hat{N}_v + \vec{x}_{vu} \bullet \hat{N} + \vec{x}_v \bullet \hat{N}_u = 0$

and re-arranging

(5.65) $2\bar{x}_{uv} \bullet \hat{N} = -\left(\bar{x}_u \bullet \hat{N}_v + \bar{x}_v \bullet \hat{N}_u\right)$

Inserting the results given by (5.60), (5.62) and (5.65) into e,f and g of the second fundamental form, we can remove the normal partial derivatives.

(5.66) $\therefore e = -\bar{x}_u \bullet \hat{N}_u = \bar{x}_{uu} \bullet \hat{N}$ from (5.60)

(5.67) $f = -\frac{1}{2}\left(\bar{x}_u \bullet \hat{N}_v + \bar{x}_v \bullet \hat{N}_u\right) = \bar{x}_{uv} \bullet \hat{N}$

(5.68) $g = -\bar{x}_v \bullet \hat{N}_v = \bar{x}_{vv} \bullet \hat{N}$ with (5.62)

and the normal is given by

(5.58) $\hat{N} = \frac{\bar{x}_u \times \bar{x}_v}{\left|\bar{x}_u \times \bar{x}_v\right|}$

We may now replace the normal $\hat{N}$ completely by partial derivatives of the surface equation.

Using Lagranges' identity $\left(\bar{a} \times \bar{b}\right) \bullet \left(\bar{a} \times \bar{b}\right) = \left(\bar{a} \bullet \bar{a}\right)\left(\bar{b} \bullet \bar{b}\right) - \left(\bar{a} \bullet \bar{b}\right)^2$

(5.69) $\left|\bar{x}_u \times \bar{x}_v\right| = \sqrt{\left(\bar{x}_u \times \bar{x}_v\right) \bullet \left(\bar{x}_u \times \bar{x}_v\right)}$

(5.70) $= \sqrt{\left(\bar{x}_u \bullet \bar{x}_u\right)\left(\bar{x}_v \bullet \bar{x}_v\right) - \left(\bar{x}_u \bullet \bar{x}_v\right)^2}$

(5.71) $= \sqrt{EG - F^2}$

Substituting (5.71) back into equation (5.58) for $\hat{N}$

(5.72) $\hat{N} = \frac{\bar{x}_u \times \bar{x}_v}{\sqrt{EG - F^2}}$

The expression for $\hat{N}$ contains only partial derivatives of the surface equation and can be substituted for $\hat{N}$ in the expressions for e,f and g.

$$(5.73)\quad \therefore e = \vec{x}_{uu} \bullet \frac{(\vec{x}_u \times \vec{x}_v)}{\sqrt{EG - F^2}}$$

expanding e into component form

$$(5.74)\quad e = \frac{\begin{vmatrix} x_{uu} & y_{uu} & z_{uu} \\ x_u & y_u & z_u \\ x_v & y_v & z_v \end{vmatrix}}{\sqrt{EG - F^2}}$$

The expression for f becomes

$$(5.75)\quad f = \vec{x}_{uv} \bullet \vec{N} = \vec{x}_{uv} \bullet \frac{\vec{x}_u \times \vec{x}_v}{\sqrt{EG - F^2}}$$

and expanding f into component form gives

$$(5.76)\quad f = \frac{\begin{vmatrix} x_{uv} & y_{uv} & z_{uv} \\ x_u & y_u & z_u \\ x_v & y_v & z_v \end{vmatrix}}{\sqrt{EG - F^2}}$$

The expression for g becomes

$$(5.77)\quad g = \vec{x}_{vv} \bullet \hat{N} = \vec{x}_{vv} \bullet \frac{\vec{x}_u \times \vec{x}_v}{\sqrt{EG - F^2}}$$

and expanding g into component form gives

$$(5.78)\quad g = \frac{\begin{vmatrix} x_{vv} & y_{vv} & z_{vv} \\ x_u & y_u & z_u \\ x_v & y_v & z_v \end{vmatrix}}{\sqrt{EG - F^2}}$$

5.3.3. Calculation of curvature for a given surface equation $\bar{Q}(u, v)$

By applying the surface equation $\bar{Q}(u, v)$ in (5.24) the equation for curvature k_n given by (5.55), the curvature at any point u,v on the corneal surface in any direction can be calculated.

After dividing (5.55) by du^2, the equation for curvature is given by

$$(5.79)\quad k_n = \frac{e + 2f\left(\frac{dv}{du}\right) + g\left(\frac{dv}{du}\right)^2}{E + 2F\left(\frac{dv}{du}\right) + G\left(\frac{dv}{du}\right)^2}$$

and e,f,g,E,F,G are given by (5.66), (5.67) and (5.68)

$$e = \bar{x}_{uu} \bullet \hat{N} \quad (5.66)$$

$$f = \bar{x}_{uv} \bullet \hat{N} \quad (5.67)$$

$$g = \bar{x}_{vv} \bullet \hat{N} \quad (5.68)$$

and (5.43)

$$E = \bar{x}_u \bullet \bar{x}_u \ , \ F = \bar{x}_u \bullet \bar{x}_v \ , \ G = \bar{x}_v \bullet \bar{x}_v \quad (5.43)$$

The values e,f,g,E,F,G need to be calculated at a given u,v and for a given direction $\frac{dv}{du}$ to give the curvature k_n. This is easily done by substituting $\bar{Q}(u, v)$ in place of the general vector $\bar{x}$ in (5.66), (5.67), (5.68), (5.43) and (5.72) giving

$$(5.80)\quad e = \bar{Q}_{uu} \bullet \hat{N}$$

$$(5.81)\quad f = \bar{Q}_{uv} \bullet \hat{N}$$

$$(5.82)\quad g = \bar{Q}_{vv} \bullet \hat{N}$$

$$(5.83)\quad E = \bar{Q}_u \bullet \bar{Q}_u \ , \ F = \bar{Q}_u \bullet \bar{Q}_v \ , \ G = \bar{Q}_v \bullet \bar{Q}_v$$

In terms of component partial derivatives, the values E,F,G are given by (5.47), (5.48), (5.49)

$$E = x_u^2 + y_u^2 + z_u^2 \quad (5.47)$$

$$F = x_u x_v + y_u y_v + z_u z_v \quad (5.48)$$

$$G = x_v^2 + y_v^2 + z_v^2 \quad (5.49)$$

and e,f,g are given by (5.74), (5.76), (5.78)

$$e = \frac{\begin{vmatrix} x_{uu} & y_{uu} & z_{uu} \\ x_u & y_u & z_u \\ x_v & y_v & z_v \end{vmatrix}}{\sqrt{EG - F^2}} \quad (5.74)$$

$$f = \frac{\begin{vmatrix} x_{uv} & y_{uv} & z_{uv} \\ x_u & y_u & z_u \\ x_v & y_v & z_v \end{vmatrix}}{\sqrt{EG - F^2}} \quad (5.76)$$

$$g = \frac{\begin{vmatrix} x_{vv} & y_{vv} & z_{vv} \\ x_u & y_u & z_u \\ x_v & y_v & z_v \end{vmatrix}}{\sqrt{EG - F^2}} \quad (5.78)$$

The component partial derivatives are now calculated directly from the surface equation, e.g. the x component of the surface $\bar{Q}(u, v)$ in (5.24) is given by

$$(5.84)\quad x(u,v)=\left[\frac{1}{36}\right]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}u^3\\ u^2\\ u\\ 1\end{bmatrix}^T\left[P_x\right]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}^T\begin{bmatrix}v^3\\ v^2\\ v\\ 1\end{bmatrix}$$

where $\left[P_x\right]$ is a matrix of x co-ordinates for 16 points

$$\left[P_x\right]=\begin{bmatrix}P_{x00} & P_{x01} & P_{x02} & P_{x03}\\ P_{x10} & P_{x11} & P_{x12} & P_{x13}\\ P_{x20} & P_{x21} & P_{x22} & P_{x23}\\ P_{x30} & P_{x31} & P_{x32} & P_{x33}\end{bmatrix}$$

we have similar equations for y(u,v) and z(u,v) replacing point matrix $\left[P_x\right]$ in equation (5.84) by point matrices of y and z co-ordinates respectively.

The first partial derivatives are given by

$$(5.85)\quad \frac{\partial x}{\partial u}=x_u=\left[\frac{1}{36}\right]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}3u^2\\ 2u\\ 1\\ 0\end{bmatrix}^T\left[P_x\right]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}^T\begin{bmatrix}v^3\\ v^2\\ v\\ 1\end{bmatrix}$$

with identical equations for y_u and z_u using y and z point matrices respectively

and

$$(5.86)\quad \frac{\partial x}{\partial v}=x_v=\left[\frac{1}{36}\right]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}u^3\\ u^2\\ u\\ 0\end{bmatrix}^T\left[P_x\right]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}^T\begin{bmatrix}3v^2\\ 2v\\ 1\\ 0\end{bmatrix}$$

with identical equations for y_v and z_v using y and z point matrices respectively.

The second partial derivatives are given by

$$(5.87)\ \frac{\partial^2 x}{\partial u^2} = x_{uu} = \left[\frac{1}{36}\right]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}6u\\ 2\\ 0\\ 0\end{bmatrix}^T [P_x] \begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}^T \begin{bmatrix}v^3\\ v^2\\ v\\ 1\end{bmatrix}$$

with identical equations for y_{uu} and z_{uu} using y and z point matrices respectively

and

$$(5.88)\ \frac{\partial^2 x}{\partial v^2} = x_{vv} = \left[\frac{1}{36}\right]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}u^3\\ u^2\\ u\\ 1\end{bmatrix}^T [P_x] \begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}^T \begin{bmatrix}6v\\ 2\\ 0\\ 0\end{bmatrix}$$

with identical equations for y_{vv} and z_{vv} using y and z point matrices respectively.

The cross partial derivative is given by

$$(5.89)\ \frac{\partial^2 x}{\partial u \partial v} = x_{uv} = \left[\frac{1}{36}\right]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}3u^2\\ 2u\\ 1\\ 0\end{bmatrix}^T [P_x] \begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}^T \begin{bmatrix}3v^2\\ 2v\\ 1\\ 0\end{bmatrix}$$

with identical equations for y_{uv} and z_{uv} using y and z point matrices respectively.

5.3.4. Calculation of maximum and minimum curvature directions.

For a given normal at a point P on the surface, an infinite number of planes can be constructed for different directions $\frac{dv}{du}$ on the surface. At a point P,

the values e,f,g,E,F,G are independent of the chosen normal plane and therefore are constants at that point. The curvature k_n is then determined by the direction $\frac{dv}{du}$.

Setting $\frac{dv}{du} = \lambda$ equation (5.79) becomes

(5.90) $$k_n = \frac{e + 2f\lambda + g\lambda^2}{E + 2F\lambda + G\lambda^2}$$

The maximum and minimum values of k_n are found from the value of λ when $\frac{dk_n}{d\lambda} = 0$

(5.91) $$\frac{dk_n}{d\lambda} = \frac{(E + 2F\lambda + G\lambda^2)(2f + 2g\lambda) - (e + 2f\lambda + g\lambda^2)(2F + 2G\lambda)}{(E + 2F\lambda + G\lambda^2)^2} = 0$$

(5.92) $$\therefore (E + 2F\lambda + G\lambda^2)(f + g\lambda) - (e + 2f\lambda + g\lambda^2)(F + G\lambda) = 0$$

expanding (5.92) gives a quadratic equation for λ

(5.93) $$(Fg - fG)\lambda^2 + (Eg - eG)\lambda + (Ef - eF) = 0$$

for which the solution is

(5.94) $$\lambda = \frac{-(Eg - eG) \pm \sqrt{(Eg - eG)^2 - 4(Ef - eF)(Fg - fG)}}{2(Fg - fG)}$$

Equation (5.94) gives 2 directions for maximum and minimum k_n which can be found by substituting the values of λ into equation (5.90). The 2 directions for maximum and minimum k_n are called principle directions and can be shown to be orthogonal .

The mathematical analysis developed in this chapter will now enable the calculation of the radius of curvature at any discrete point on the corneal surface, for any shape of cornea whether spherical, aspherical or highly irregular.

Chapter 6

Topography measurements results

6.1 Accuracy and precision

6.1.1. Theoretical surface depth resolution

The measurement system depth resolution depends on three factors -

1/ The angle of the light planes relative to the camera axis.

2/ The magnification of the camera system.

3/ The pixel resolution of the computer imaging system.

To view the whole of the cornea and assign the geometric centre of the cornea as a reference point, the camera magnification was fixed to image 12 mm horizontally on the video screen. The angle of the light planes was set to approximately 60° (measured accurately during calibration) to allow measurements to be made with particular attention to the central 6mm of the cornea. The frame grabber resolution was 768 pixels horizontally x 512 pixels vertically. This configuration gave a pixel width of 15.62 μm. The depth resolution is calculated as shown in fig. 6.1 and is given by

(6.1) depth resolution=tan 30° x 15.62μm = 9.01μm

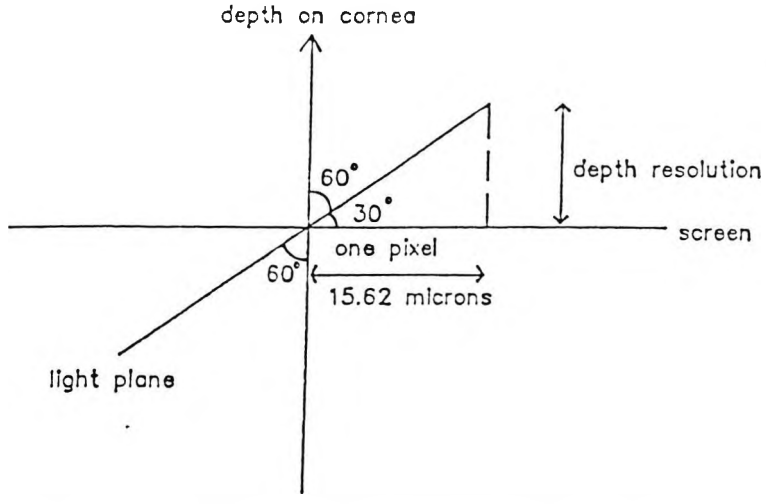


fig. 6.1 The depth resolution is calculated from the pixel width and angle of light plane.

6.1.2. Practical surface depth resolution.

To test if the theoretical depth resolution calculated in 6.1.1. could be achieved, a flat screen was mounted on a high precision linear stage micrometer and placed in front of the video camera. The linear stage micrometer was accurate to within 1.0μm. Screen distance measurements were made at increasing screen distances of 200 μm intervals up to 1600μm from the initial screen position. For each distance, 5 measurements were made with each measurement consisting of a matrix of 12 horizontal x 8 vertical readings, covering an area of 10mm x 8mm to test uniformity. The results are shown in table T.6.1. and show a mean distance measurement error of -3.0 ± 6.5μm.

distance moved μm	mean measured distance in μm	distance s.d. μm	variation across screen (s.d. μm)
200	196.5	6.2	3.6
400	398.5	6.2	5.5
600	598.0	6.5	6.0
800	799.8	4.4	5.0
1000	1003.5	5.2	6.0
1200	1205.0	6.5	5.3
1400	1393.8	5.7	6.0
1600	1597.2	8.5	3.7

Table T.6.1. Results of screen distance measurements at 8 different screen positions covering 1.6mm depth.

A typical set of results for the screen distance measurement is shown in table T.6.2., covering one side of the screen and an area of 5mm horizontally x 8mm vertically.

RIGHT measurement results					
x[142.502]	x[142.510]	x[142.505]	x[142.500]	x[142.507]	x[142.503]
x[142.502]	x[142.498]	x[142.493]	x[142.500]	x[142.507]	x[142.503]
x[142.490]	x[142.498]	x[142.505]	x[142.500]	x[142.507]	x[142.503]
x[142.490]	x[142.498]	x[142.505]	x[142.500]	x[142.507]	x[142.503]
x[142.490]	x[142.498]	x[142.505]	x[142.500]	x[142.507]	x[142.503]
x[142.490]	x[142.498]	x[142.505]	x[142.500]	x[142.507]	x[142.503]
x[142.502]	x[142.498]	x[142.505]	x[142.500]	x[142.507]	x[142.515]
x[142.502]	x[142.498]	x[142.505]	x[142.500]	x[142.507]	x[142.515]

Table T.6.2. Depth measurements of one side of screen at a distance of 142.50mm from the camera. All values are given in mm.

6.1.3. Theoretical curvature accuracy.

To test if the surface fitting and curvature calculations would work in principle, theoretical x,y,z points were generated at equal intervals on a sphere and a surface equation was fitted to them. The x,y,z values of each point was calculated using spherical polar co-ordinates.

(6.2) $x = r \sin\varphi \cos\vartheta$

(6.3) $y = r \sin\varphi \sin\vartheta$

(6.4) $z = r \cos\varphi$

This is shown below in fig 6.2

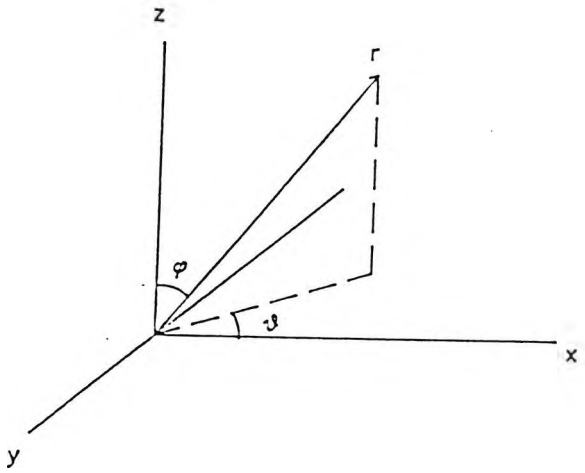


fig 6.2 Standard spherical polar co-ordinates used to calculate points on a theoretical sphere.

The theoretical sphere calculations were made for radius of curvature values ranging from 6.5mm to 9.0mm in 0.5mm steps. The chord distance between points ranged from 0.2mm to 1.2mm. The results are shown in table T.6.3.

Theoretical radius of curvature. mm.	Point separation in degrees	Measured radius of curvature(min) mm	Measured radius of curvature (max)mm
6.5	0.5	6.500	6.500
6.5	1.0	6.499	6.499
6.5	1.5	6.497	6.497
6.5	2.0	6.495	6.495
7.0	0.5	7.000	7.000
7.0	1.0	6.999	6.999
7.0	1.5	6.997	6.997
7.0	2.0	6.995	6.995
7.5	0.5	7.500	7.500
7.5	1.0	7.499	7.499
7.5	1.5	7.497	7.497
7.5	2.0	7.494	7.495
8.0	0.5	8.000	8.000
8.0	1.0	7.998	7.999
8.0	1.5	7.997	7.997
8.0	2.0	7.994	7.994
8.5	0.5	8.500	8.500
8.5	1.0	8.498	8.498
8.5	1.5	8.496	8.496
8.5	2.0	8.494	8.494
9.0	0.5	9.000	9.000
9.0	1.0	8.998	8.998
9.0	1.5	8.996	8.996
9.0	2.0	8.993	8.994

Table T.6.3 Curvature measurements for different theoretical spheres.

The results in table T.6.3. show a mean error of 0.004mm. The results are calculated by finding the maximum and minimum radius of curvature values at 25 equally spaced points across the fitted surface. A typical results printout is shown in table T.6.4.

patch results

x	[1.473]	x	[1.472]	x	[1.472]	x	[1.471]	x	[1.471]
y	[0.281]	y	[0.284]	y	[0.286]	y	[0.289]	y	[0.291]
z	[7.858]	z	[7.858]	z	[7.858]	z	[7.858]	z	[7.858]
r1	[-8.000]	r1	[-8.000]	r1	[-8.000]	r1	[-8.000]	r1	[-8.000]
r2	[-8.000]	r2	[-8.000]	r2	[-8.000]	r2	[-8.000]	r2	[-8.000]
x	[1.486]	x	[1.486]	x	[1.485]	x	[1.485]	x	[1.484]
y	[0.284]	y	[0.286]	y	[0.289]	y	[0.291]	y	[0.294]
z	[7.856]	z	[7.856]	z	[7.856]	z	[7.856]	z	[7.856]
r1	[-8.000]	r1	[-8.000]	r1	[-8.000]	r1	[-8.000]	r1	[-8.000]
r2	[-8.000]	r2	[-8.000]	r2	[-8.000]	r2	[-8.000]	r2	[-8.000]
x	[1.500]	x	[1.499]	x	[1.499]	x	[1.498]	x	[1.498]
y	[0.286]	y	[0.289]	y	[0.291]	y	[0.294]	y	[0.297]
z	[7.853]	z	[7.853]	z	[7.853]	z	[7.853]	z	[7.853]
r1	[-8.000]	r1	[-8.000]	r1	[-8.000]	r1	[-8.000]	r1	[-8.000]
r2	[-8.000]	r2	[-8.000]	r2	[-8.000]	r2	[-8.000]	r2	[-8.000]
x	[1.513]	x	[1.513]	x	[1.512]	x	[1.512]	x	[1.511]
y	[0.289]	y	[0.291]	y	[0.294]	y	[0.297]	y	[0.299]
z	[7.850]	z	[7.850]	z	[7.850]	z	[7.850]	z	[7.850]
r1	[-8.000]	r1	[-8.000]	r1	[-8.000]	r1	[-8.000]	r1	[-8.000]
r2	[-8.000]	r2	[-8.000]	r2	[-8.000]	r2	[-8.000]	r2	[-8.000]
x	[1.526]	x	[1.526]	x	[1.525]	x	[1.525]	x	[1.524]
y	[0.291]	y	[0.294]	y	[0.297]	y	[0.299]	y	[0.302]
z	[7.847]	z	[7.847]	z	[7.847]	z	[7.847]	z	[7.847]
r1	[-8.000]	r1	[-8.000]	r1	[-8.000]	r1	[-8.000]	r1	[-8.000]
r2	[-8.000]	r2	[-8.000]	r2	[-8.000]	r2	[-8.000]	r2	[-8.000]

Table T.6.4. Curvature results for theoretical surface with 8.0mm radius of curvature and 0.28mm chord distance between points. r1 and r2 are the maximum and minimum radius of curvature values at each point. All values are in mm.

6.1.4. Practical curvature accuracy.

To test the accuracy and reproducibility of the Corneal Topography System (CTS) on real objects, measurements were made on steel calibration spheres of 6.5mm, 7.5mm, 8.0mm and 9.0mm diameter. Five readings were taken on each sphere, with the sphere moved slightly and system re-focused for each measurement. The results are shown in table T.6.5.

Radius of sphere mm	Measured radius of curvature mm	s.d.	average flattening mm/mm
6.5	6.51	0.05	0.020
7.5	7.51	0.03	0.000
8.0	8.02	0.03	0.005
9.0	8.98	0.05	0.010

Table T.6.5 Radius of curvature measurements on steel calibration spheres.

The results show a mean error of 0.01±0.04mm in radius of curvature measurement. A typical results printout for an 8.0mm radius of curvature sphere is shown in fig 6.3.

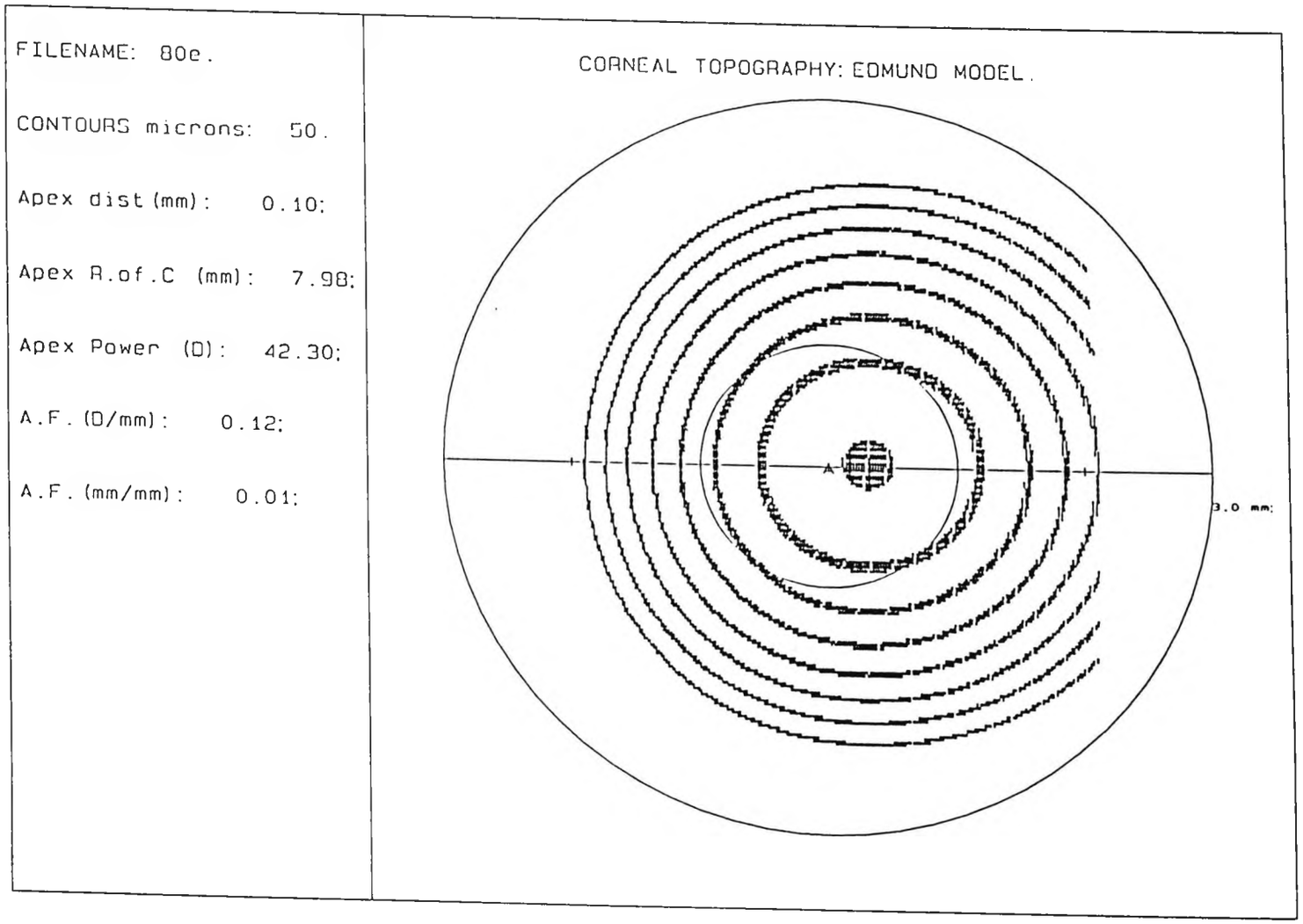


fig.6.3 Results for 8.0mm radius of curvature sphere.

6.2 Typical corneal topography results

Two typical plots of normal corneas are shown below in fig 6.4 and 6.5. The two plots show the two alternative forms of output available from the system, with or without radius of curvature values printed on the plot. The apex position is shown by the letter 'A'.

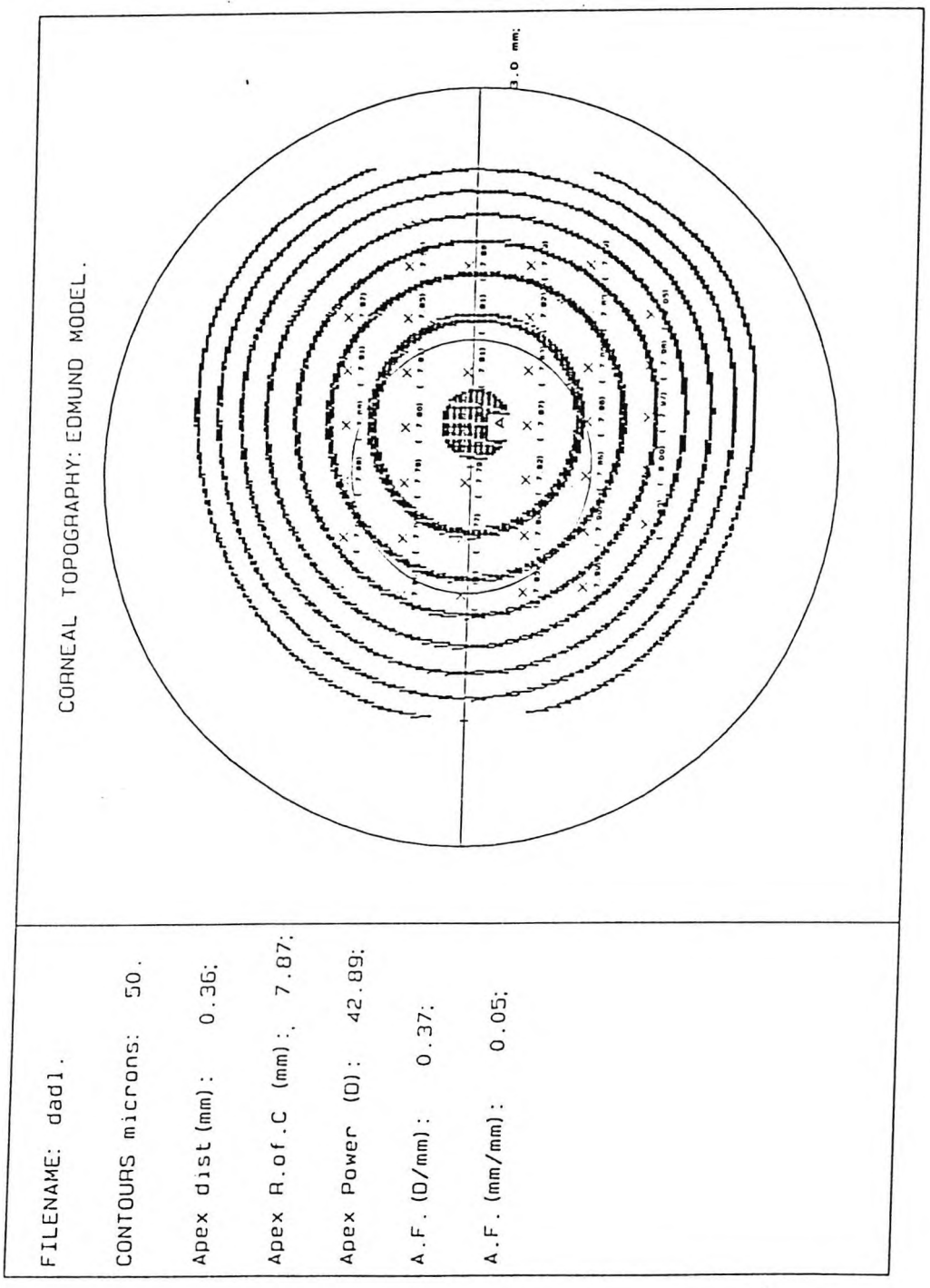


fig 6.4 Corneal plot showing radius of curvature values

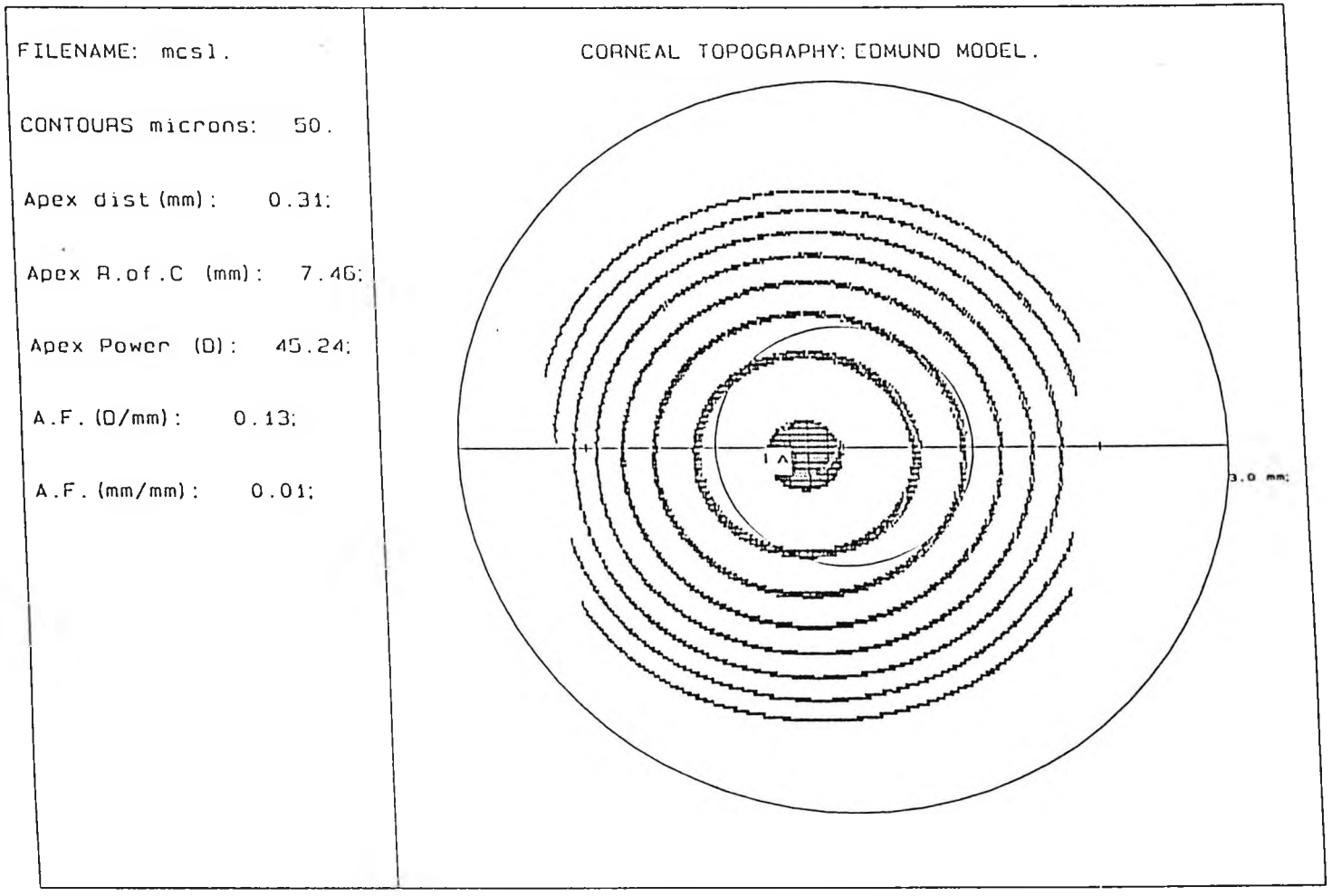
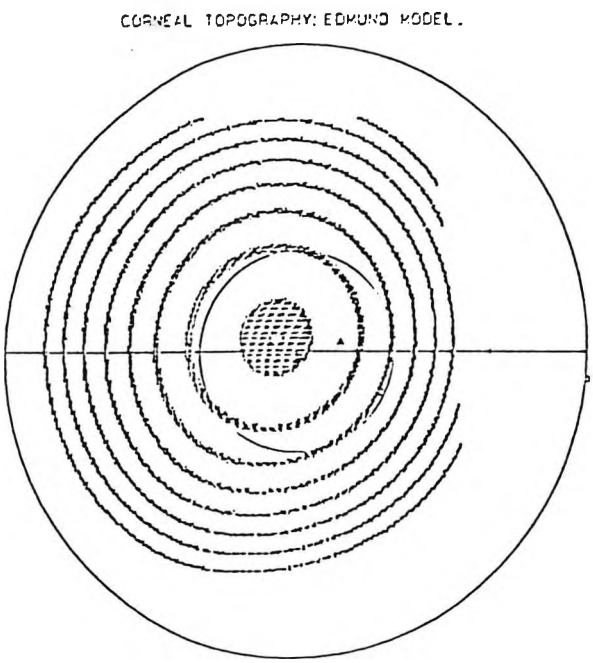


fig 6.5 Corneal plot without radius of curvature values

The two figures below show results for a relatively flat and steep cornea respectively. Fig. 6.6 shows an eye at the flat end of the range with apex radius of curvature 8.06mm. Fig 6.7 shows an eye at the steep end of the normal range with a radius of curvature of 7.03mm.



FILENAME: dadr1.

CONTOURS microns: 50.

Apex dist (mm): 0.13;

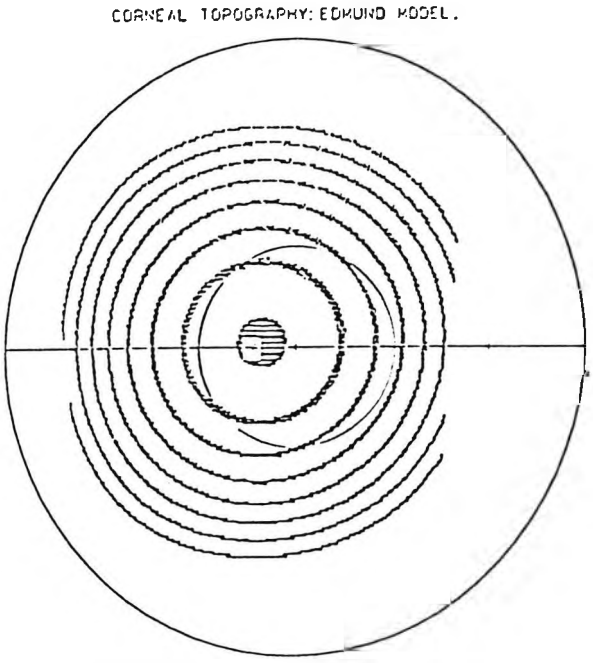
Apex R.of.C (mm): 8.06;

Apex Power (D): 41.87;

A.F. (D/mm): 0.01;

A.F. (mm/mm): 0.00;

fig 6.6 Plot of flat cornea



FILENAME: js1.

CONTOURS microns: 50.

Apex dist (mm): 0.27;

Apex R.of.C (mm): 7.03;

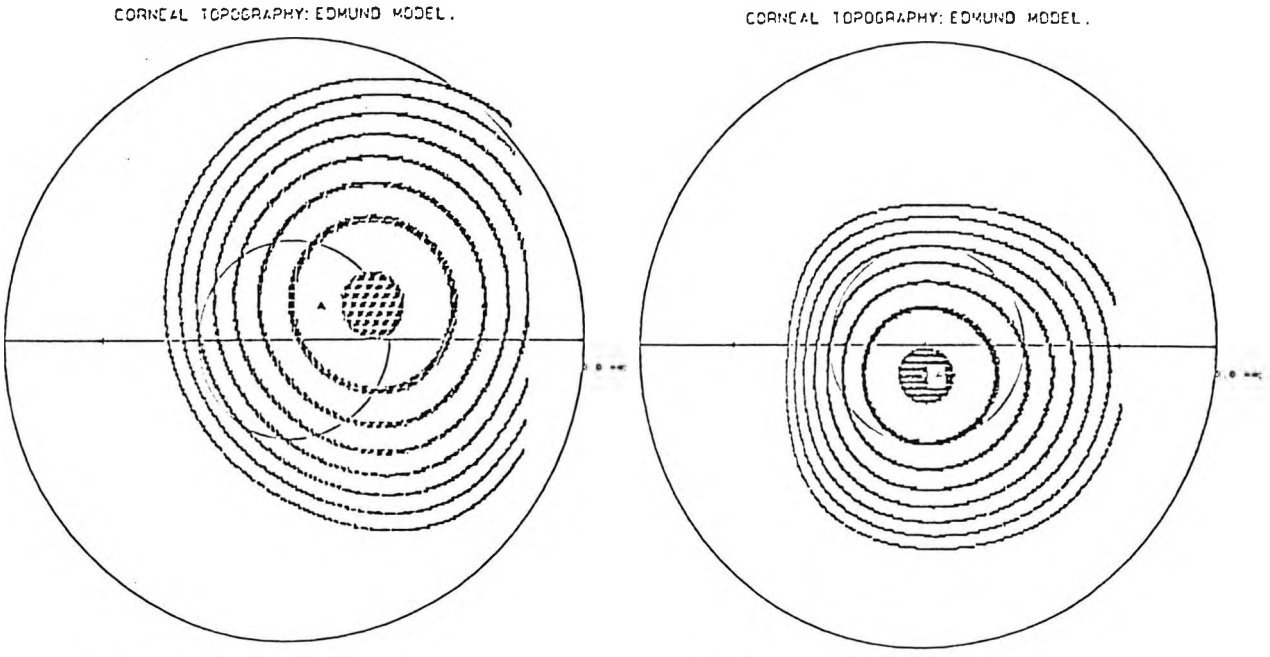
Apex Power (D): 48.00;

A.F. (D/mm): 0.20;

A.F. (mm/mm): 0.02;

fig 6.7 Plot of steep cornea

Below are typical results for an astigmatic eye (fig 6.8) and a keratoconic eye (fig. 6.9). The keratoconic eye shows a much smaller than normal apex radius of curvature of 5.13mm. The rate of flattening from the apex of 0.23mm/mm is far from spherical and the apex position is shifted downwards.



FILENAME: nhr2.

CONTOURS microns: 50.

Apex dist (mm): 0.38;

Apex R.of.C (mm): 7.46;

Apex Power (D): 45.24;

A.F. (D/mm): 0.16;

A.F. (mm/mm): 0.02;

fig 6.8 Astigmatic cornea

FILENAME: wil4.

CONTOURS microns: 50.

Apex dist (mm): 0.52;

Apex R.of.C (mm): 5.13;

Apex Power (D): 65.76;

A.F. (D/mm): 4.16;

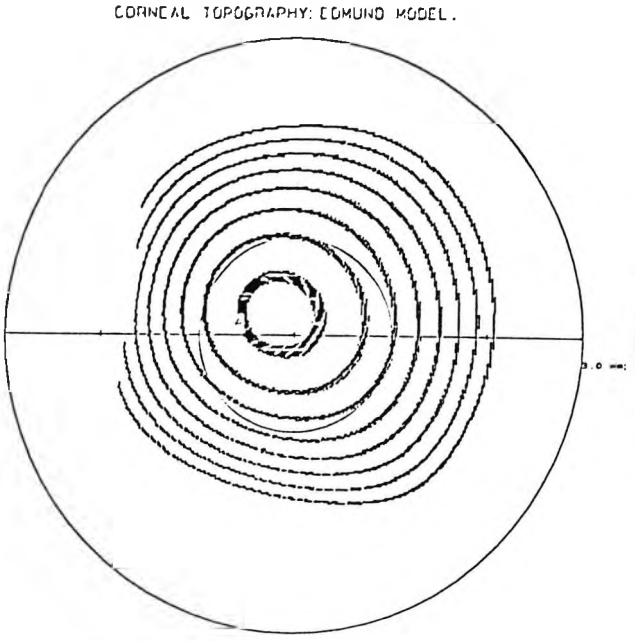
A.F. (mm/mm): 0.23;

fig 6.9 Keratoconic cornea

6.3 Results from subjects with Keratoconus.

Previous systems for measuring corneal topography have been unable to analyse accurately irregular shaped corneas as in the case of Keratoconus. The measurement system described in this thesis overcomes problems faced by previous instruments and in this section, results from three keratoconic subjects are presented.

The first subject MP has been diagnosed as having advanced Keratoconus in both eyes with approximately equal severity. The left eye had an apex radius of curvature = 6.36mm (fig. 6.10). The right eye had an apex radius of curvature = 6.28mm (fig. 6.11). Both these radius of curvature values are more than 0.5mm outside the 'normal' range but are not near the steepest values measured for Keratoconus. Both eyes had considerable scarring and a scar map for the left eye is shown in fig 6.12. The system used to measure the corneal scarring is described in chapter 7.



FILENAME: mat9.

CONTOURS microns: 50.

Apex dist (mm): 0.36;

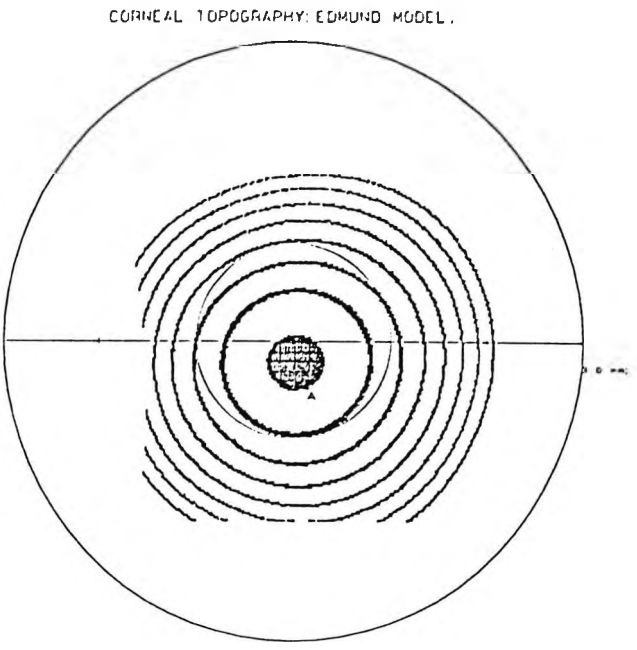
Apex R.of.C (mm): 6.36;

Apex Power (D): 53.11;

A.F. (D/mm): 0.92;

A.F. (mm/mm): 0.07;

fig 6.10 Advanced Keratoconus (left eye)



FILENAME: mat3.

CONTOURS microns: 50.

Apex dist (mm): 0.59;

Apex R.of.C (mm): 6.28;

Apex Power (D): 53.72;

A.F. (D/mm): 0.19;

A.F. (mm/mm): 0.01;

fig 6.11 Advanced Keratoconus (right eye)

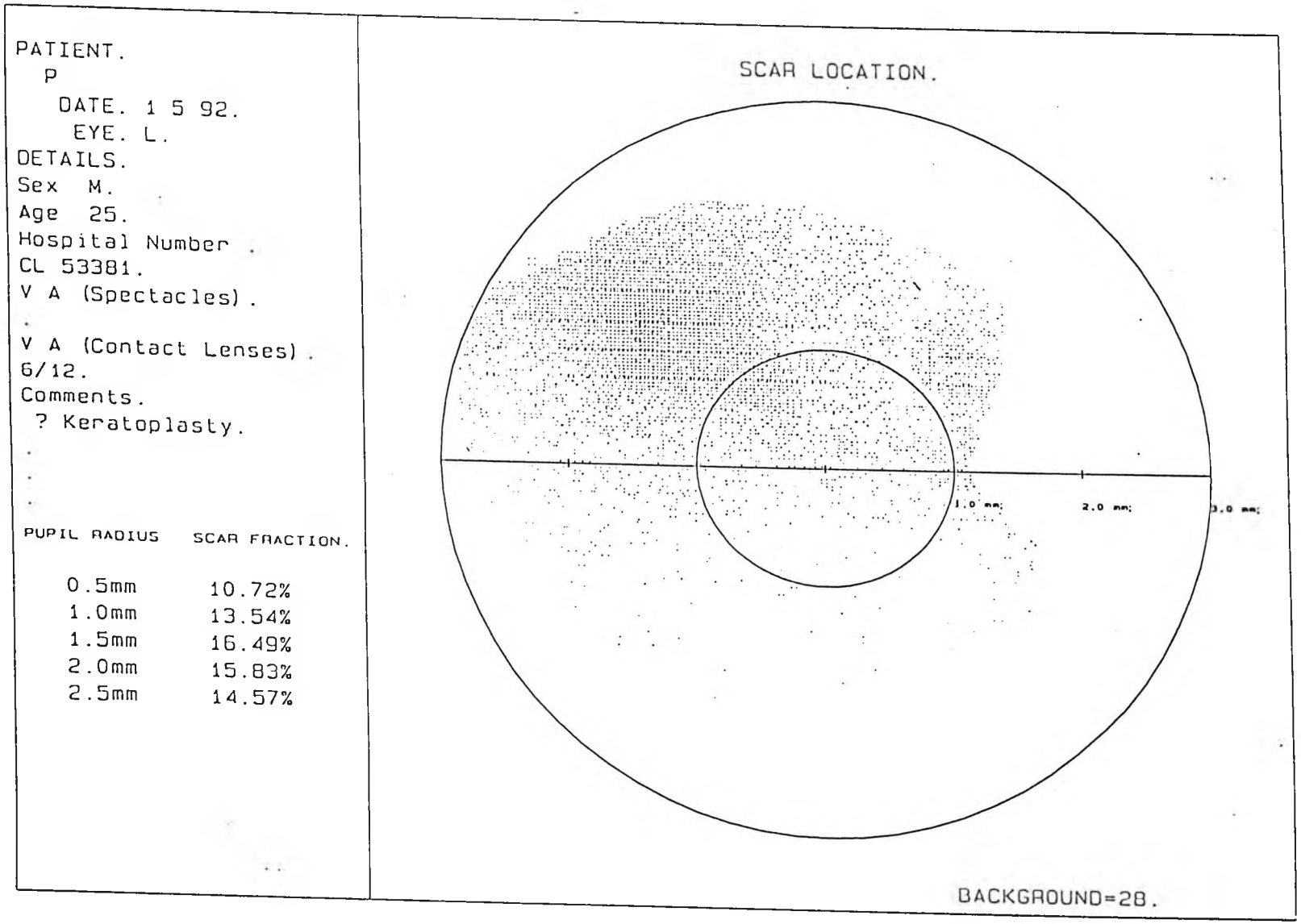
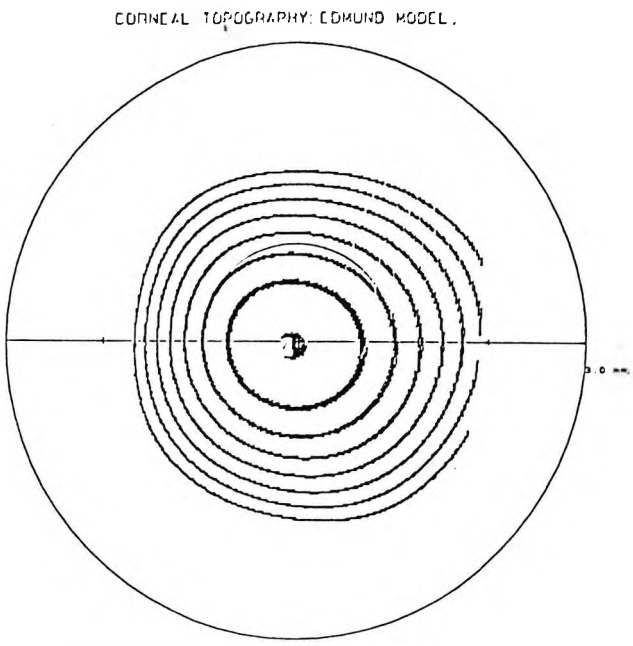


fig 6.12 Scar map of advanced Keratoconus

The second subject JMH has severe Keratoconus in the right eye (fig 6.13) and has sub-clinical Keratoconus in the left eye (fig 6.14). The left eye shows an apex radius of curvature of 7.41 and a rate of flattening from the apex (A.F.) of 0.03 mm/mm which is effectively spherical. No scarring was observed in the left eye but corneal thinning was present together with increased visibility of the corneal nerves. The right eye (fig 6.13) shows severe Keratoconus with an apex radius of curvature of 5.50mm and a rate of flattening from the apex of 0.17 mm/mm which is far from spherical. A contrast sensitivity measurement of both eyes was taken (fig 6.15) and shows the relative loss of vision in the right eye.



FILENAME: jmhr2.

CONTOURS microns: 50.

Apex dist (mm): 0.06;

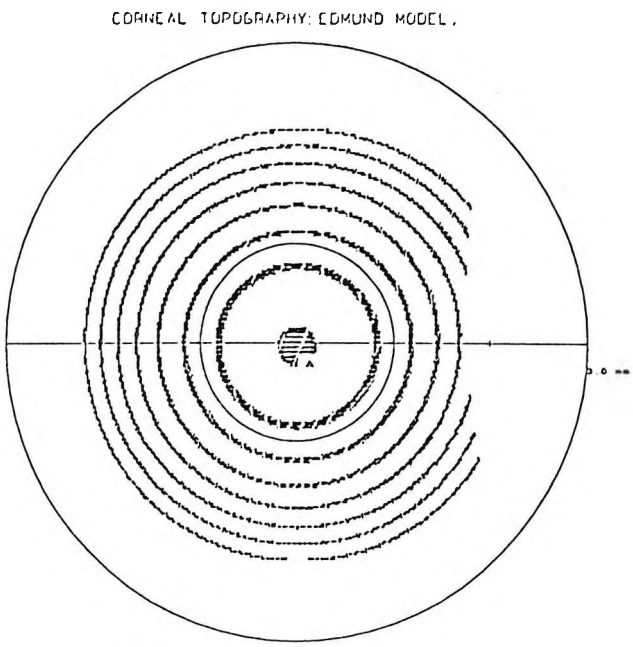
Apex R.of.C (mm): 5.50;

Apex Power (D): 61.38;

A.F. (D/mm): 2.69;

A.F. (mm/mm): 0.17;

fig 6.13 Severe Keratoconus



FILENAME: jmh12.

CONTOURS microns: 50.

Apex dist (mm): 0.25;

Apex R.of.C (mm): 7.41;

Apex Power (D): 45.55;

A.F. (D/mm): 0.27;

A.F. (mm/mm): 0.03;

fig 6.14 Sub-clinical Keratoconus

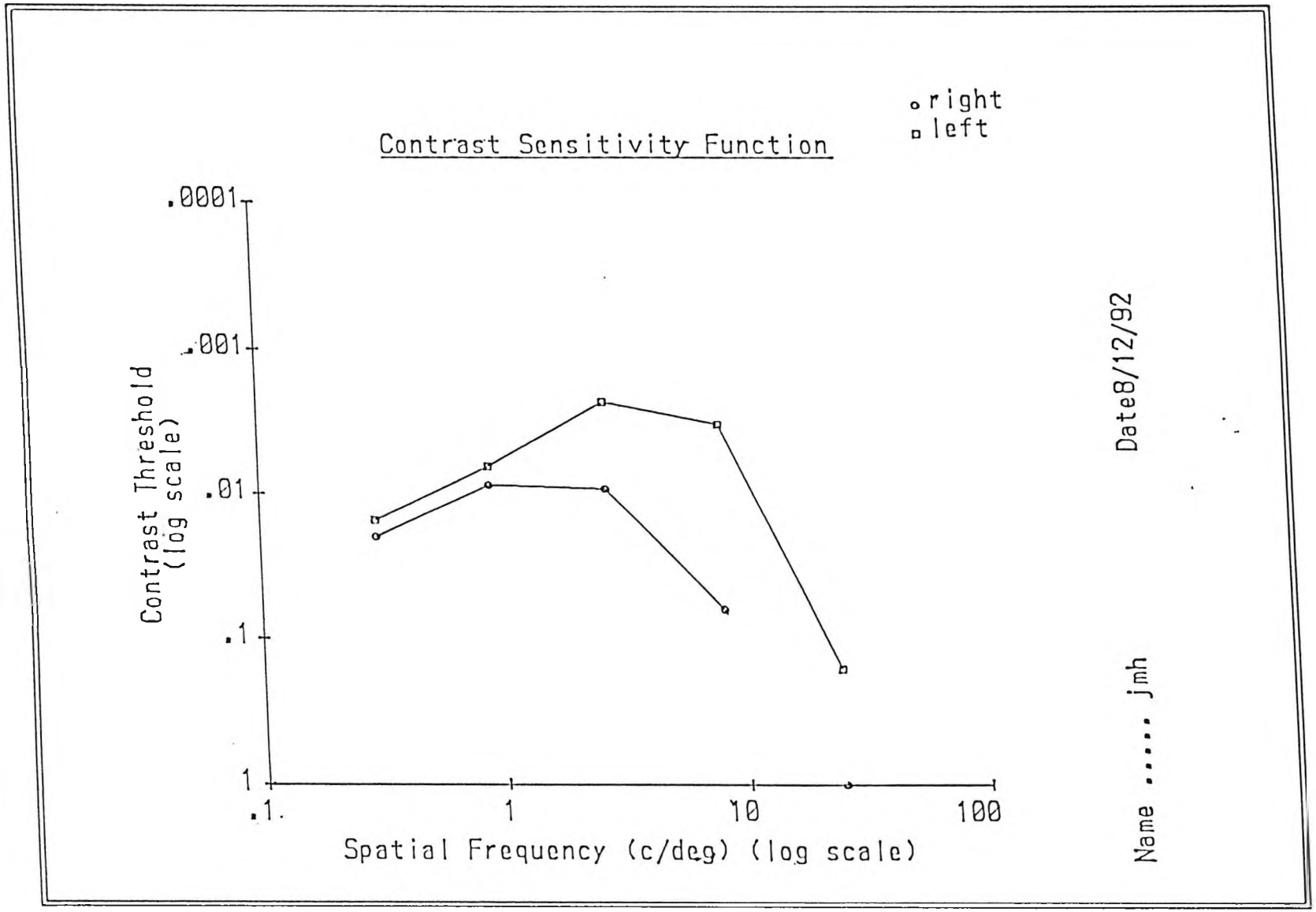
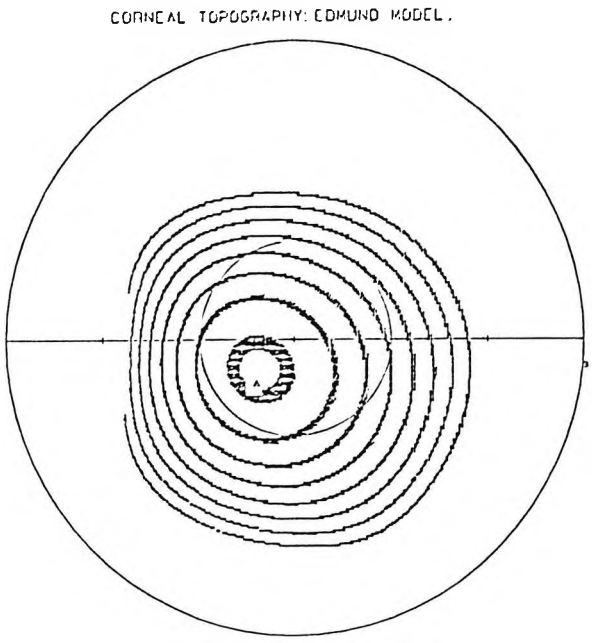


fig 6.15 Contrast sensitivity results for subject JMH

The third subject CH has severe Keratoconus in the left eye (fig 6.16) and a corneal graft in the right eye a year previously (fig 6.17). The left eye shows a very steep cornea of apex radius of curvature = 5.40mm and flattening from apex = 0.20 mm/mm. A map of the scarring in the left eye is shown in fig 6.18. The subject indicated that there was an overall post-operative improvement in vision in the right eye, except in viewing very small objects. This is confirmed by the contrast sensitivity measurement (Kelly, 1977) of each eye shown in fig. 6.19.



FILENAME: chick1.

CONTOURS microns: 50.

Apex dist (mm): 0.50;

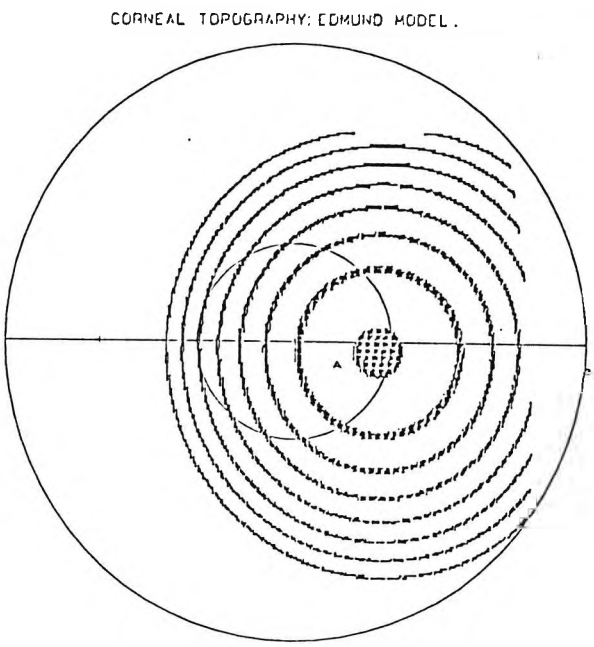
Apex R.of.C (mm): 5.40;

Apex Power (D): 62.50;

A.F. (D/mm): 3.34;

A.F. (mm/mm): 0.20;

fig 6.16 Severe Keratoconus



FILENAME: chick.

CONTOURS microns: 50.

Apex dist (mm): 0.53;

Apex R.of.C (mm): 7.64;

Apex Power (D): 44.16;

A.F. (D/mm): 0.33;

A.F. (mm/mm): 0.04;

fig 6.17 Corneal graft

PATIENT.
H
DATE. 3 8 92.
EYE. L.
DETAILS.
Sex F.
Age 25.
Hospital Number .
M 89809.
V A (Spectacles) .
.
V A (Contact Lenses) .
.
Comments.
R graft Aug 91.
.
.
.

PUPIL RADIUS	SCAR FRACTION.
0.5mm	7.71%
1.0mm	9.70%
1.5mm	12.12%
2.0mm	8.90%
2.5mm	6.10%

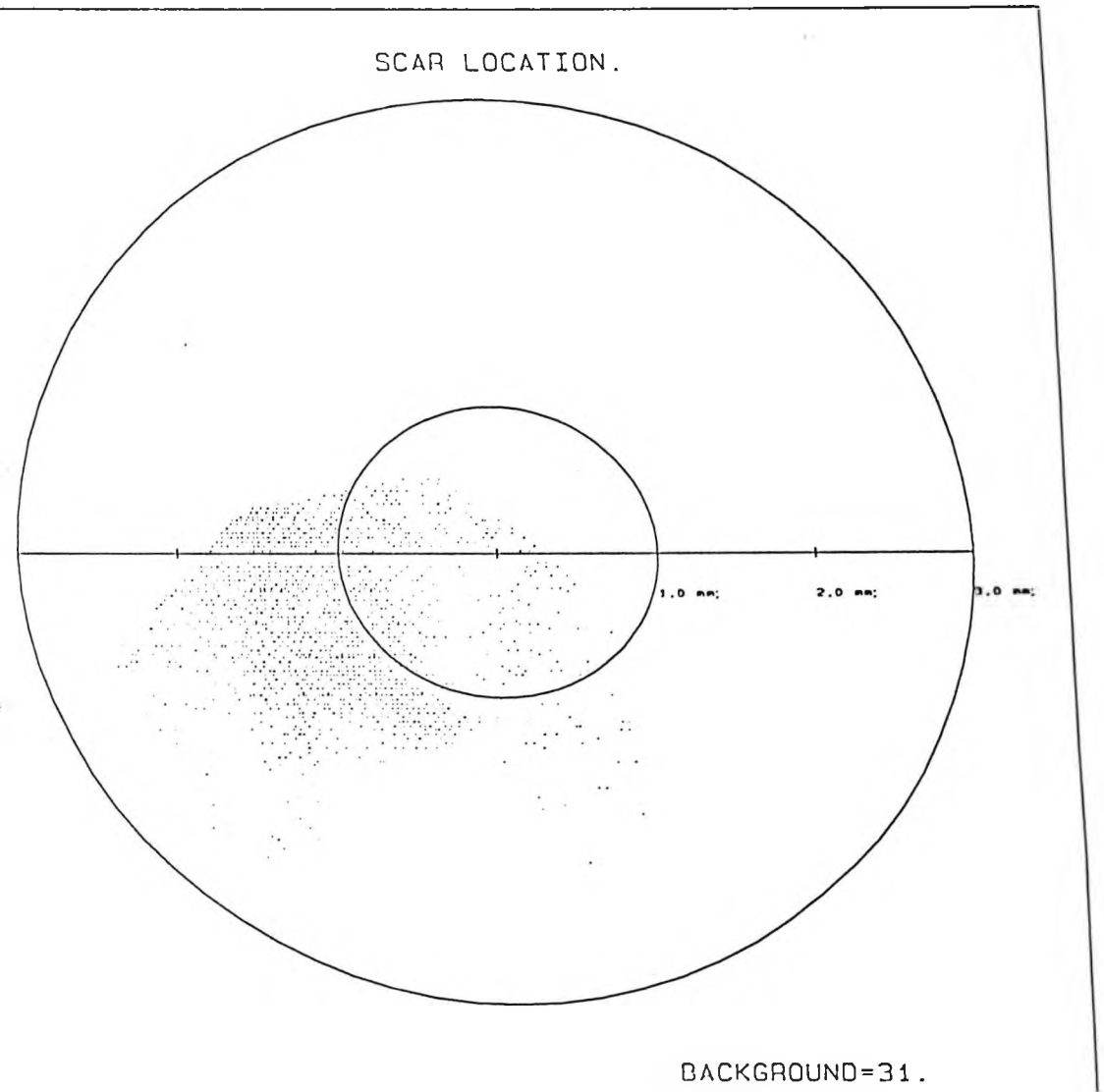


fig 6.18 Severe Keratoconus (left eye) scarring for CH

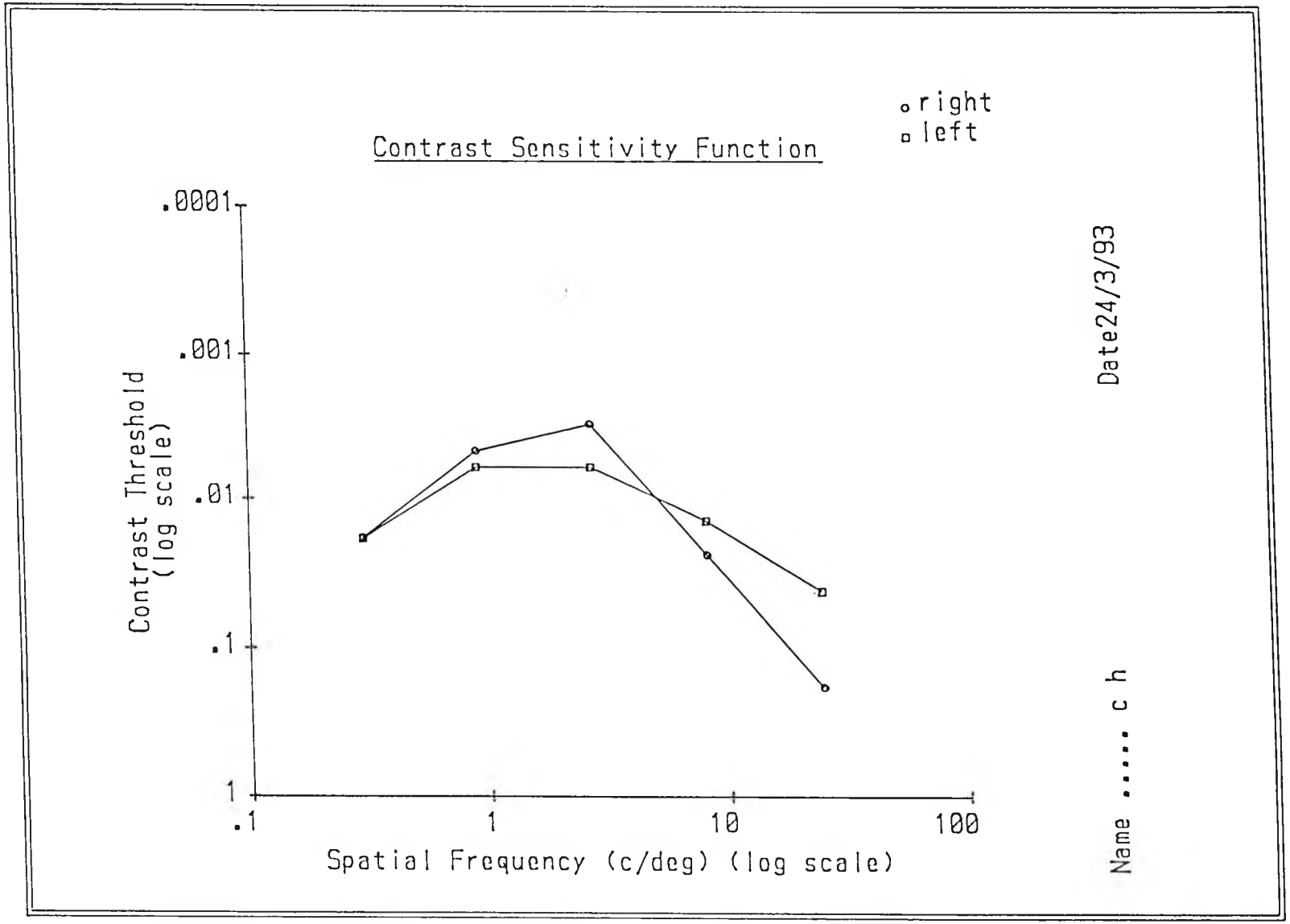


fig 6.19 Contrast sensitivity results for subject CH

6.4. Comparison of corneal topography measurements using CTS and using a photokeratoscope for normal corneas.

The measurement and analysis method employed by the CTS is, in theory, superior to the methods employed by keratoscopes especially when assessing irregular corneas.

In practice, the CTS has been shown to produce accurate measurements on calibrated spheres, but it was of interest to know how the system performed on various corneal shapes and how the CTS results compared with the results of a conventional keratoscope.

Measurements of 30 normal corneas were made with the CTS and also with the Wesley-Jessen PEK. This was chosen on the basis that the PEK is a well documented and established keratoscope. To enable comparisons between the results of each instrument, equivalent shape measurements concisely describing the topography must be extracted from each set of results. This was possible using the analysis method for PEK results developed by Edmund(1987) which described the corneal shape in terms of a)the apex location relative to the visual axis, b)the rate of flattening from the apex in a given meridian and c)the area of the spherical zone around the apex. These parameters were modified slightly to enable concise statistical analysis and compatibility with the CTS. The final form of the extracted parameters gives the apex location relative to the centre of the cornea, the average rate of flattening from the apex and the value of curvature at the apex.

6.5 Shape measurement results.

Shown below in Table 6.6 are the mean shape measurement results from 30 normal corneas, taken with the Corneal Topography System(CTS) and the Wesley-Jessen PEK. The raw data for all shape and measurements are given in appendix B.

Scatter diagrams of the apex radius of curvature, apex decentration and average rate of flattening, obtained by the two systems are shown in figs. 6.20, 6.22 and 6.23 respectively.

variable	mean value	standard deviation	range
CTS apex radius of curvature. mm.	7.82	0.32	7.03-8.35
PEK apex radius of curvature. mm.	8.04	0.29	7.37-8.51
CTS apex distance from corneal centre. mm.	0.48	0.28	0.05-1.23
PEK apex distance from corneal centre. mm.	0.27	0.16	0.02-0.68
CTS rate of flattening from apex. mm/mm.	0.04	0.03	0.00-0.11
PEK rate of flattening from apex. mm/mm	0.01	0.02	0.00-0.13

Table T. 6.6. Shape measurement results.

Fig. 6.20 shows a scatter diagram of Apex radius of curvature results for PEK and CTS. The results show generally lower radius of curvature values for CTS relative to PEK and also poor correlation. Previous photokeratoscope measurements of mean central corneal radius of curvature for 164 normal corneas were made by Clark (1974).

The results showed an average value of 7.75±0.26 mm compared to 7.82±0.32 mm for CTS.

In general, there was a *fair* correlation between the apex radius of curvature measured by the two systems ($R^2 = 0.61$) although the CTS tended to produce lower values than the PEK. To investigate if the differences between the two instruments varied systematically with the radius of curvature, the difference in CTS and PEK radius of curvature measurements were plotted against the mean of the measurements (fig. 6.21). No obvious trend was apparent in this respect.

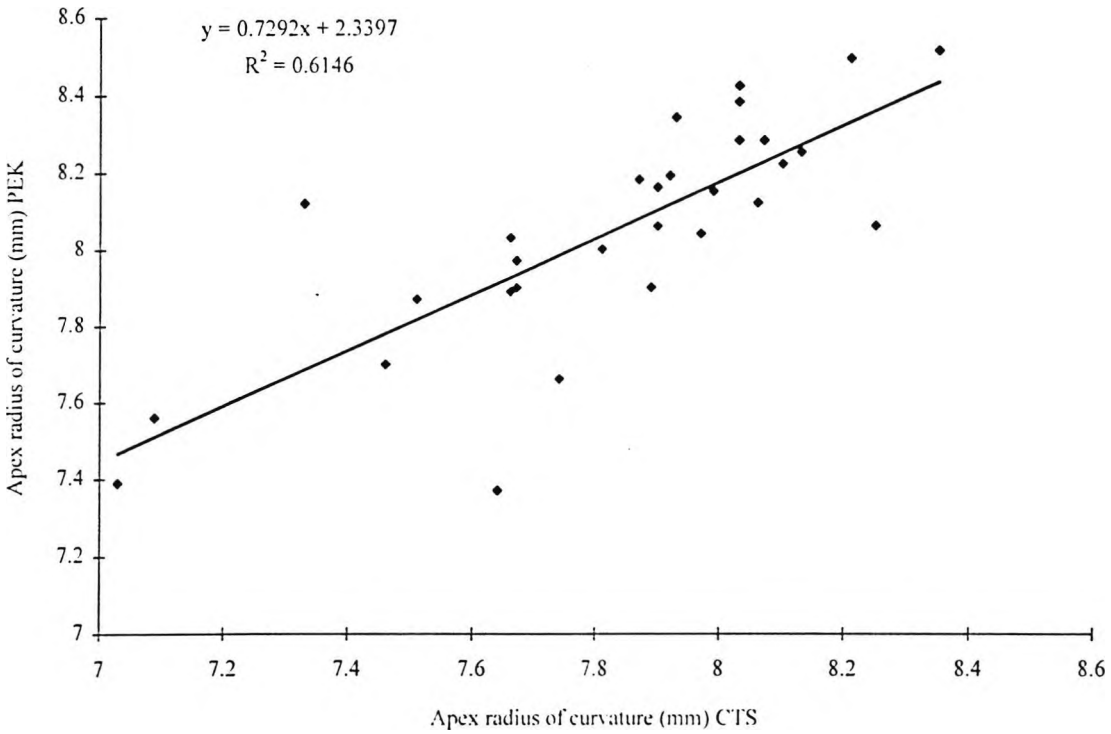


fig. 6.20. Comparison of apex radius of curvature measured by CTS and PEK systems.

(R = correlation coefficient)

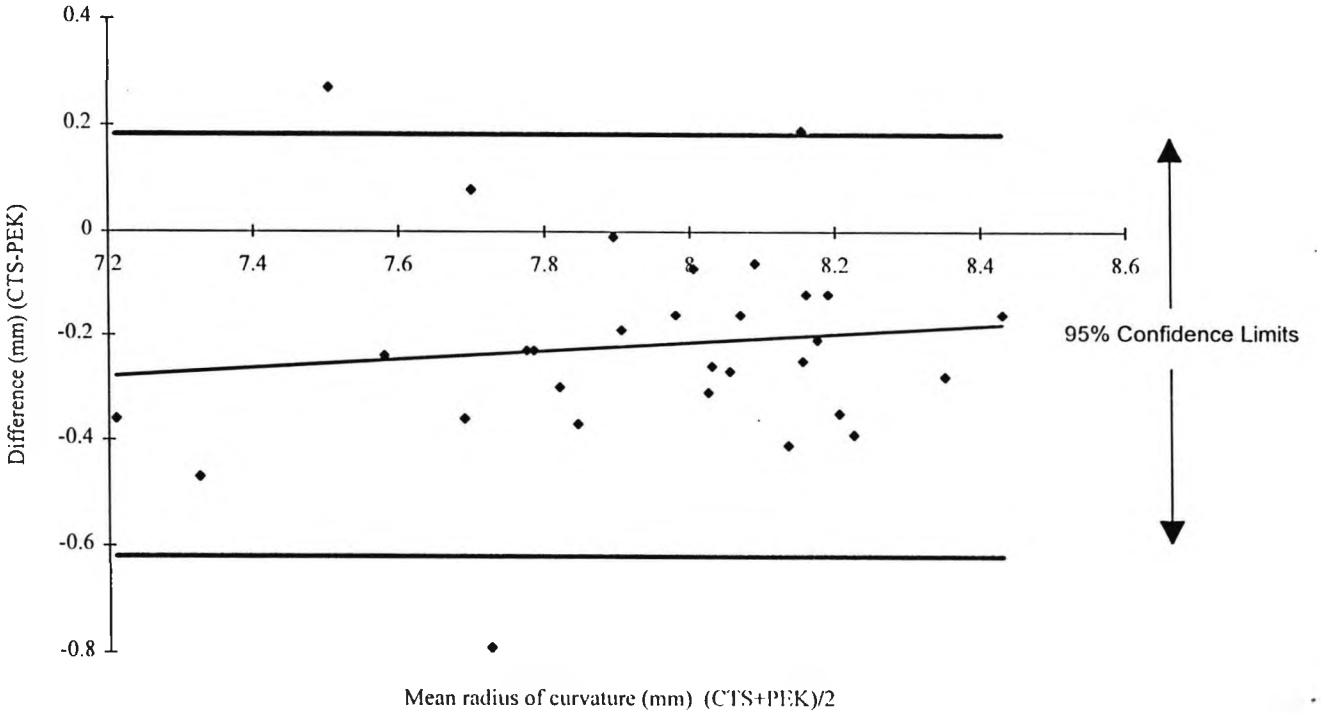


fig. 6.21 Difference between apex radius of curvature for CTS and PEK systems against mean radius of curvature.

Measurements of the decentration of the apex from the geometrical centre of the cornea showed very little correlation for the two instruments (fig. 6.22). This was not entirely unexpected in view of the fact that the PEK is not inherently designed to identify the corneal apex. However, Edmund (1987) has previously studied apex displacement from PEK measurements and found an average apex displacement of 0.4mm from the centre of the cornea. This value is found from Edmund's value of displacement from the visual axis by assuming the optical and visual axes are displaced by 5°. The value found by CTS is 0.48±0.28mm.

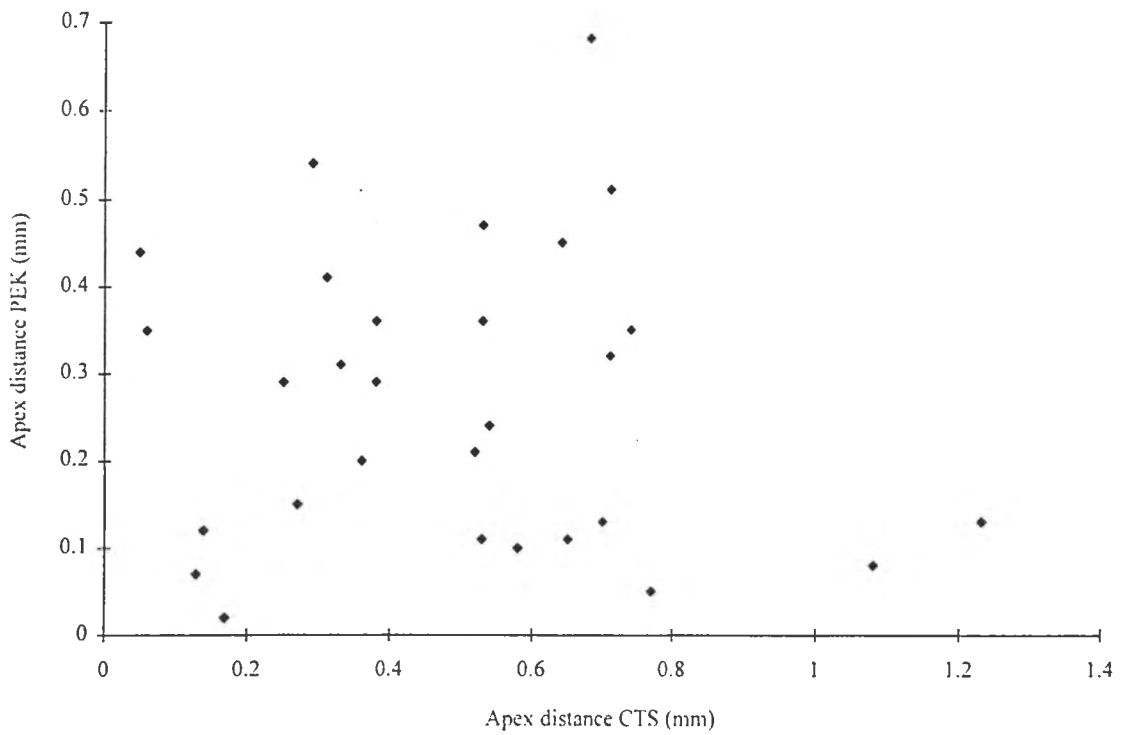


fig 6.22. Distance of apex from geometrical centre of cornea.

Measurements of the average rate of flattening from the apex also showed very little correlation for the CTS and PEK (fig. 6.23). This is probably attributable to the fact that the PEK is unable to produce reliable measurements in this respect because the mire images are formed from reflections from the whole corneal surface and radius calculations assume spherical surfaces.

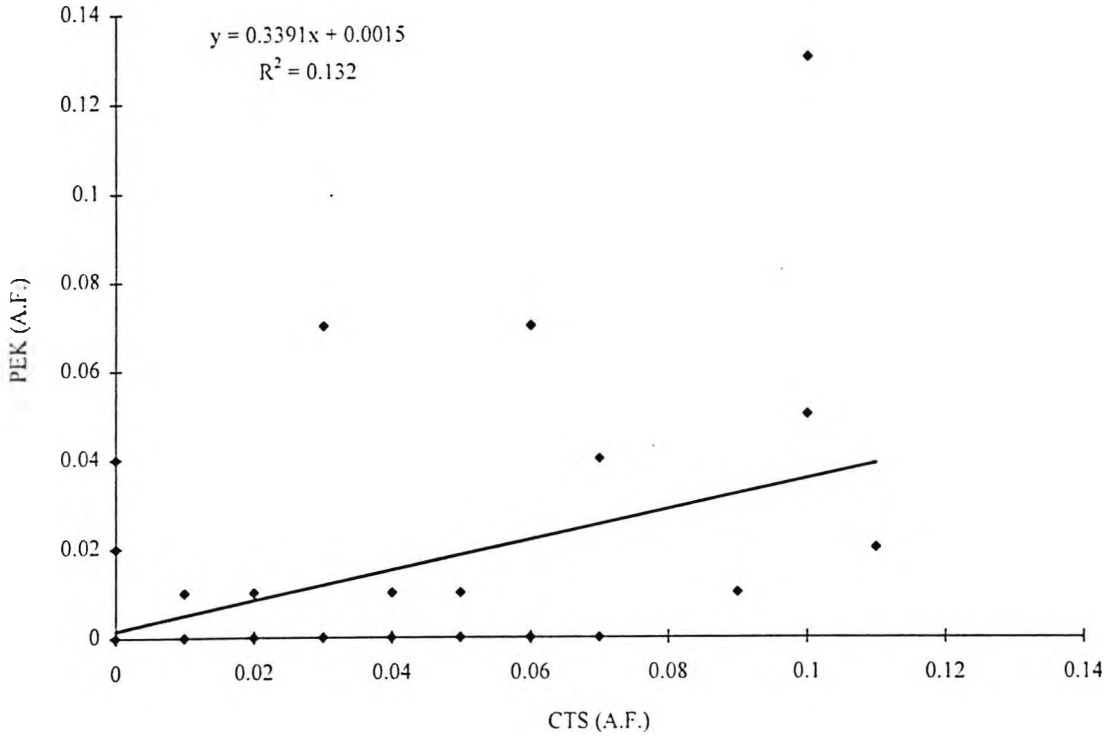


fig. 6.23 Average rate of flattening from apex (mm/mm)

In summary, the CTS and PEK produce similar values for the apex radius of curvature for normal corneas. Estimates of the decentration of the apex and the average rate of flattening are not well correlated for the two instruments.

Chapter 7
System for the measurement of corneal transparency and scarring.

7.1 Introduction.

Optical clarity is a fundamental property of a healthy cornea. However, even in the normal cornea a small amount of light scatter occurs which illuminates the retina away from the image of the source. This scattering has the effect of reducing the contrast of the retinal image thus reducing contrast sensitivity and visual function(Barbur, 1993). When the cornea is subject to disease, trauma or surgical intervention, any resultant scarring causes further light scatter which can severely effect vision especially if the scarring is over the pupil area.

The form of scars varies from localised dense opacity to faint haze covering large areas of the cornea. In corneas which have undergone PRK, a faint haze often occurs covering the ablated area and must be treated to restore good quality vision. In diseased corneas such as with Keratoconus, both dense scarring and faint haze can occur and necessitates Penetrating Keratoplasty as the only effective treatment.

Efforts have been made to quantify scarring (Smith, 1990; Olsen, 1982) and a scar measurement system (Lohmann, 1991) has been used to measure the ratios of scattered light for treated and untreated corneal areas in PRK patients. Results from the Lohmann (1991) system showed a correlation between backscatter light levels and 5% contrast visual acuity measurements for PRK treated corneas.

At present there is no system available to objectively measure and map the area and density of scars. In this chapter, a system is described which objectively measures and maps corneal scarring. The area and density of scars were measured in 12 keratoconic eyes and results recorded on a map of the cornea showing the area and location of the

scarring with histograms of scar density. The system was used for a preliminary study of the effects of different scar densities on visual function.

7.2 Method.

In developing the system to measure corneal scarring and evaluate the density of scars, two assumptions were made. The first assumption was that in the absence of scar tissue, the cornea is perfectly transparent and the amount of light scattered from the scar is therefore a measure of its opacity. Secondly, that the diffuse reflection from the scar tissue follows approximately a Lambertian distribution.

The system consists of a Nikon FS-2 Zoom Photo Slit Lamp through which the backscatter from the scar is photographed using a black and white CCD camera of 768 x 576 pixel resolution. The camera is connected to a Pluto II frame grabber which stores the images in a PC. A photograph of the system is shown in fig. 7.1.

fig. 7.1. System for measuring corneal scarring.

To ensure correct and consistent subject fixation, two small LEDs were located above and below the slit lamp objective. When the reflections of the LEDs were located on a line vertically through the pupil centre and the scars in focus, the image of the scars was grabbed by the computer. It was essential to dilate the pupil before the image was grabbed to enable all scarring within the central 5mm of the cornea to be visible. A view of a scarred keratoconic cornea is shown in fig. 7.2.

fig. 7.2 Keratoconic cornea showing scarring

The image of the scarred cornea was displayed on a high resolution monitor after being grabbed (fig. 7.3.)

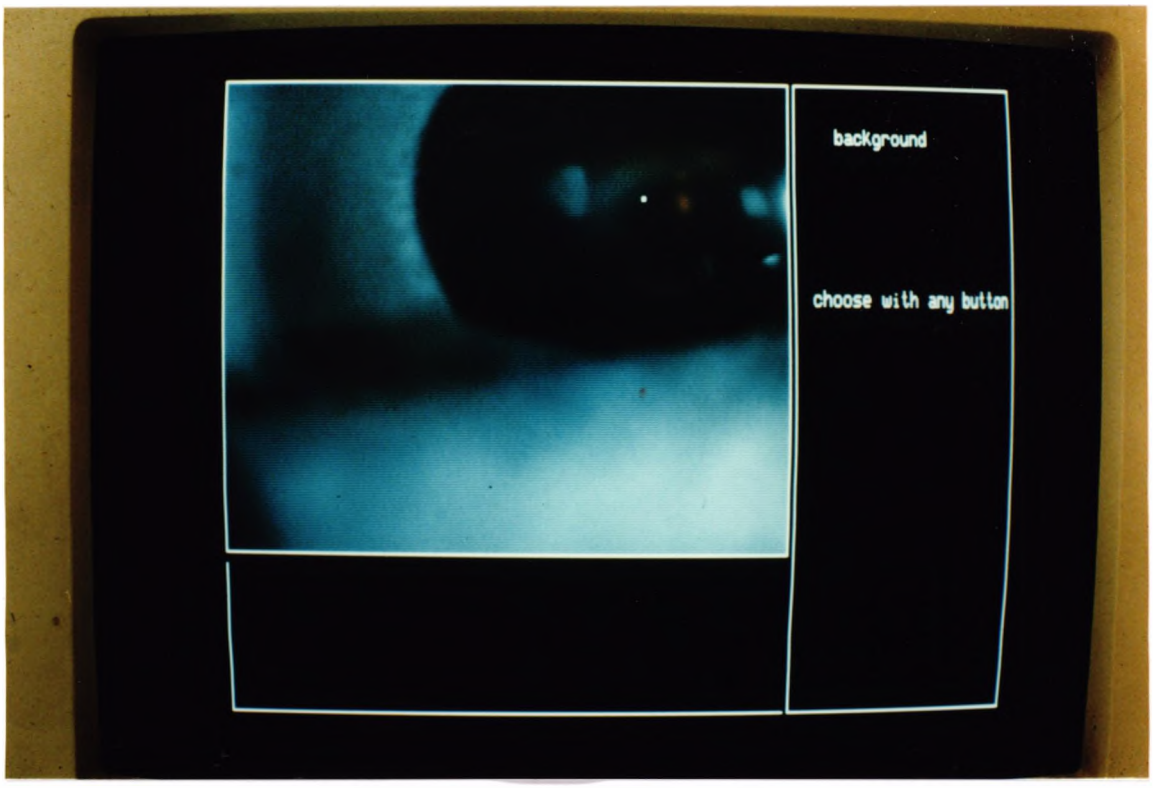


fig. 7.3. Grabbed image of scarred cornea.

The centre of the pupil was located by the computer program finding the transition points between pupil and iris in horizontal and vertical meridians. These points were taken to lie on a circle whose centre corresponded to the pupil centre(fig. 7.4).

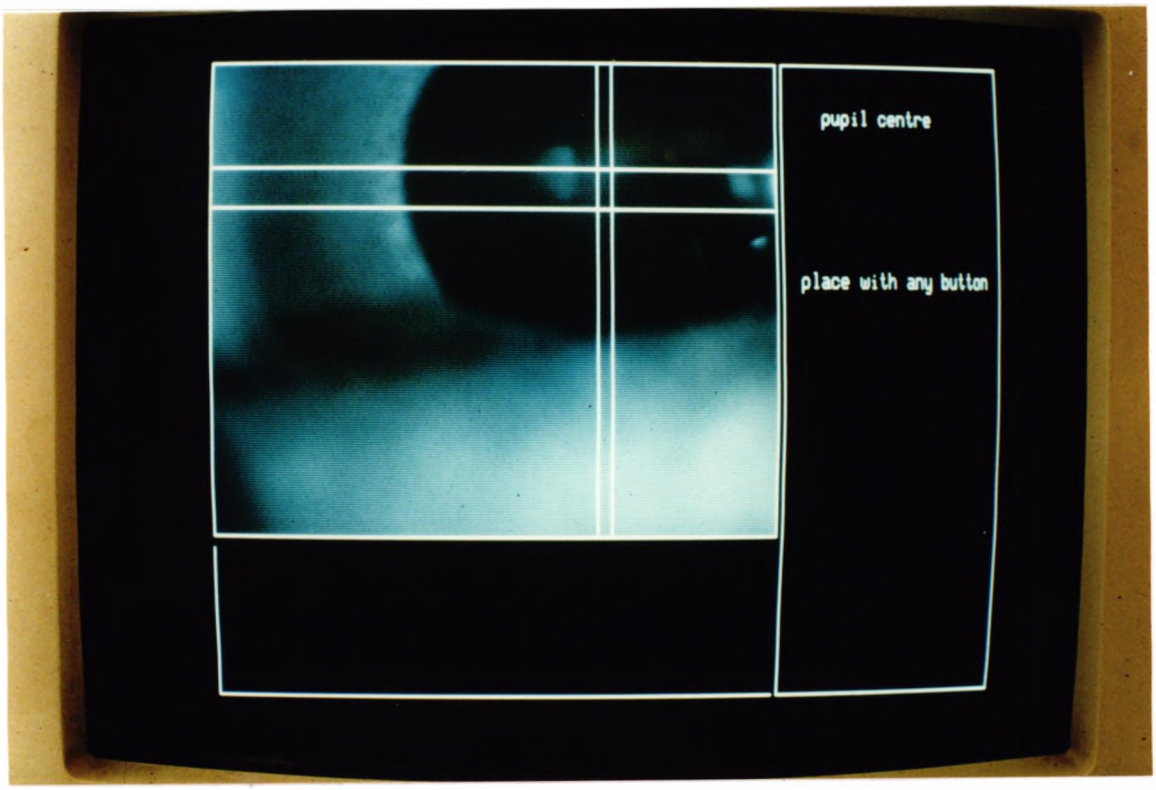


fig 7.4. Location of the pupil centre.

If the location was not correct, the operator had the option of indicating the best position of the pupil centre with the mouse.

The corneal background brightness level was chosen by positioning the mouse on a clear area of cornea. The scar is represented by higher levels of backscatter which was measured by the system. The region of scarring was then isolated from the non-scarred areas in the image by placing a border around the scarring using the mouse. This is shown in fig. 7.5. and fig. 7.6. The computer program then removes the part of the image which is not associated with scarring (fig. 7.7).

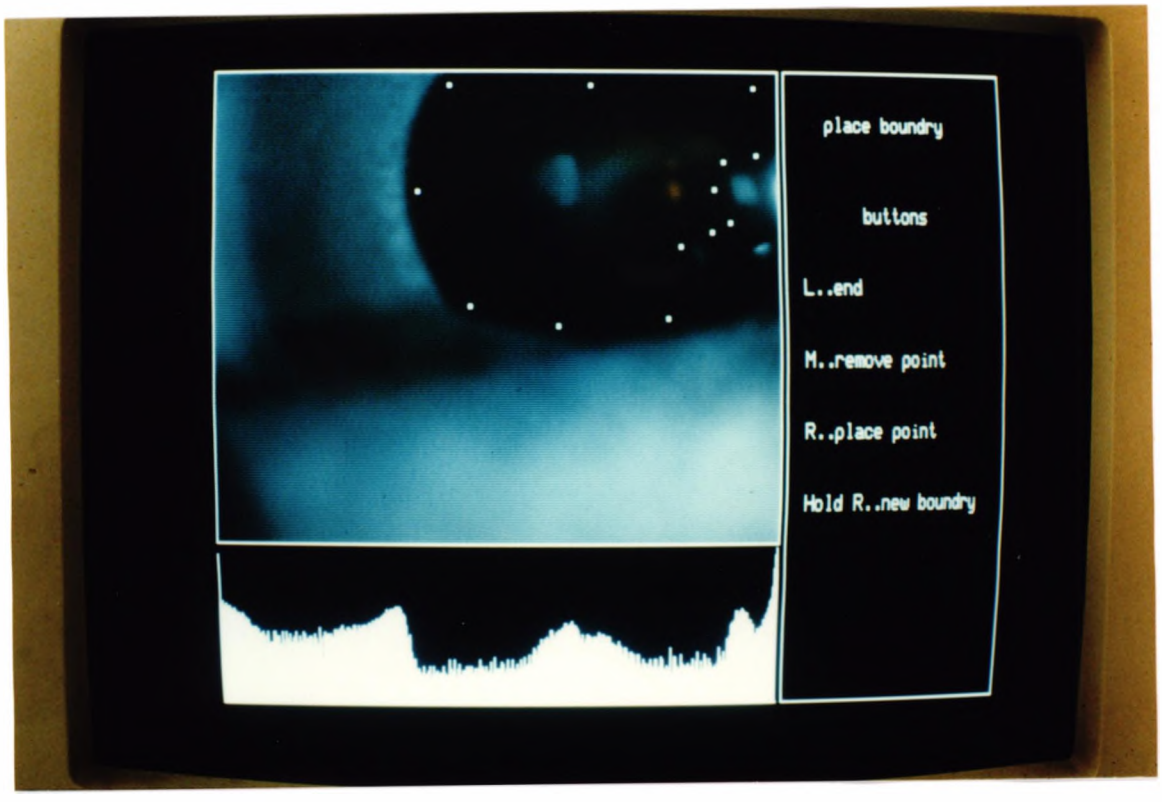


fig 7.5. Scar perimeter identified using the mouse.

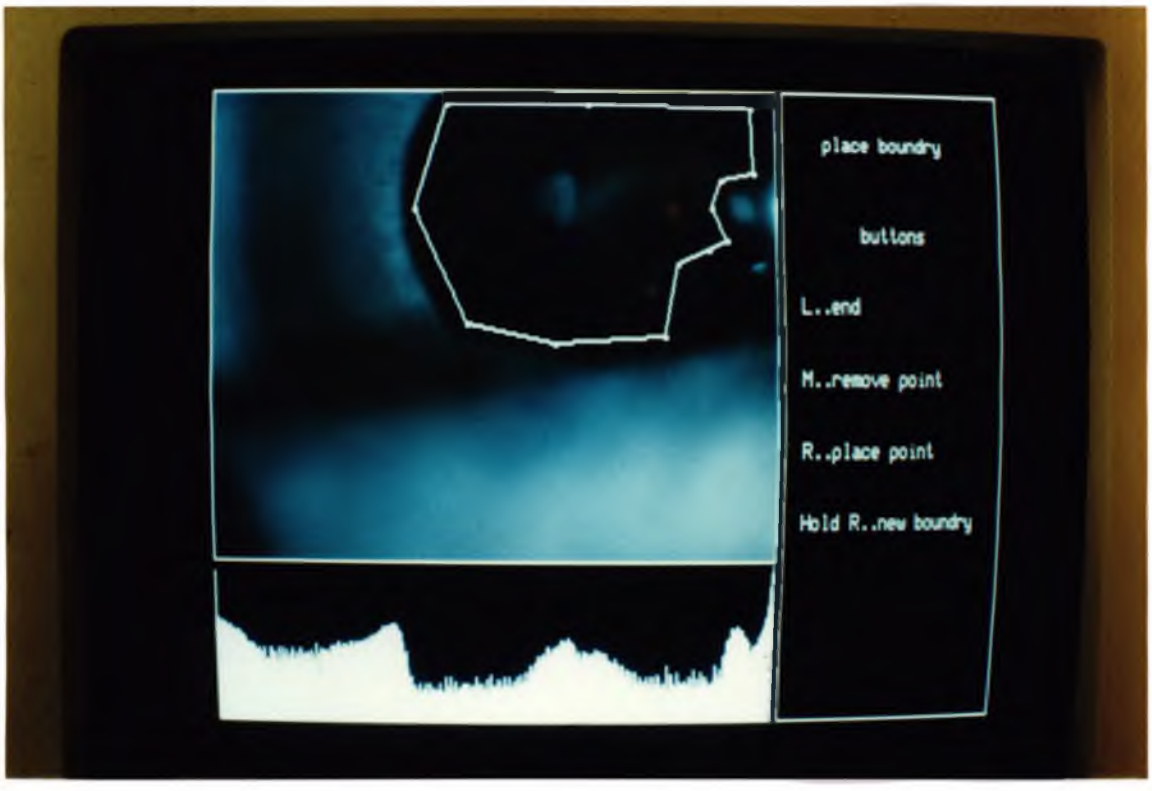


fig 7.6. Border placed around scar.

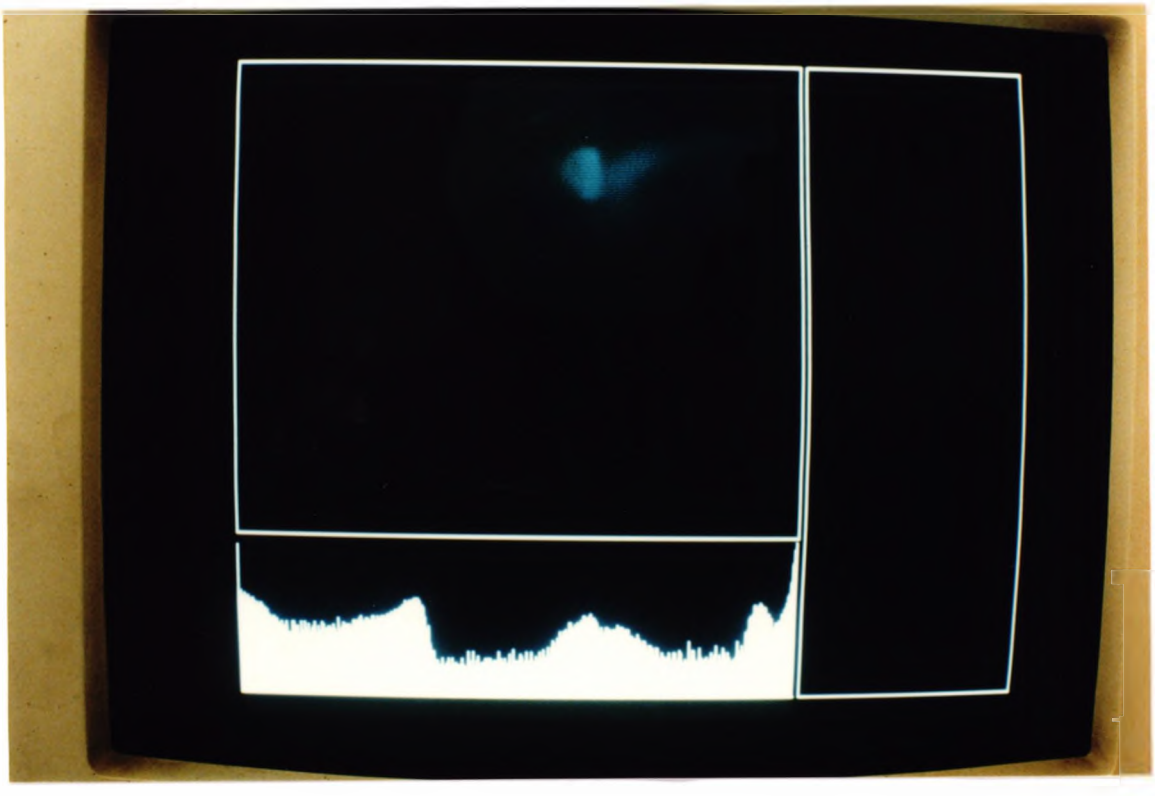


fig. 7.7. Non-scarred areas were removed from the image.

A map of the scar location relative to the pupil centre was plotted on a laser printer as shown in fig. 7.8.

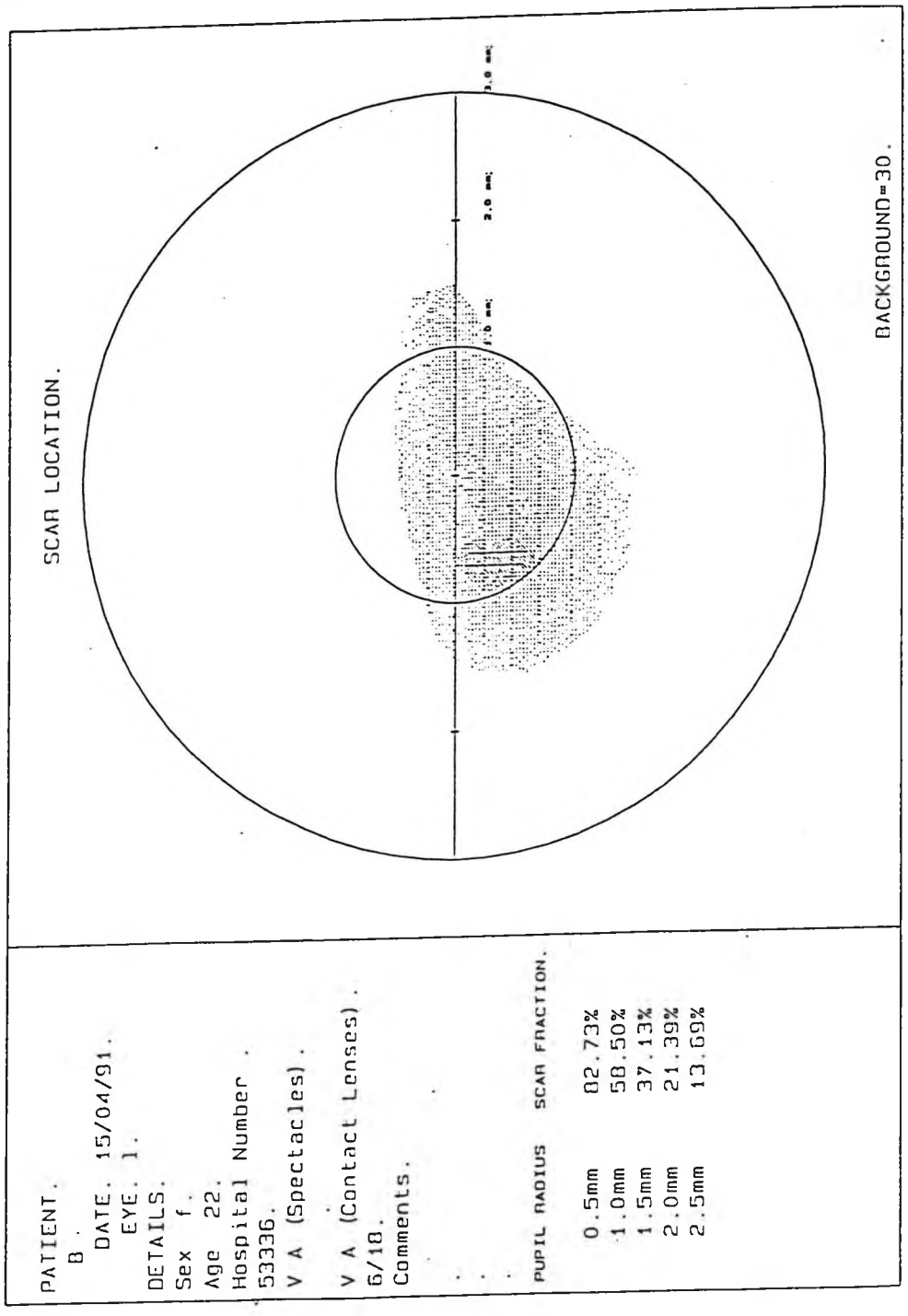


fig. 7.8. Scar map output

The 127 grey intensity levels in the scar image were further quantized into 14 levels for ease of analysis. Each quantized level corresponds to a pixel range of 9 grey levels. Normalised grey level intensity histograms of the scar overlaying pupillary areas of 0.5mm to 2.5mm in radius were printed out. The percentage of the scar overlaying different pupil sizes was also printed. A complete set of results is illustrated in fig. 7.9. and fig 7.10.

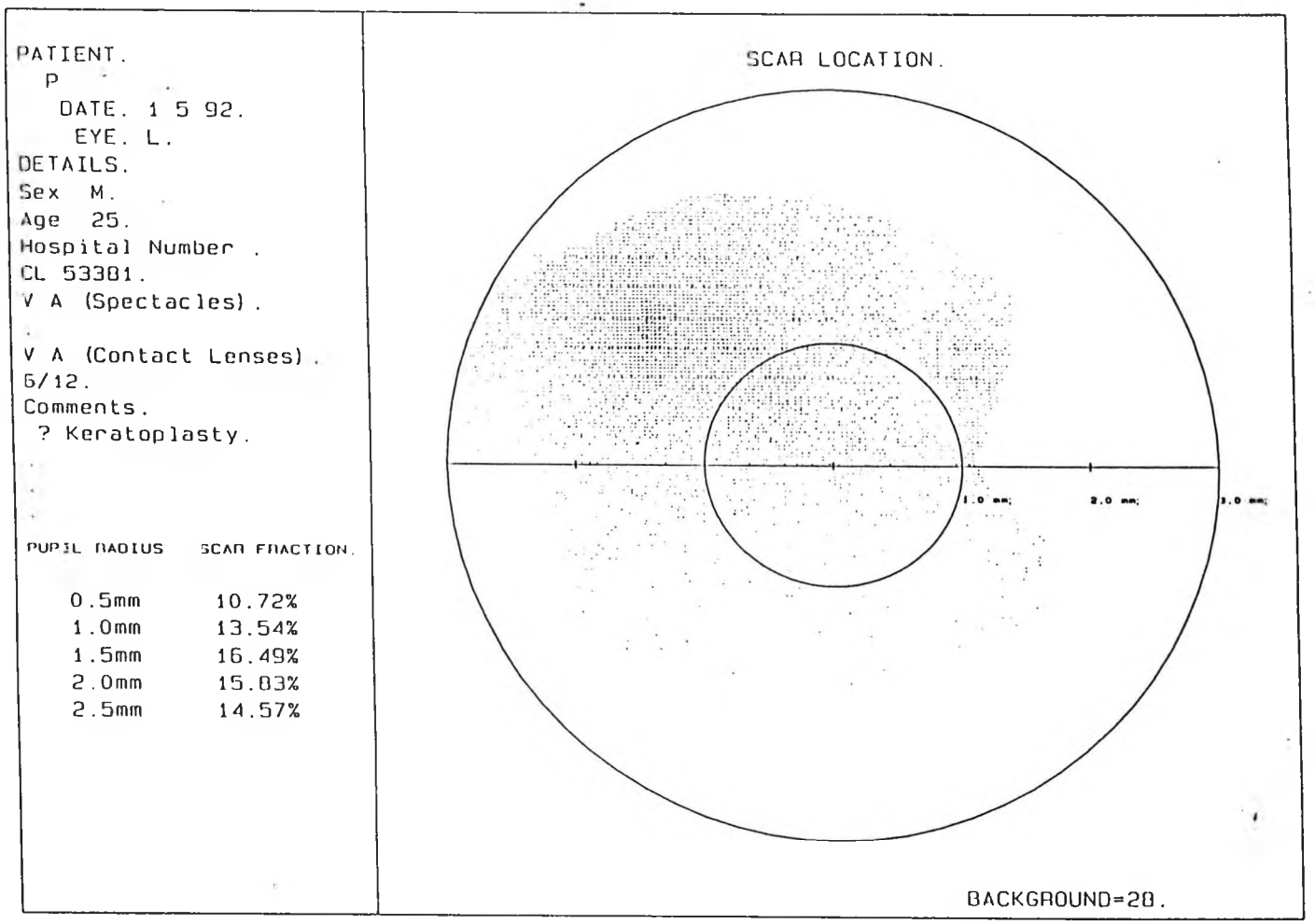


fig. 7.9. Scar location map.

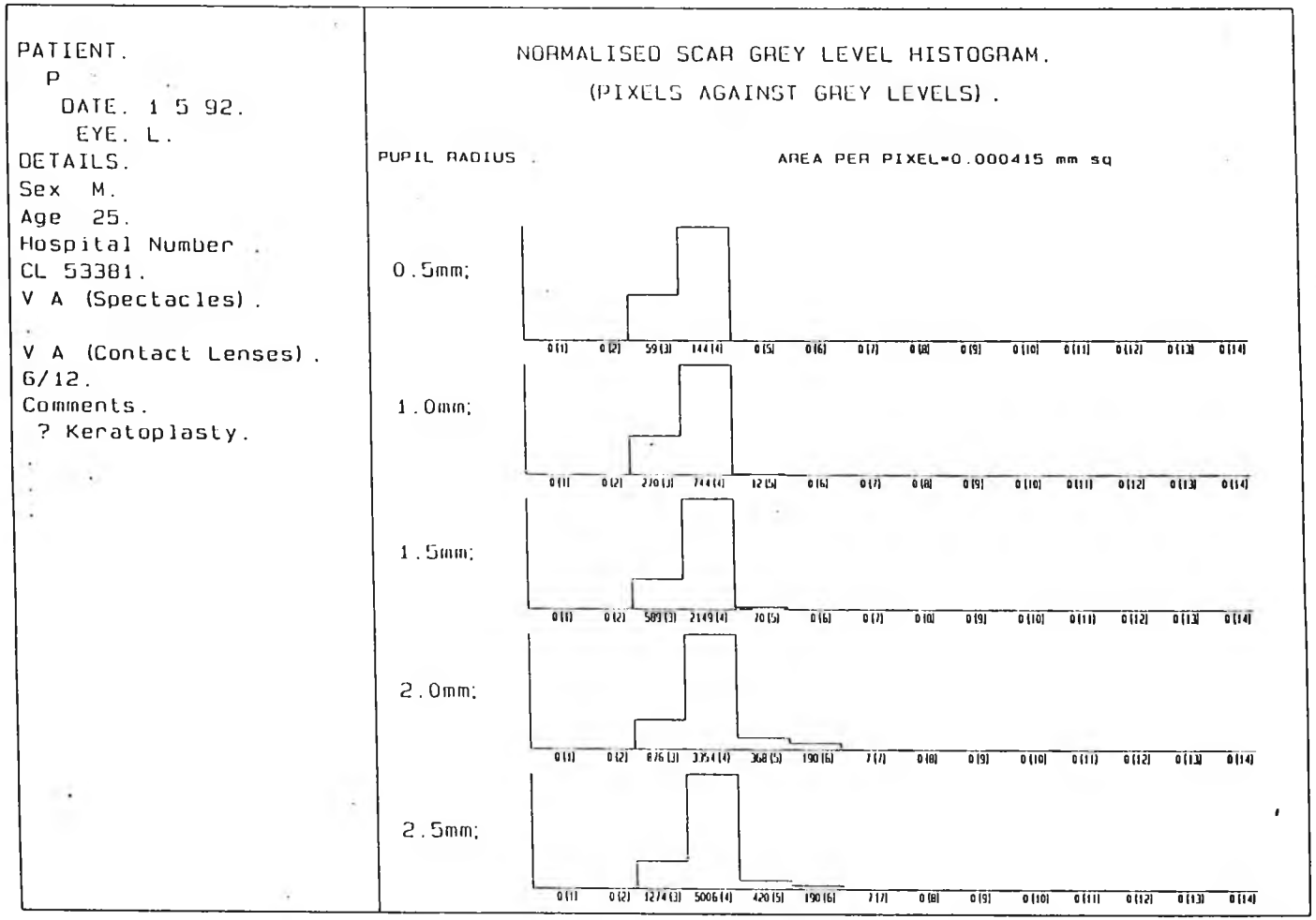


fig. 7.10. Scar grey levels.

7.3. System calibration

To convert the scar backscatter 'brightness' to luminance values, luminance was measured against recorded grey levels. A white card (Lambertian surface) was placed at the working distance of the slit lamp. The illumination of the card by the slit lamp was adjusted to correspond to the mean pixel brightness for each quantized grey level. The luminance of the card at each illumination level was measured by a photometer. The spectral response of the camera was approximately the same as the photometer response. The results are shown below in table T.7.1.

Grey level	Mean luminance cd/m^2
1	--
2	100
3	200
4	370
5	550
6	750
7	1100
8	1400
9	1750
10	2600
11	5000
12	--
13	--

Table T.7.1. Luminance measurement for different grey levels.

The value for quantized level 1 was below the background level. The values above 12 could not be achieved using the slit lamp illumination. To assess the noise in the

system, two pictures of the white card were taken at maximum illumination and the images were subtracted from each other. The resulting image left the system noise and showed that the noise value range was within ±3 pixel grey levels. This noise value was lower than the quantization range (10 pixel grey levels).

The value of the slit lamp illumination was fixed at 3100 lux for all subjects. The slit lamp illumination was checked and adjusted to this value before examination of each subject and slit lamp magnification fixed at 16x.

7.4.1. Effect of different scar densities on visual function.

Any loss of transparency of the cornea will tend to reduce the quality of the retinal image. The extent and nature of the degradation is likely to be related to the size, position, density and scattering properties of the opacity. Common sense would suggest that large, dense opacities aligned with the centre of the pupil would be the most destructive to vision. However, anecdotal evidence suggests that the apparent density of an opacity is not always a good guide to its effect on vision; dense opacities sometimes having surprisingly little effect on vision while diffuse scarring can produce a relatively large reduction in the quality of vision.

A consideration of the scattering properties of opacities provides a possible explanation for this paradoxical result. A dense opacity will reflect and/or absorb much of the light which is incident upon it and relatively little light will be scattered forward onto the retina. This is in contrast to a diffuse opacity which will reflect/absorb less light but scatter more light onto the retinal image. It is this forward scatter of light which is particularly destructive to vision.

To investigate the relationship between scar density / size and visual function, twelve eyes with corneal scarring secondary to keratoconus, were analysed. Vision was assessed by measuring visual acuity (Snellen chart) and contrast sensitivity (Pelli-

Robson charts viewed from 3.6 m ≅ 3.6 cycles / degree). Corneal scarring was assessed using the scar measurement system.

All subjects were keratoconic and wore rigid contact lenses to correct the resultant irregular astigmatism. While in theory, irregular astigmatism is neutralised by the tear lens which forms between the cornea and the back surface of the contact lens, in practice, decentration and movement of the contact lens often results in a less than perfect refractive correction. It is recognised that this may have confounded the results and reduced the strength of any correlation between scar size/density and visual function.

7.4.2. Results.

For ease of analysis, scarring was categorised as either high, medium or low density and the ratio of scarred cornea to clear cornea was calculated for a central 4mm diameter pupil. The output quantization levels 3 and below were taken as low density, 4 and 5 as medium density and 6 or above as high density. Data for twelve eyes (9 subjects) are shown in table T 7.2.

In fig. 7.11 contrast sensitivity has been plotted against standard deviation of scar density with a 2.0mm radius pupil. No significant correlation was found. Standard deviation of scar density was used as an indication of the compactness of the scarring. The correlation was no better when the total scar area was plotted and when scar area was broken down into high, medium and low density (figs. 7.12 - 7.15).

Four scar maps showing the typical range of the visual appearance of the scars are presented in figs. 7.16 - 7.19.

It is clear from these results that the relationship between visual function and the area /density of corneal scarring is complex and probably involves an interaction between

the size, density, position and scattering properties of the scar. The small number of eyes used in this study precluded further analysis of the form of such an interaction. Furthermore, it is likely that the results of this study were confounded by the fact that the subjects were keratoconic and had various degrees of corneal distortion. It would be of interest to repeat the study on non-keratoconic eyes. However, since it is principally forward scatter which is responsible for degradation of the retinal image and there is no simple relationship between back scatter and forward scatter, it is unlikely that simple measurements of back scatter would ever correlate well with the quality of vision.

					scar area %			
Subject	Eye	Scar s.d.	Log C.S.	V.A.	Total	Low	Medium	High
1	L	0.656	1.25	6/12	15.83	13.96	1.84	0
1	R	1.023	1.02	6/12	8.84	4.70	3.63	0.50
2	L	0.476	1.00	6/18	10.89	10.58	0.30	0
2	R	0.804	0.95	6/18	11.61	10.23	1.27	0
3	L	1.010	1.05	6/18	37.38	30.47	6.05	0.86
3	R	0.770	0.50	6/18	27.65	23.58	4.04	0
4	L	0.782	0.72	6/24	28.41	2.33	23.58	2.48
5	R	0.664	0.77	6/36	12.29	10.54	1.74	0
6	R	0.731	1.27	6/9	14.25	2.00	11.87	0.38
7	R	0.762	0.97	6/12	23.18	6.39	16.30	0.47
8	L	0.435	1.05	6/18	21.38	17.25	4.12	0
9	R	1.350	1.10	6/9	44.68	18.21	16.72	9.56

Table T.7.2. Scar area and density, Visual acuity and Log Contrast Sensitivity.

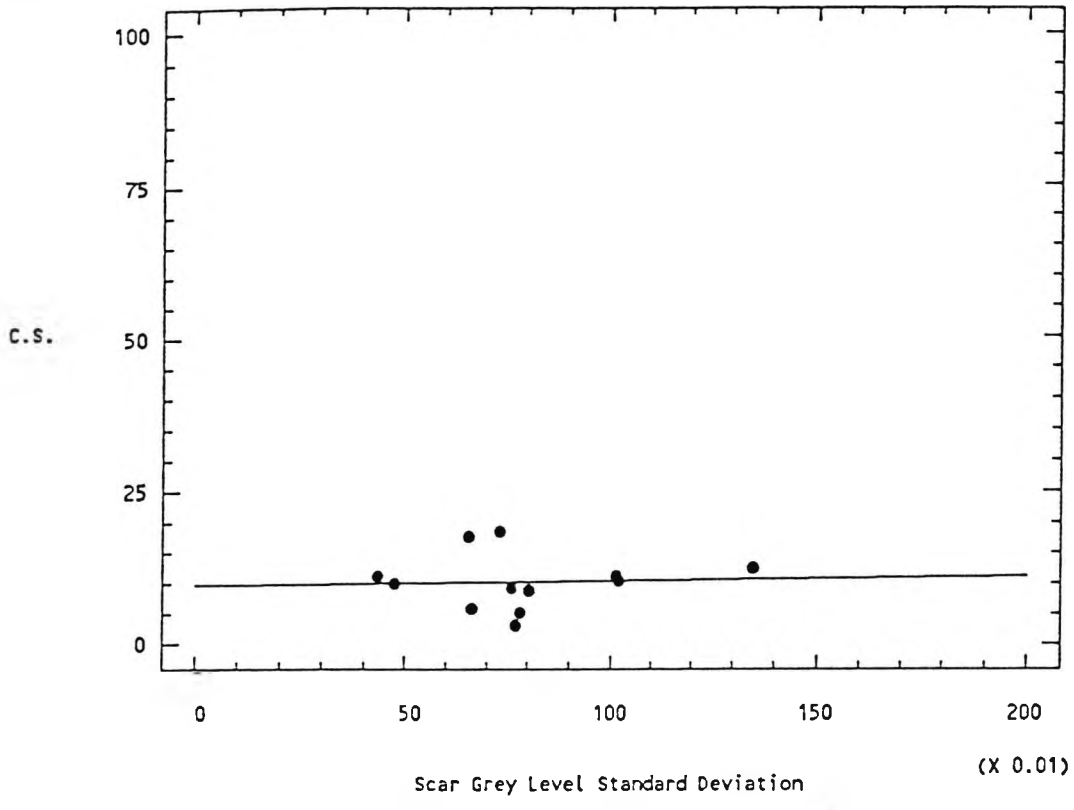


fig. 7.11. Contrast Sensitivity (C.S.) at 3.6 c/deg against standard deviation of Scar Density .

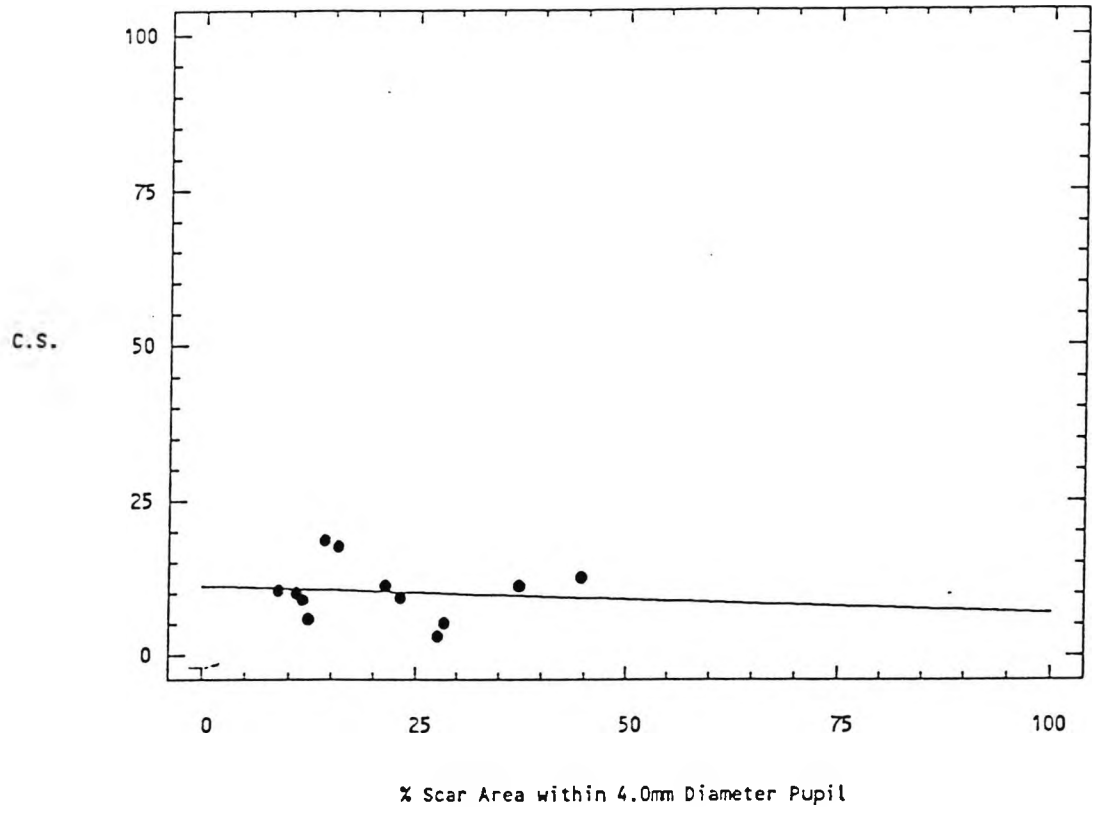


fig. 7.12. Contrast Sensitivity (C.S.) at 3.6 c/deg against % of Pupil Covered by Scar for a 4.0mm Diameter Pupil

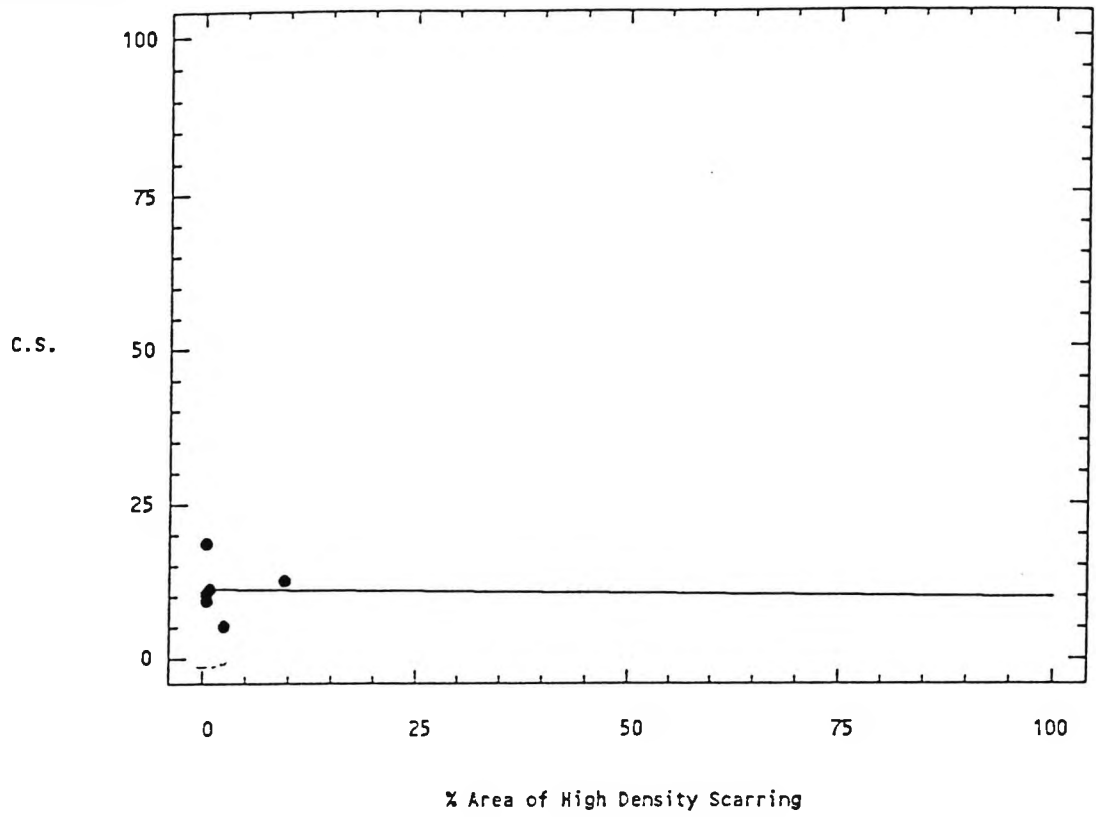


fig. 7.13. Contrast Sensitivity (C.S.) at 3.6 c/deg against % of Pupil Covered by High Density Scarring for a 4.0mm Diameter Pupil

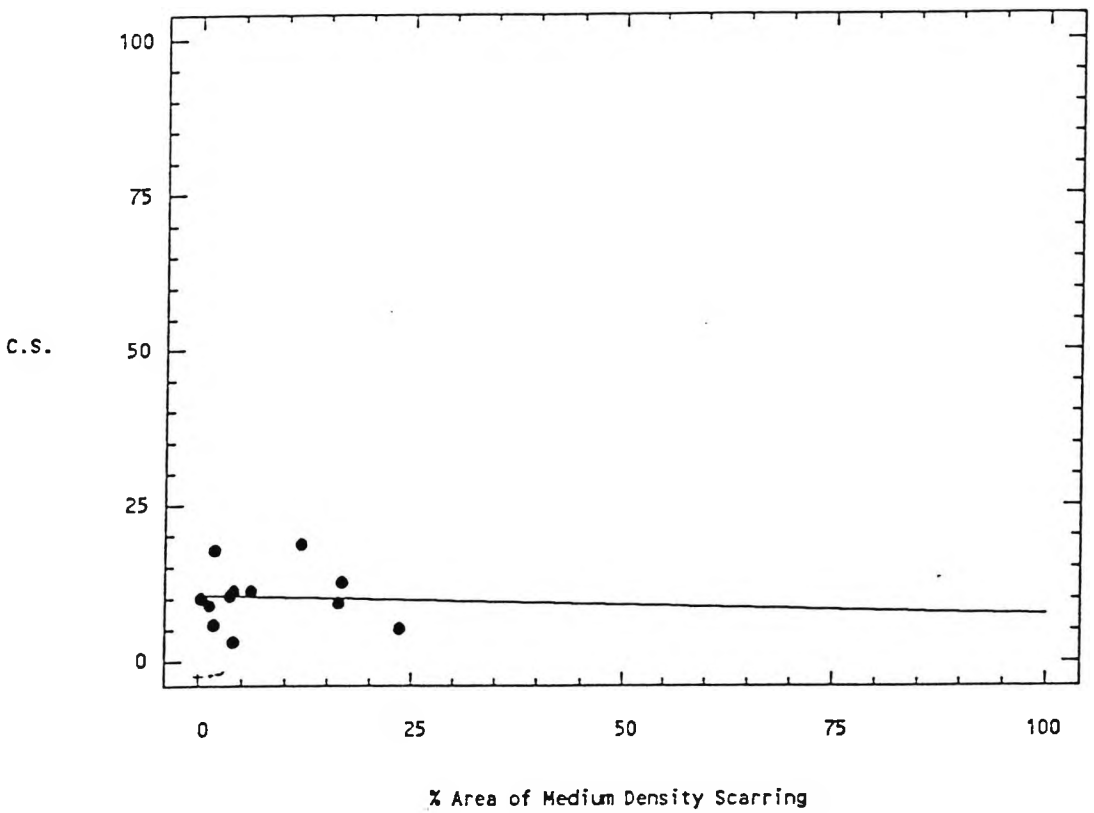


fig. 7.14. Contrast Sensitivity (C.S.) at 3.6 c/deg against % of Pupil Covered by Medium Density Scarring for a 4.0mm Diameter Pupil

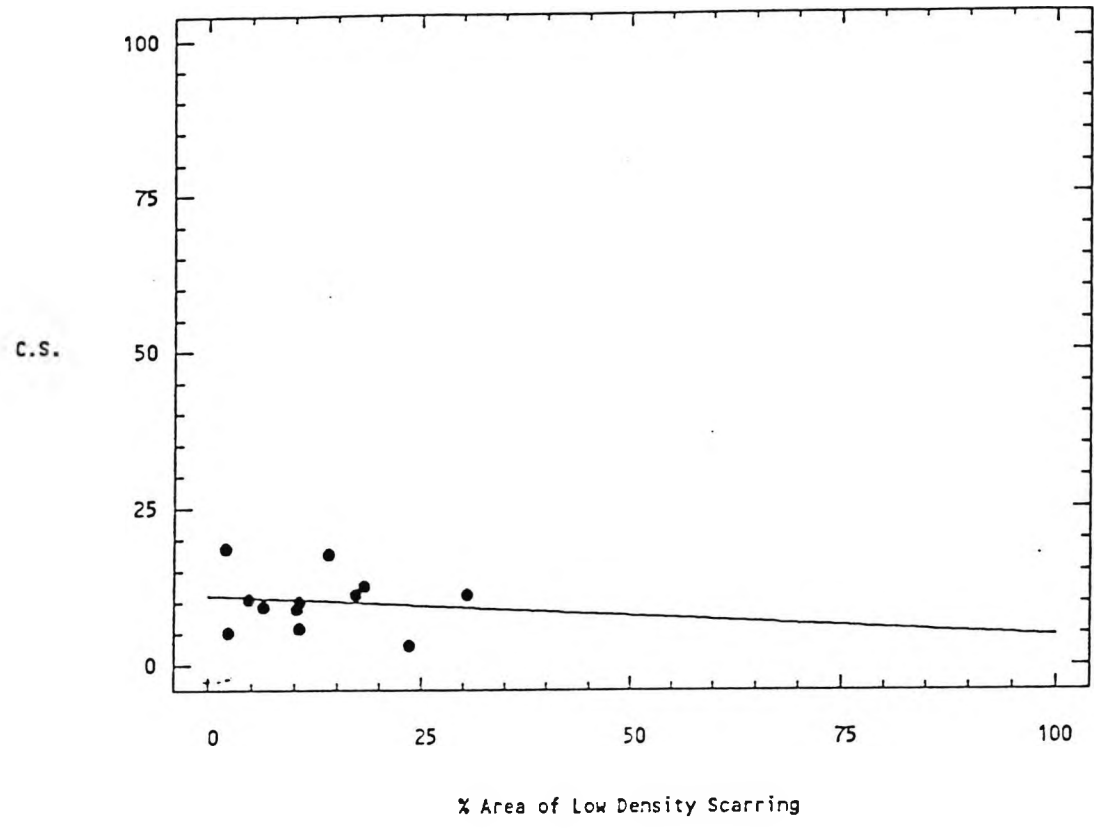


fig. 7.15. Contrast Sensitivity (C.S.) at 3.6 c/deg against % of Pupil Covered by Low Density Scarring for a 4.0mm Diameter Pupil

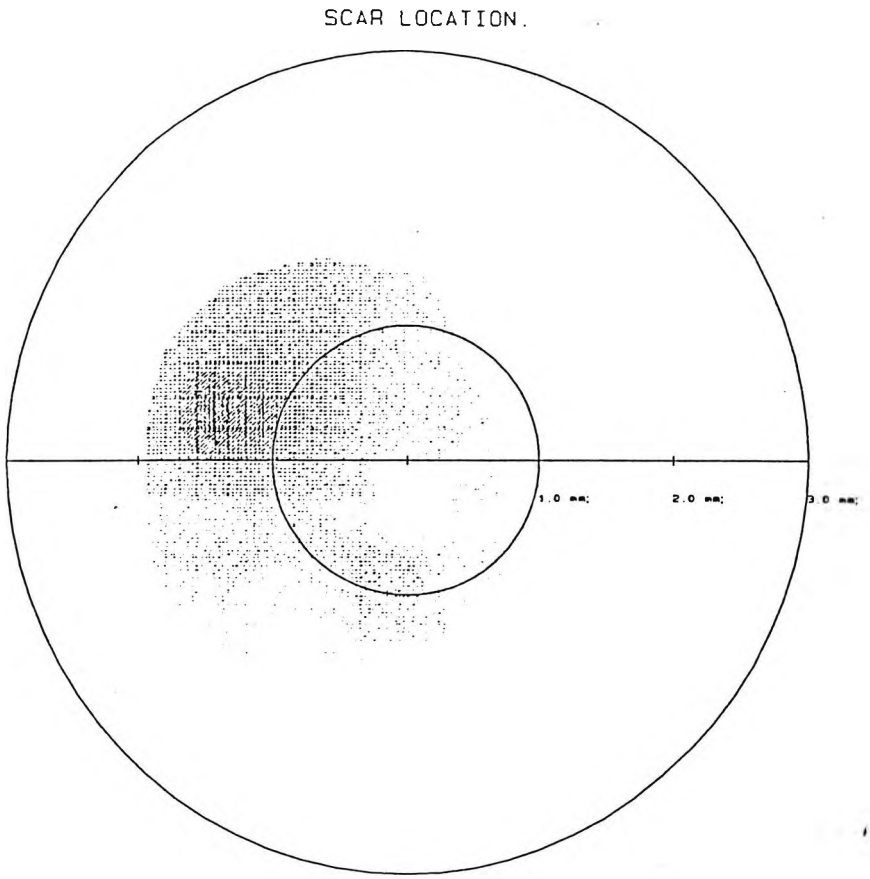


fig. 7.16. subject 4 (left eye). Log C.S.= 0.72, Scar s.d.= 0.782, Scar area% (low density)= 2.33, Scar % area within 2.0mm of pupil centre = 28.41.

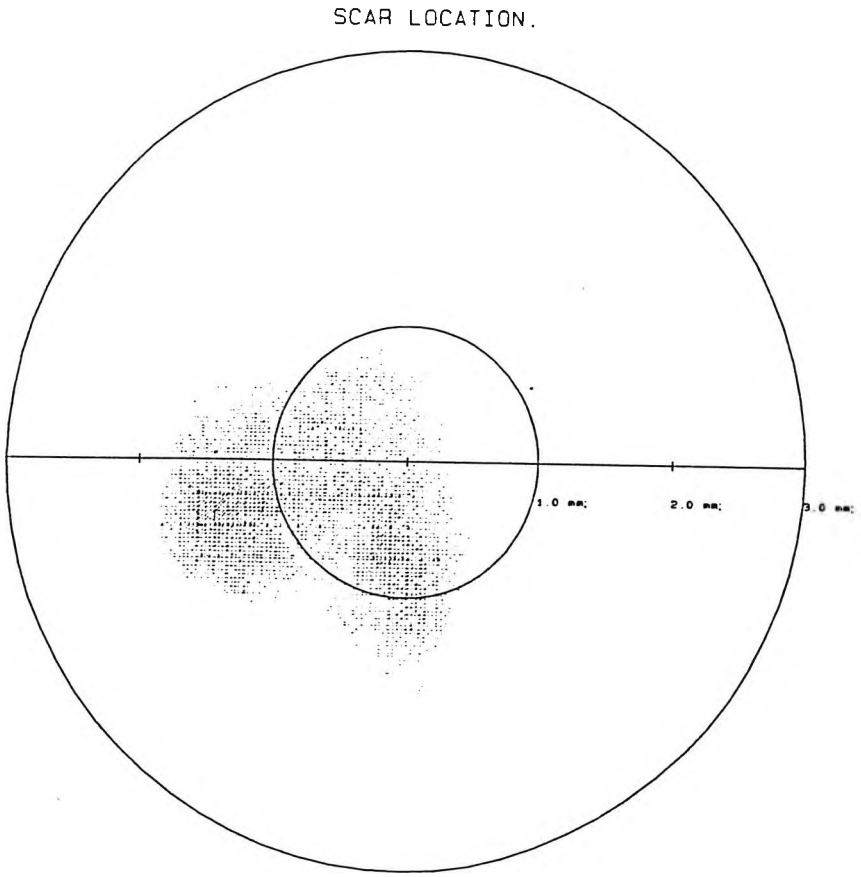


fig. 7.17. subject 7 (right eye). Log C.S. = 0.97, Scar s.d. = 0.762, Scar area% (low density)= 6.39, Scar % area within 2.0mm of pupil centre = 23.18.

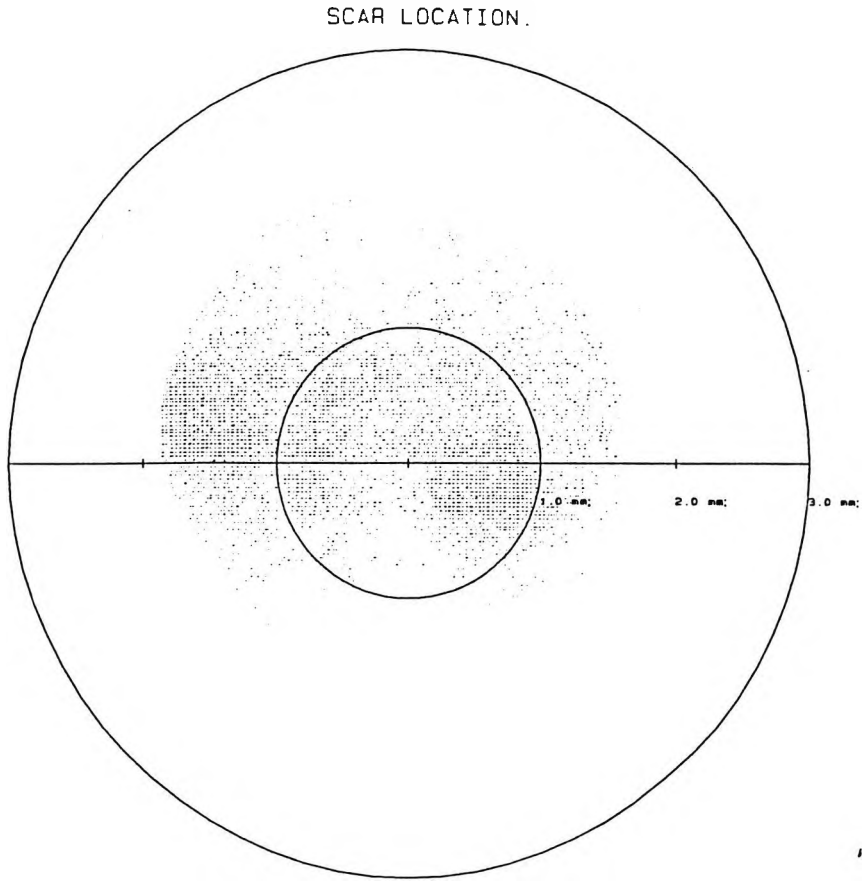


fig. 7.18. subject 8 (left eye). Log C.S. = 1.05, Scar s.d. = 0.435, Scar area% (low density)= 17.25, Scar % area within 2.0mm of pupil centre = 21.38.

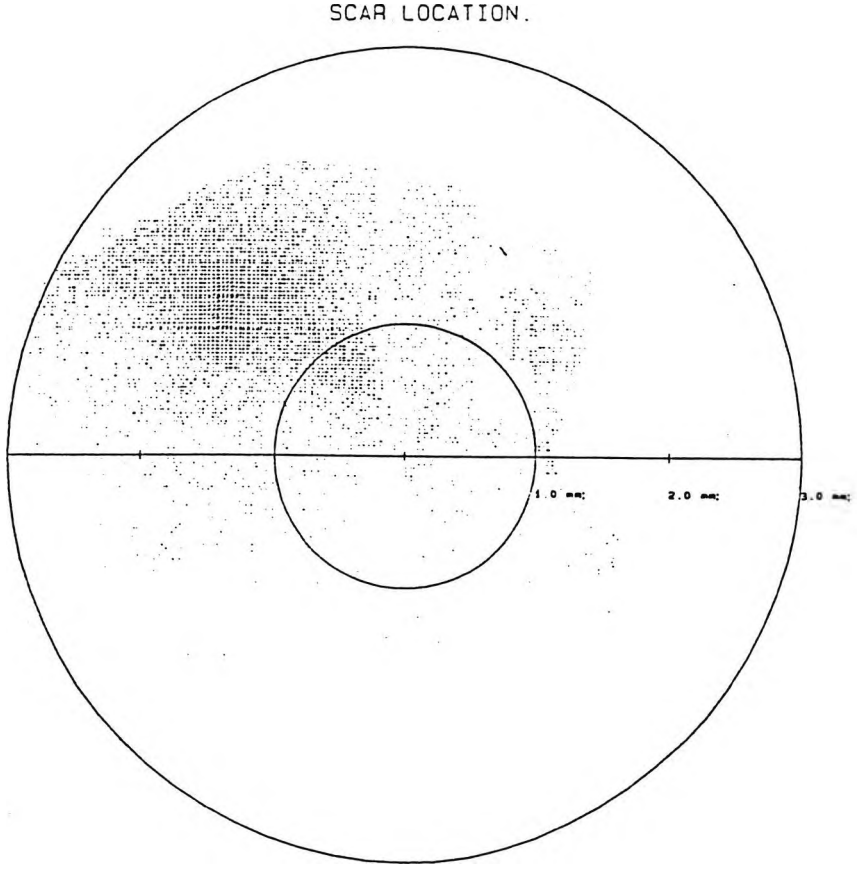


fig. 7.19. subject 1 (left eye). Log C.S. = 1.25, Scar s.d. = 0.656, Scar area% (low density)= 13.96, Scar % area within 2.0mm of pupil centre = 15.83.

The software written for the scar measurement system is given in Appendix C.

Chapter 8
Summary and Conclusion

8.1 Summary

A new system has been developed to measure the topography of the cornea and calculate the radius of curvature at any point on the corneal surface. The system has been shown to be able to measure both normal and irregular corneal surfaces. The results are output in the form of a contour map which gives a good visual impression of the cornea being measured. To give a concise description of the cornea, shape parameters giving the apex radius of curvature, apex distance from the centre of the cornea and rate of flattening from the apex are output with the contour map.

This system is particularly relevant to the reshaping of corneas in PRK surgery. Only a system which can measure any corneal shape and still calculate curvature accurately can be used to monitor the results of PRK. This system is theoretically capable of achieving this measure but the current system hardware requires substantial refinement to be suitable for clinical use.

The system developed uses stereophotogrammetry methods to calculate discrete points which lie on the corneal surface (or more accurately, the air-tear layer boundary where refraction occurs). A polynomial surface is fitted to the set of surface points and differential geometry is applied to the surface equation to calculate radius of curvature at any point. The position of the apex (position of smallest radius of curvature) is identified and other surface parameters are calculated relative to this point. The centre of the cornea is also identified by the system and is used as a reference point when presenting a map of the cornea. Results show that the system configuration used in this project gives a depth resolution of approximately 10 microns and a practical mean error

of 0.01±0.04mm in radius of curvature measurement. Results are presented in chapter 6 of measurements on normal and keratoconic eyes. In total, 30 normal eyes and 10 keratoconic eyes were measured to provide the results for this project, although many other subjects were measured during the development stages of the system.

Measurements of highly distorted keratoconic corneas taken with the Corneal Topography System (CTS) have shown it capable of measuring down to 5.13mm for apex radius of curvature (fig. 6.9).

Shape measurements obtained with the new system (CTS) were compared to values obtained by a conventional keratoscope for 30 normal eyes. A Wesley-Jessen PEK was chosen as the keratoscope for comparison as this instrument is well known and fully documented in the literature.

Values of apex radius of curvature (see table 6.6) show a mean value of 7.82 mm. This is a higher value for CTS relative to the PEK with a mean of 8.04mm. This would be expected as keratoscopes form images from light reflected from a large area of the corneal surface and therefore from areas flatter than the apex. Measurements of rate of flattening from the apex were noticeably different between the two systems, with a rate of 0.04mm/mm for the CTS compared to 0.01 for the PEK. Results suggest that photokeratoscopes are biased towards showing a spherical central curvature area, although CTS results indicate that the region around the apex is very close to spherical in any case (see full results in appendix B).

Measurement of apex distance from the centre of the cornea (table 6.6) show a range of values measured by CTS of 0.05 to 1.23 mm, twice that of the PEK with a range of 0.00 to 0.13. It should be noted that the PEK is not designed to identify the apex position and assumes for correct operation of the instrument that the apex lies along the instrument axis. If the corneal apex position is required to be measured accurately then a photogrammetry type instrument will be required.

Further to the shape measurement system, a system has been developed to quantify corneal scarring. The system utilises a slit lamp with CCD camera attached to grab images of scarred corneas and quantify the scarring. The system is simple in operation and has been found to be convenient to use in a clinical environment. The scar measurement system has been used to study the effect of different scar densities on visual function. The variation and irregularity of scar densities found in patients with scarred corneas means that a single scar density cannot be assessed in isolation to the rest of the scar. However, the density variation of scars was measured for each cornea and correlated with visual function. Results indicate that loss of visual function does not correlate with area of scarring but that low density scarring (haze) may reduce vision more severely than highly dense scarring. This may be due to the fact that measured high density scarring corresponds to a large amount of backscatter radiation. In the case of large backscatter, a proportionately smaller amount of light is transmitted and scattered onto the retina than in the case of low density haze. This system is also very relevant to corneas which have undergone PRK, as they often develop haze over the ablated area.

8.2 Applications.

Applications of the topography measurement and scar measurement systems would include -

1. Assessment of irregular corneal shape prior, during and after photoablative surgery.
2. Assessment of corneal haze after surgery for post-operative planning of treatment.
3. Early detection of keratoconus.
4. Post operative assessment of corneal grafts.
5. Accurate surface description of aspheric contact lenses.

8.3 Further work.

The main improvements to be made in both the topography mapping system and scar mapping system is to fully automate the analysis. The scar mapping system could be refined to automatically identify the pupil boundary and corneal reflex . Then the method would not require the operator to place a boundary around the scar, thus requiring a minimum of operator intervention. The topography system could be developed to include automatic identification of band edges on the corneal image. These developments are necessary for the systems to be able to be used in the clinic by any operator. However, these modifications present a significant challenge for computer image processing techniques in analysing all possible types of images.

With respect to improving the topography system resolution by changing the configuration, the depth resolution would be approximately doubled (to about 4 microns) if the magnification was doubled. This is a viable change if the system is adjusted to use the centre of the pupil as the corneal reference point. At present the centre of the cornea is used as the reference point. In this new configuration, only the central 6mm of the cornea would be in view on the monitor screen.

Currently, the topography system is under further development at Guy's Hospital, London where the projection system is being replaced by an endoscopic projection system. It is hoped that this will create a compact system for use in the clinical environment and will lead to one system, combining functions for measuring scarring and topography.

8.4. Presentations / Publications

The description of the topography measurement system was presented at the 1993 meeting of the Society of Experimental Optometry, under the title 'Measurement of Corneal Topography'.

The mathematical details of the theory and results for keratoconic eyes have been published in Ophthalmic and Physiological Optics 1993 vol. 13, pg 377 - 382 under the title 'Measurement of Corneal Topography in Keratoconus'. (de Cunha , 1993).

Scar measurement results have been published in the Journal of the Optical Society of America. (Barbur, 1993).

REFERENCES

Arffa R. C., Warnicki J. W., Rehkopf P. G. (1989) Corneal topography using rasterstereography. Refractive and Corneal Surgery 5, 414 - 417.

Barbur J L, de Cunha D A, Harlow A J and Woodward E G (1993). In Noninvasive assessment of the visual system Technical Digest, J. Opt. Soc. Am., Washington DC.

Beach R. C. (1990) Curves and surfaces of computer-aided design. Pg 95 - 129. (N.Y. : Van Nostrand Reinhold).

Belin M. W., Litoff D., Strods S. J., Winn S. S., Smith R. S., (1992). The PAR Technology Corneal Topography System. Ref. and Corn. Surg. 8 : 88 - 96.

Benjamin W. J. (1991) in Clinical contact lens practice Ch. 14. Pg 4 - 5. Editors: Bennett E. & Weissman B. (Philadelphia : Lippincott) .

Bennett. A. G. (1964) A new keratoscope and its application to corneal topography. Brit. J. Physiol. Opt. 21 :234. – 238

Bennett A.G. and Rabbetts R.B. Clinical Visual Optics. (1984(a)). Pg 230. (London : Butterworths) .

Bennett A.G. and Rabbetts R.B. Clinical Visual Optics. (1984(b)). Pg 426. (London : Butterworths).

Bergmanson J.P.G. (1991) in Clinical contact lens practice Ch. 4. Pg 1 - 11. Editors: Bennett E. & Weissman B.(Philadelphia : Lippincott).

Bertotto E. V. (1948) The stereophotogrammetric study of the anterior segment of the eye. Amer. J. Ophthal., 31, 573 - 579.

Bibby M. M. (1976) Computer-assisted photokeratoscopy and contact lens design. The Optician 171, 2 - 12.

Boff K. R. and Lincoln J. E. (1988) Engineering data compendium : Human perception and performance. AAMRL, Wright-Patterson AFB, OH.

Bogan S. J., Waring G. O., Ibrahim. O., Drews. C., Curtis L. (1990). Classification of normal corneal topography based on computer-assisted videokeratography. Arch Ophthalmol 108 : 945 - 949.

Bonnet R. and Cachet P. (1962) New method of topographical Ophthalmometry - its theoretical and clinical application. Am. J. of Optom and Arch. Amer. Acad. Optom. 39 : 227 - 251.

Bookstein F. (1979) Fitting conic sections to scattered data. Comp. Graphics. and Image. Proc. 9 : 56 - 71.

Braunstein R.E., Stark W.J., McCally R.L., Connolly P.J., Azar D.T. (1994) Invest. Ophthal. Vis. Sci. 35 (4) : 3493

Caroline P.J., Zilge L.B. (1991) in Clinical contact lens practice Ch. 47. Pg 1 - 7. Editors: Bennett E. & Weissman B. (Philadelphia : Lippincott).

Clark B. A. (1972) Autocollimating photokeratoscope. J. Opt. Soc. Am. 62 : 169 - 176.

Clark B. A. (1973) Less common methods of measuring corneal topography. Aust. J. Optom. 56, 182 - 192.

Clark B. A. (1973b) Systems for describing corneal topography. Aust. J. Optom. 56 : 48 - 56.

Clark B. A. (1974) Mean topography of normal corneas. Aust. J. Optom 57 : 107 - 114

Cohen K. L., Tripoli N. K., Pellom A. C., Kupper L. L., and Fryczkowski A. W. (1984) A new photogrammetric method for quantifying corneal topography. Invest. Ophthalmol Vis. Sci. 25 : 323 - 330.

Crouch J. E. (1978) Functional Human Anatomy. Pg 398 - 400 (Philadelphia : Lea & Febiger).

Davson H. (1972) The Physiology of the eye. Pg 69 - 70 (London : Churchill Livingston).

de Cunha D. A., Woodward E. G. (1993) Measurement of corneal topography in

keratoconus. Ophthal. Physiol. Opt., 13 : 377 - 382.

Drysdale C. V. (1900) On a simple direct method of determining the curvature of Small lenses. Trans. Opt. Soc. 2 : 1900 - 1901

Edmund C. (1987) Location of the corneal apex and its influence on the stability of the central corneal curvature. A photokeratoscope study. Am. J. of Optom. Physiol. Optics. 64 : 846 - 852.

Ellerbrock V. J. (1961) Variables in corneal topography. Amer. J. Optom. 38 : 556 - 563.

Fujii T., Maruyama S., Ikeda M. (1972) Determination of corneal configuration by the measurement of its derivatives. Optica Acta. 19 : 425 - 430.

Gartry D. Kerr Muir M. Marshall J. (1991) Exicimer laser treatment of corneal surface pathology: a laboratory and clinical study. Br. J. Ophthalmol. 75 : 258 - 269.

Gonzalez R. C. and Wintz P. (1987) Digital Image Processing. Pg 146 - 152 (Reading, Ma. : Addison-Wesley).

Gormley D. J., Gersten M., Koplin R. S., Lubkin V. (1988) Corneal modeling. Cornea 7(1), 30 - 35.

Gullstrand A. (1924) in Physiological Optics by H. von Helmholtz. Vol 1. 311. (Rochester. N.Y : Optical Society of America)

Hall E., Tio J., McPherson C. and Sadjadi F. (1982) Measuring curved surfaces for robot vision. Computer 15(12) : 42 - 54.

Hannush S. B., Crawford S. L., Waring G. O., Gemmill M. C., Lynn M. J., Nizam A. (1989) Accuracy and precision of keratometry, photokeratoscopy and corneal modeling on calibrated steel balls. Arch. Ophthalmol. 107 : 1235 - 1239.

Hannush S. B., Crawford S. L., Waring G. O., Gemmill M. C., Lynn M. J., Nizam A. (1990) Reproducibility of normal corneal power measurements with a keratometer, photokeratoscopy and Video imaging system. Arch. Ophthalmol. 108 : 539 - 544.

Hirst L.W.(1991) in Clinical contact lens practice Ch. 6. Pg 1 - 5. Editors:

Bennett E. & Weissman B. (Philadelphia : Lippincott).

Idesawa M., Yatagai T. and Soma T. (1977) Scanning moiré method and automatic measurement of 3-D shapes. Applied Optics 16 : 2152 - 2162.

Karabatsas C., Cook S.D., Easty D.L. (1994) Invest. Ophthal. Vis. Sci. 35 (4) : 2902.

Kaufman H.E., Werblin T.P. (1982) Epikeratoplasty for the treatment of Keratoconus. Am. J. Ophthalmol 93 : 342 - 348 .

Kawara T. (1979) Corneal topography using moiré contour fringes. Applied Optics 18, 3675 - 3678.

Kelly D. H. (1977) Visual contrast sensitivity. Optica Acta. 24 (2) : 107 -129.

Kiely P. M., Smith G. and Carney L. G. (1982) The mean shape of the human

Optica Acta. 29 : 1027 - 1040

Knoll H.A. (1961) Corneal contours in the general population as revealed by the photokeratoscope. Am. J. Optom. and Arch. Am. Acad. Optom. 38(July): 389 - 397.

Krueger R.R., Saedy N.F., McDonnell P.J. (1994) Clinical analysis of topog-ap-- steep central islands following excimer laser photorefractive keratectomy. Invest. Ophthal. Vis. Sci. 35 (4) :2250.

Lamberts DW. (1983)Physiology of the tear film. In Smolin G, Thoft RA. The Cornea. Scientific Foundations and Clinical Practice, pg 31(Boston; Little, Brown & Co.)

Lohmann C.P., Gartry D.S. Kerr Muir M., Timberlake G.T., Fitzke F.W., Marshall J. (1991) Corneal haze after excimer laser refractive surgery : objective measurements and functional implications. Eur. J. Ophthmol. 1(4) : 173 - 180.

Longhurst R.S. (1973) Geometrical and Physical Optics. Pg 424. (New York : Longman).

Maguire L.J., Klyce S.D., Singer D.E. McDonald M.B., Kaufman H.E. (1987) Corneal topography in myopic patients undergoing Epikeratophakia. Am. J. Ophthalmol. 103 : 404 - 416.

Mandell R. B. (1966) Corneal curvature measurements by the aid of moiré fringes.

J. Amer. Optom. Ass. 37 : 219. – 220

Mandell R. B. and St. Helen R. (1971) Mathematical model of the corneal contour. Brit. J. Physiol. Opt. 26 : 183 - 197.

Maurice D. (1957) The structure and transparancy of the cornea. J. Physiol 136 : 263 - 272.

McCarey B. E., Zurawski C. A., O'Shea D. S. (1992) Practical aspects of a corneal topography system. The CLAO Journal 18(4), 248 - 254.

McDonnell P.J., Garbus J. (1989) Corneal topographic changes after radial keratotomy. Ophthalmology 96 : 45 - 49.

McMonnies C. W. (1971) Corneal curvature from profile measurements. Aust. J. Optom., 54, 153 - 162.

Olsen T. (1982) Light scatter in the human cornea. Invest. Ophthalmol. Vis. Sci. 23 : 81 - 86.

Porrill J. (1990) Fitting ellipses and predicting confidence envelopes using a bias corrected Kalman filter. Image and Vision Computing. 8 : 37 - 41.

Rabinowitz Y. S., Garbus. J., McDonnell P. J. (1990) Computer-assisted corneal topography in family members of patients with keratoconus. Arch. Ophthalmol. 108 : 365 - 371.

Roberts C. (1994) Characterization of the inherent error in a spherically-based corneal topography system in mapping a radially aspheric surface. Refractive and Corneal Surgery 10 : 103 - 116.

Rogers D. F. and Adams J. A.(1976) Mathematical elements for computer graphics. Pg 158 - 182. (N.Y. : McGraw - Hill).

Rowsey J. J. Reynolds A. E. Brown R. (1981) Corneal Topography : Corneascope.

Rubin M.. (1975) Contact Lens Practice. Pg 108 - 111. (London : Bailierê Tundall.)

M. (1993) A two-year experience with excimer laser photorefractive keratectomy

for myopia. Ophthalmology. 100 : 873 - 882.

Sampson P. D. (1982) Fitting conic sections to "very scattered" data : an iterative refinement of the Bookstein algorithm. Comp. Graph. and Image. Proc. 18 : 97 - 108 .

Sampson W. G., Soper J. W. and Girard L. J. (1965) Topographical keratometry and contact lenses. Trans. Amer. Acad. Ophthal. Otolaryng. 69 : 959.– 960

Sato Y., Kitagawa H., Fujita H. (1982) Shape measurements of curved objects using multiple slit-ray projections. IEEE Transactions on pattern analysis and machine intelligence , PAMI - 4, 641 - 646.

Seiler T. Holschbach A. Derse M. Jean B. Genth U. (1994) Complications of myopic photorefractive keratectomy with the excimer laser. Ophthalmology. 101 : 153 - 160

Smith G. T. H., Brown N. A. P., Shun-Shin G. A. (1990) Light scatter from the central human cornea. Eye. 4 : 584 - 588.

Stoker J. J. (1969) Differential Geometry. Pg 74 - 92. (N.Y. : Wiley).

Stone J. (1975) Contact Lens Practice. M. Ruben. Bailierê Tundall. Pg 106 - 108.

Struik D. J. (1950) Lectures on classical differential geometry. Pg 55 -113. (N.Y. : Dover).

Taylor S. Fields C. Barker F. (1994) Effect of depth upon the smoothness of excimer laser corneal ablation. Optom. and Vis. Sci. 71 : 104 - 108.

Theocaris P. (1969) Moiré fringes in strain analysis. Pg 219 - 255 . (Oxford : Pergamon).

Townsley M. G. (1967) New equipment and methods for determining the contour of the human cornea. Contacto 11, 72 - 81.

Townsley M. G. (1970) New knowledge of the corneal contour. Contacto 14 : 38 - 43.

Troutman R. and Buzard K. (1992) Corneal Astigmatism. Pg 114 - 115. (St. Louis. : Mosby-Year Book).

Tsilimbaris M., Vlachonckolis I., Siganos D., Makridakis G., Pallikaris I. (1991) Comparison of keratometric readings as obtained by Javal Ophthalmometer and Corneal Analysis System (EyeSys). Ref. and Corn. Surg. 7 : 368 - 373.

Van Saarloos P. and Constable I. (1991) Improved method for calculation of corneal topography for any photokeratoscope geometry. Optom. and Vis. Sci. 68 : 960 - 965.

Vest (1979) Optical Holography. (N.Y. : Wiley).

Warnicki J. W., Rehkopf P. G., Curtin D. Y., Burns S. A., Arffa R. C., and Stuart J. C. (1988) Corneal topography using computer analyzed rasterstereographic images. Applied Optics 27, 1135 - 1140.

Wilson S. and Klyce S. (1991) Advances in the analysis of corneal topography. Surv. Ophthalmol 35 : 269 - 277.

Wilson S. and Klyce S. (1994) Screening for corneal topographic abnormalities before refractive surgery. Ophthalmology. 101 : 147 - 152.

Wray L. (1970a) Holography:part 1. The Optician 159 : 703 - 706.

Wray L. (1970b) Holography:part 2. The Optician 160 : 113 - 116.

Yoon Phin Kon (1990) Some methods of corneal measurement. Contact Lens Journal 19, No. 8. Continuing Education No. 1.

Yuen H. K., Illingworth J., Kittler J. (1989) Detecting partially occluded ellipses using the Hough transform. Image and Vis. Comp. 7 : 31 - 37.

Appendix A.

```
#----------------------------------------
#       HELLOWIN.MAK make file
#----------------------------------------

hellowin.exe:hellowin.obj hellowin.def hellowin.res
        link /NOD hellowin ,hellowin,Nul,llibcew libw itxvspml itxofgml cmpofg
             itxstbml, hellowin
        rc hellowin.res

hellowin.obj:hellowin.c
        cl /c /DAT /DMSC /AL /Gsw /Ow /W2 /Zp hellowin.c


hellowin.res:hellowin.rc
        rc -r hellowin.rc
```

```
#MAKEFILE


#COMPILE FLAG MACRO DEFINITION
#/Zi PREPARE FOR CODEVIEW DEBUGGING
#/AL LARGE MEMORY MODEL (LIB LLIBCE.LIB)

CFLAGS=/DAT /DMSC /AL /Zi /F 1024

#LINK FLAG MACRO DEFINITION

LFLAGS=/CO


#COMPILE LINE
#TARGET : {PATH;}DEPENDENTS
# ! DIRECTIVE $(cc) COMPILE WITH cl, COMPILER FLAGS,
# COMPILE ONLY, SAME AS (TARGET FILENAME).C

hello.obj : {c:\mc\progs;}hello.c
        !cl $(CFLAGS) /c hello.c


#LINK COMAND LINE

hello.exe : {c:\mc\progs;}hello.obj
        !link $(LFLAGS) hello.obj,,,llibce itxvspml itxofgml cmpofgml
         itxstbml, NUL
```

```
/*--------------------------------------*/
/*    HELLOWIN.RC resource file         */
/*--------------------------------------*/
#include <resource.h>

topomenu MENU
BEGIN
   POPUP "Camera"
       BEGIN
         MENUITEM "Initilise system",INIT
         MENUITEM "Camera On",VIEW
         MENUITEM "Freeze Image",FREEZE
         MENUITEM "Alignment Lines",ALIGN
         MENUITEM "Input Enhancement",C_ENHANCE
         MENUITEM "Save Image",SAVEIM
       END
   POPUP "Calibrate"
       BEGIN
          MENUITEM "Center Left",CL_CAL
          MENUITEM "Center Right",CR_CAL
          MENUITEM "Outside Right",OR_CAL
          MENUITEM "Outside Left",OL_CAL
          MENUITEM "Calibrate left",L_CAL
          MENUITEM "Calibrate Right",R_CAL
          MENUITEM "Systematic Correction",C_ERROR
          MENUITEM "Test Plane Image",C_PLANE
       END
   POPUP "Pictures"
       BEGIN
           MENUITEM "Restore from Disk",P_RESTORE
           MENUITEM "Clear Overlay",P_COVERLAY
           MENUITEM "Extract Points",P_EXTRACT
           MENUITEM "Plot Results",P_PLOT
           MENUITEM "Surface Fit",P_SURFACE
           MENUITEM "Curvature",P_CURVE
       END
   POPUP "Enhancements"
       BEGIN
            MENUITEM "Equalization",E_EQUIL
            MENUITEM "Sharpen",E_SHARPEN
            MENUITEM "Smooth",E_SMOOTH
       END
END

map ICON map.ico
```

```
/*---------------------------*/
/*   resource.h header file  */
/*---------------------------*/
#define INIT 1
#define VIEW 2
#define FREEZE 3
#define ALIGN 4
#define CL_CAL 5
#define CR_CAL 6
#define OL_CAL 7
#define OR_CAL 8
#define L_CAL 9
#define R_CAL 10
#define SAVEIM 11
#define P_RESTORE 12
#define P_COVERLAY 13
#define P_EXTRACT 14
#define C_ENHANCE 15
#define E_EQUIL 16
#define E_SHARPEN 17
#define E_SMOOTH 18
#define P_SURFACE 19
#define P_PLOT 20
#define C_ERROR 21
#define C_PLANE 22
#define P_CURVE 23
```

```
;--------------------------------------
; HELLOWIN.DEF module definition file
;--------------------------------------

NAME            HELLOWIN

DESCRIPTION     'Hello windows Program'
EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE DISCARDABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       8192
EXPORTS         wndproc
```

```
/************* hellowin.c ************/

#include <windows.h>
#include <process.h>
#include <resource.h>
#define CX 420
#define CY 460

int lcflag=0,rcflag=0;

long FAR PASCAL wndproc(HWND,WORD,WORD,LONG);


int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
                   LPSTR lpszCmdParam, int nCmdShow){
    static char szAppName[]="hellowin";
    HWND hwnd;
    MSG msg;
    WNDCLASS wndclass;
    int x;

    if (!hPrevInstance){
       wndclass.style         =CS_HREDRAW|CS_VREDRAW;
       wndclass.lpfnWndProc   =wndproc;
       wndclass.cbClsExtra    =0;
       wndclass.cbWndExtra    =0;
       wndclass.hInstance     =hInstance;
       wndclass.hIcon         =LoadIcon (hInstance,"map");
       wndclass.hCursor       =LoadCursor(NULL,IDC_ARROW);
       wndclass.hbrBackground =GetStockObject(WHITE_BRUSH);
       wndclass.lpszMenuName  ="topomenu";
       wndclass.lpszClassName =szAppName;

       RegisterClass(&wndclass);
       }
  hwnd=CreateWindow(szAppName,            /*window class name*/
              "Topography ",              /*window caption*/
              WS_OVERLAPPEDWINDOW,        /*window style*/
              CW_USEDEFAULT,              /*initial x position*/
              CW_USEDEFAULT,              /*initial y position*/
              CX,                         /*initial x size*/
              CY,                         /*initial y size*/
              NULL,                       /*parent window handle*/
              NULL,                       /*window menu handle*/
              hInstance,                  /*program instance handle*/
              NULL);                      /*creation parameters*/

  ShowWindow(hwnd,nCmdShow);
  UpdateWindow(hwnd);

  while(GetMessage(&msg,NULL,0,0))

       {
       TranslateMessage(&msg);
       DispatchMessage(&msg);
       }
```

```
    return(msg.wParam);
    }

long FAR PASCAL wndproc(HWND hwnd, WORD message, WORD wParam, LONG lParam){
    HDC         hdc;
    PAINTSTRUCT ps;
    RECT        rect;
    HMENU       hMenu;
    static HPEN hPen1;
    int x;

    switch(message){
          case WM_CREATE:/*sent when createwindow is called in winmain*/
               hPen1=CreatePen(PS_SOLID,1,RGB(0,0,255));
               break;
          case WM_PAINT:
               hdc=BeginPaint(hwnd,&ps);
               GetClientRect(hwnd,&rect);
               SetMapMode(hdc,MM_ISOTROPIC);
               SetWindowExt(hdc,5000,5000);
               SetViewportExt(hdc,CX/2,-CY/2);
               SetViewportOrg(hdc,CX/2,CY/2-15);
               SelectObject(hdc,hPen1);
               Ellipse(hdc,-4000,4000,4000,-4000);
               SelectObject(hdc,GetStockObject(BLACK_PEN));
               MoveTo(hdc,-4000,0);
               LineTo(hdc,4000,0);
               MoveTo(hdc,0,4000);
               LineTo(hdc,0,-4000);
               TextOut(hdc,-4900,100,"-4mm",4);
               TextOut(hdc,4050,100,"4mm",3);
               TextOut(hdc,-250,4500,"4mm",3);
               TextOut(hdc,-250,-4000,"-4mm",4);
               EndPaint(hwnd, &ps);
               break;
         case WM_COMMAND:
               hMenu=GetMenu(hwnd);
               CheckMenuItem(hMenu,wParam,MF_CHECKED);
               switch(wParam){
                     case INIT:WinExec("hello 1",SW_SHOWNA);
                               CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                               break;
                     case VIEW:WinExec("hello 2",SW_SHOWNA);
                               CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                               break;
                     case FREEZE:WinExec("hello 3",SW_SHOWNA);
                               CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                               break;
                     case ALIGN:WinExec("hello 4",SW_SHOWNA);
                               CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                               break;
                     case C_ENHANCE:WinExec("hello 15",SW_SHOWNA);
                               CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                               break;
                     case SAVEIM:WinExec("hello 11",SW_SHOWNA);
                               CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                               break;
```

```
                    case CL_CAL:WinExec("hello 5",SW_SHOWNA);
                           break;
                    case CR_CAL:WinExec("hello 6",SW_SHOWNA);
                           break;
                    case OL_CAL:WinExec("hello 7",SW_SHOWNA);
                           break;
                    case OR_CAL:WinExec("hello 8",SW_SHOWNA);
                           break;
                    case L_CAL:WinExec("hello 9",SW_SHOWNA);
                          lcflag=1;
                          break;
                    case R_CAL:WinExec("hello 10",SW_SHOWNA);
                          rcflag=1;
                          break;
                    case C_ERROR:WinExec("hello 21",SW_SHOWNA);
                          if((lcflag==1)&&(rcflag==1))
                           {lcflag=rcflag=0;
                            for(x=CL_CAL;x<=R_CAL;x++)CheckMenuItem(
hMenu,x,MF_UNCHECKED);
                           CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                           }
                         break;
                    case C_PLANE:WinExec("hello 22",SW_SHOWNA);
                           CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                           break;
                    case P_RESTORE:WinExec("hello 12",SW_SHOWNA);
                           CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                           break;
                    case P_COVERLAY:WinExec("hello 13",SW_SHOWNA);
                           CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                           break;
                    case P_EXTRACT:WinExec("hello 14",SW_SHOWNA);
                           CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                           break;
                    case P_SURFACE:WinExec("hello 19",SW_SHOWNA);
                           CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                           break;
                    case P_CURVE:WinExec("tplot 23",SW_SHOWNA);
                           CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                           break;
                    case E_EQUIL:WinExec("hello 16",SW_SHOWNA);
                           CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                           break;
                    case E_SHARPEN:WinExec("hello 17",SW_SHOWNA);
                           CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                           break;
                    case E_SMOOTH:WinExec("hello 18",SW_SHOWNA);
                           CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                           break;
                   case P_PLOT:WinExec("hello 20",SW_SHOWNA);
                           CheckMenuItem(hMenu,wParam,MF_UNCHECKED);
                           break;

            }


            break;
```

```
        case WM_DESTROY:
             DeleteObject(hPen1);
             PostQuitMessage(0);
             return 0;
        }


    return (DefWindowProc(hwnd, message, wParam, lParam));
    }
plot(){

}
```

```
/******************  hello.c Topography program **************/
#include <stdio.h>
#include <graph.h>
#include <itexvsp.h>
#include <math.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <process.h>
#include <resource.h>
#define ORDER 3         /*polynomial order for basis of patch*/
#define BAND 2          /*take points every 2,3, or 4 (BAND) bands*/
#define POINTS 9        /*number of lines to take points*/
#define PLANES 6        /*number of light bands to measure on each side*/
#define CONST3 1.10     /*3mm zone correction*/
#define CONST15 1.25    /*apex R. of C. correction*/
#define CONSTA 1.35     /*radius of curvature correction*/
#define R_FACTOR 0.3

int gaoi,gaoil,gaoir,top=255,bottom=1;
/*kernel for image dilation*/
int v_kernel[15]={
                1,0,-1,
                1,0,-1,
                1,0,-1,
                1,0,-1,
                1,0,-1
                };
char name[10];
int ftop=3,fbottom=509;/*limits of extracted points*/
int centline;
float fcx=0.0,fcy=0.0;/*cornea centre displacement*/
double factor=1.00,dist_per_pixel=20.11;
/*chosen co-ordinate matrices*/
static double px[ORDER][ORDER],py[ORDER][ORDER],pz[ORDER][ORDER];
double pxrect[ORDER][ORDER],pyrect[ORDER][ORDER],pzrect[ORDER][ORDER];
double curve[20][20];
double centrx=0.0;
double rightup[3],rightdown[3];/*x,y,z points from right matrix for*/
                               /*interpolation 'wing points'       */

double matrix();
double matsp();
FILE *fp;

main(argc,argv)
int argc;
char *argv[];
{
char opt;
int arg;
load_cnf("c:\\visnplus\\itex\\lib\\vsp.cnf");
gaoi=ofg_gaoi_fbcreate(CURRENT_F,0,0,768,512);
gaoil=ofg_gaoi_fbcreate(CURRENT_F,0,0,384,512);
gaoir=ofg_gaoi_fbcreate(CURRENT_F,384,0,384,512);
ofg_setframe(I);
```

```
ofg_camera(VIDEO0);
centline=POINTS/2;
if(argc==1){
     while(opt!='e' && opt!='E'){
          menu();
          opt=getch();
          printf("%c",opt);
          switch(opt){
                case 'a':real_t();
                         break;
                case 'b':grab();
                         break;
                case 'c':sharpen();
                         break;
                case 'd':sobel();
                         break;
                case 'f':save();
                         break;
                case 'g':restore();
                         break;
                case 'h':equalise();
                         break;
                case 'i':auto_equalise();
                         break;
                case 'j':median();
                         break;
                case 'k':mul_mat();
                         break;
                case 'l':calib_pix();
                         break;
                case 'm':upper_thresh();
                         break;
                case 'n':lower_thresh();
                         break;
                case 'p':m_calibration();
                         break;
                case 'q':r_extract();
                         break;
                case 'u':l_extract();
                         break;
                case 's':surface();
                         break;
                case 'v':clear_overlay();
                         break;
                case 'z':grabinit();
                         break;
                }/*end case*/
          }/*end while*/

          ofg_freeze();}
/*****************************************/
/************** window section **************/
/*****************************************/
else{
    arg=atoi(argv[1]);
    switch(arg){
        case INIT:grabinit();
```

```
            break;
 case VIEW:real_t();
            break;
 case FREEZE:grab();
             break;
 case ALIGN:m_calibration();
            break;
 case C_ENHANCE:w_auto_equalise();
            break;
 case SAVEIM:save();
             break;
 case CL_CAL:real_t();
             grab();
             printf("\n\nsharpening image");
             sharpen();
             printf("\n\nsaving image");
             ofg_im_save(gaoi,EIGHT_BIT,"l1");
             break;
 case CR_CAL:real_t();
             grab();
             printf("\n\nsharpening image");
             sharpen();
             printf("\n\nsaving image");
             ofg_im_save(gaoi,EIGHT_BIT,"r2");
             break;
 case OL_CAL:real_t();
             grab();
             printf("\n\nsharpening image");
             sharpen();
             printf("\n\nsaving image");
             ofg_im_save(gaoi,EIGHT_BIT,"l2");
             break;
 case OR_CAL:real_t();
             grab();
             printf("\n\nsharpening image");
             sharpen();
             printf("\n\nsaving image");
             ofg_im_save(gaoi,EIGHT_BIT,"r1");
             break;
 case L_CAL:win_calib('l');
            break;
 case R_CAL:win_calib('r');
            break;
 case P_RESTORE:restore();
            break;
 case P_COVERLAY:clear_overlay();
            break;
 case P_EXTRACT:restore();
                centre();
                limits();
                r_extract();
                l_extract();
            break;
 case P_SURFACE:mul_mat();
            break;
 case P_CURVE:mul4_mat();
            break;
```

```
        case E_EQUIL:equalise();
                     break;
        case E_SHARPEN:sharpen();
                     break;
        case E_SMOOTH:median();
                     break;
        case P_PLOT:plot();
                     break;
        case C_ERROR:systemerr();
                     break;
        case C_PLANE:real_t();
                     grab();
                     save();
                     break;
        }/*end switch*/
    };/*end if else*/
/*******************************************/
}/*end main*/

cls(){
      printf("\033[2J");
}
menu(){
      cls();
      printf("\n");
      printf("                          \033[7mTopography image test\033
[0m\n\n");
      printf("                            a..real time view\n");
      printf("                            b..grab image\n");
      printf("                            c..sharpen image\n");
      printf("                            d..sobel operator\n");
      printf("                            f..save image\n");
      printf("                            g..restore image\n");
      printf("                            h..histogram equalisation\n");
      printf("                            i..input lut equalisation\n");
      printf("                            j..median filter\n");
      printf("                            k..curvature calculation\n");
      printf("                            l..pixel calibration\n");
      printf("                            m..upper threshold\n");
      printf("                            n..lower threshold\n");
      printf("                            p..movement calibration\n");
      printf("                            q..extract points right\n");
      printf("                            u..extract points left\n");
      printf("                            s..display surface\n");
      printf("                            v..clear overlay frame\n");
      printf("                            z..initialise frame grabber\n");
      printf("                            e..end\n");
      printf("                     ENTER OPTION==>");
      printf("%c",7);
}

grabinit(){
     load_cnf("c:\\visnplus\\itex\\lib\\vsp.cnf");
     initsys();
     ofg_init();
     gaoi=ofg_gaoi_fbcreate(CURRENT_F,0,0,768,512);
     gaoil=ofg_gaoi_fbcreate(CURRENT_F,0,0,384,512);
```

```
     gaoir=ofg_gaoi_fbcreate(CURRENT_F,384,0,384,512);
     ofg_setframe(O);
     ofg_setvframe(I);
     ofg_clf(gaoi,0);
     ofg_setframe(I);
     ofg_camera(VIDEO0);
     ofg_grab(CAMERA,gaoi);

}
plot(){
    char fname[10],depth[10];
    int i,j;
     /* matrices of points saved on disk */
     static double trx_pos[POINTS][PLANES],try_pos[POINTS][PLANES],
trz_pos[POINTS][PLANES];
     static double tlx_pos[POINTS][PLANES],tly_pos[POINTS][PLANES],
tlz_pos[POINTS][PLANES];
     /*wing coordinates*/
     double wu_xpos=0.0,wu_ypos,wu_zpos,wl_xpos=0.0,wl_ypos,wl_zpos;
     cls();
     printf("\n file name : ");
     gets(name);
     strcpy(fname,name);
     /*input left side points*/
     strcat(fname,".dal");
     fp=fopen(fname,"rb");
     fread(tlx_pos,sizeof(tlx_pos),1,fp);
     fread(tly_pos,sizeof(tly_pos),1,fp);
     fread(tlz_pos,sizeof(tlz_pos),1,fp);
     fclose(fp);
     /*input right point file */
     strcpy(fname,name);
     strcat(fname,".dar");
     fp=fopen(fname,"rb");
     fread(trx_pos,sizeof(trx_pos),1,fp);
     fread(try_pos,sizeof(try_pos),1,fp);
     fread(trz_pos,sizeof(trz_pos),1,fp);
     fclose(fp);
     /*input upper wing point*/
     strcpy(fname,name);
     strcat(fname,".dwu");
     if((fp=fopen(fname,"r"))!=NULL){
        fscanf(fp,"%lf,%lf,%lf",&wu_xpos,&wu_ypos,&wu_zpos);
        fclose(fp);}
     /*input lower wing point*/
     strcpy(fname,name);
     strcat(fname,".dwl");
     if((fp=fopen(fname,"r"))!=NULL){
        fscanf(fp,"%lf,%lf,%lf",&wl_xpos,&wl_ypos,&wl_zpos);
        fclose(fp);}
    /*fprintf(stdprn,"%c&l32259.1057J",27);*/
    /*fprintf(stdprn,"PG;");*/
    /*fflush(stdprn);   */
    /*initialise plotter*/
    fprintf(stdprn,"IN;SP1;SC0,1000,0,720;DT.;");
    /* draw box */
    /* total box size = 1000 x 720, map box = 720 x 720          */
```

```
    /* map box central coordinate = 640,360                          */
    /* every 100 pixels = 1mm, therefore every pixel = 10 microns*/
    fprintf(stdprn,"PU;PA0,0;PD;PA0,720;PA1000,720;PA1000,0;PA0,0;PU;");
    /*finish patient box*/
    fprintf(stdprn,"PA280,720;PD;PA280,0;PU;");
    /*print details*/

    fprintf(stdprn,"PA10,680;LBFILENAME: %s.;",fname);
    fprintf(stdprn,"PA540,680;LBCORNEAL TOPOGRAPHY.;");
    /*PLOT POINTS*/
    fprintf(stdprn,"DT*;SR0.2,0.5;");

    for(i=0;i<POINTS;i++)
       for(j=0;j<PLANES;j++){
           if(tlx_pos[i][j]!=0.0){
             fprintf(stdprn,"PA%d,%d;LB*;",(int)(tly_pos[i][j]*100.0+
640.0),(int)(tlz_pos[i][j]*100.0+360.0));
             sprintf(depth,"%8.3f",tlx_pos[i][j]-142.0);
             fprintf(stdprn,"PA%d,%d;DT,;LB%s,;DT*;",(int)(tly_pos[i][j]*
100.0+625.0),(int)(tlz_pos[i][j]*100.0+355.0),depth);
             sprintf(depth,"l(%d.%d)",i,j);
             fprintf(stdprn,"PA%d,%d;DT,;LB%s,;DT*;",(int)(tly_pos[i][j]*
100.0+633.0),(int)(tlz_pos[i][j]*100.0+350.0),depth);
             }
           if(trx_pos[i][j]!=0.0){
             fprintf(stdprn,"PA%d,%d;LB*;",(int)(try_pos[i][j]*100.0+
640.0),(int)(trz_pos[i][j]*100.0+360.0));
             sprintf(depth,"%8.3f",trx_pos[i][j]-142.0);
             fprintf(stdprn,"PA%d,%d;DT,;LB%s,;DT*;",(int)(try_pos[i][j]*
100.0+625.0),(int)(trz_pos[i][j]*100.0+355.0),depth);
             sprintf(depth,"r(%d.%d)",i,j);
             fprintf(stdprn,"PA%d,%d;DT,;LB%s,;DT*;",(int)(try_pos[i][j]*
100.0+633.0),(int)(trz_pos[i][j]*100.0+350.0),depth);             }
       }
    /*plot wing points if they exist*/
    if(wu_xpos!=0.0){
             fprintf(stdprn,"PA%d,%d;LB*;",(int)(wu_ypos*100.0+640.0),
(int)(wu_zpos*100.0+360.0));
             sprintf(depth,"%8.3f",wu_xpos-142.0);
             fprintf(stdprn,"PA%d,%d;DT,;LB%s,;DT*;",(int)(wu_ypos*
100.0+625.0),(int)(wu_zpos*100.0+350.0),depth);
             }
    if(wl_xpos!=0.0){
             fprintf(stdprn,"PA%d,%d;LB*;",(int)(wl_ypos*100.0+640.0),
(int)(wl_zpos*100.0+360.0));
             sprintf(depth,"%8.3f",wl_xpos-142.0);
             fprintf(stdprn,"PA%d,%d;DT,;LB%s,;DT*;",(int)(wl_ypos*
100.0+625.0),(int)(wl_zpos*100.0+350.0),depth);
             }
    fprintf(stdprn,"DT.;SR;");
      /*PLOT LOCATION GRID*/
    fprintf(stdprn,"PA340,360;PD;PA940,360;PU;");
    fprintf(stdprn,"PA340,360;XT;PA440,360;XT;PA540,360;XT;");
    fprintf(stdprn,"PA640,360;XT;PA740,360;XT;PA840,360;XT;PA940,360;XT;");
    fprintf(stdprn,"PA640,360;CI100;CI300;");
    /*change test size*/
    fprintf(stdprn,"SR0.5,0.8;DT;");
```

```
    fprintf(stdprn,"PA940,330;LB3.0 mm%c;DT.;",3);

    fprintf(stdprn,"PG;");
    fflush(stdprn);
    }
contours(xmin)double xmin;{
    char pause,opt,fname[10],str[12],vstr[10],ustr[10];
    int uaxis,vaxis,i;
    /*double xmin=142.50;*/
    double u,v,rv[ORDER],cv[ORDER],x,y,z,oldinc,inc=0.05,error=0.005;
    cls();
    do{
    printf("\nprint u,v co_ordinates ?  y/n :");
    pause=getch();
    printf("%c\n",pause);
    printf("\ninput contour interval in mm [%lf] : ",inc);
    gets(str);
    oldinc=inc;
    inc=(double)atof(str);
    if(inc==0.0){inc=oldinc;
                 sprintf(str,"%6.1lf",inc*1000);}
    fprintf(stdprn,"IN;SP1;SC0,1000,0,720;DT.;");
    /* draw box */
    /* total box size = 1000 x 720, map box = 720 x 720          */
    /* map box central coordinate = 640,360                      */
    /* every 100 pixels = 1mm, therefore every pixel = 10 microns*/
    fprintf(stdprn,"PU;PA0,0;PD;PA0,720;PA1000,720;PA1000,0;PA0,0;PU;");
    /*finish patient box*/
    fprintf(stdprn,"PA280,720;PD;PA280,0;PU;");
    /*print details*/
    strcpy(fname,name);
    fprintf(stdprn,"PA10,680;LBFILENAME: %s.;",fname);
    fprintf(stdprn,"PA470,680;LBCORNEAL TOPOGRAPHY:CONTOUR PLOT.;");
    /*PLOT POINTS*/
    fprintf(stdprn,"DT*;SR0.2,0.5;");
       /*build patch for different values of u and v*/
       u=v=0.0;
       for(uaxis=0;uaxis<200;uaxis++){
          u=0.0+(double)uaxis*0.005;
          for(vaxis=0;vaxis<200;vaxis++){
             v=0.0+(double)vaxis*0.005;

             /*construct surface patch*/
             /*column and row vectors*/
             rv[0]=u*u;
             rv[1]=u;
             rv[2]=1.0;
             cv[0]=v*v;
             cv[1]=v;
             cv[2]=1.0;

             /*calculate x,y,z*/
             x=matrix(rv,px,cv);
             y=matrix(rv,py,cv);
             z=matrix(rv,pz,cv);

             for(i=0;i<8;i++){
```

```
                    if((x>((xmin-error)+inc*i)) && (x<((xmin+error)+inc*i)))
                    fprintf(stdprn,"PA%d,%d;LB*;",(int)(y*100.0+640.0),
 (int)(z*100.0+360.0));
                    }

                 }
              }

   if(pause=='y'){
      u=v=0.0;
      for(uaxis=0;uaxis<200;uaxis+=20){
         u=0.0+(double)uaxis*0.005;
         sprintf(ustr,"%d",uaxis);
         for(vaxis=0;vaxis<200;vaxis+=20){
            v=0.0+(double)vaxis*0.005;
            sprintf(vstr,"%d",vaxis);

            /*construct surface patch*/
            /*column and row vectors*/
            rv[0]=u*u;
            rv[1]=u;
            rv[2]=1.0;
            cv[0]=v*v;
            cv[1]=v;
            cv[2]=1.0;

            /*calculate x,y,z*/
            x=matrix(rv,px,cv);
            y=matrix(rv,py,cv);
            z=matrix(rv,pz,cv);
            fprintf(stdprn,"PA%d,%d;SR0.7,1.0;DTX;LBX;SR0.2,0.5;DT*;",
 (int)(y*100.0+640.0),(int)(z*100.0+360.0));
            fprintf(stdprn,"PA%d,%d;SR0.3,0.6;DT);LB(%s,%s);SR0.2,0.5;
 DT*;",(int)(y*100.0+625.0),(int)(z*100.0+350.0),ustr,vstr);

            }
         }
      }/*end if*/

    fprintf(stdprn,"DT.;SR;");
    fprintf(stdprn,"PA10,620;LBCONTOURS microns:%s.;",str);
      /*PLOT LOCATION GRID*/
    fprintf(stdprn,"PA340,360;PD;PA940,360;PU;");
    fprintf(stdprn,"PA340,360;XT;PA440,360;XT;PA540,360;XT;");
    fprintf(stdprn,"PA640,360;XT;PA740,360;XT;PA840,360;XT;PA940,360;XT;");
    fprintf(stdprn,"PA640,360;CI100;CI300;");
    /*change test size*/
    fprintf(stdprn,"SR0.5,0.8;DT;");
    fprintf(stdprn,"PA940,330;LB3.0 mm%c;DT.;",3);

    fprintf(stdprn,"PG;");
    fflush(stdprn);
    printf("\nplot another map with different contour increment y/n? : ");
    opt=getch();

    }while(opt=='y');
}
```

```
con4(xmin,apex_u,apex_v,apex_y,apex_z)
    double xmin;int apex_u,apex_v;double apex_y,apex_z;{
    char pause,opt,fname[10],str[12],cstr[10],apexstr[10],cur15[10],cur3[10];
    char flatstr[10],sphstr[10],radstr[10],apowstr[10],afstr[10];
    int uaxis,vaxis,i,j,uelement,velement,count15,count3;
    double apex=0.0,curve15=0.0,curve3=0.0,ydist,zdist,dist,flatt=0.0;
    double apow=0.0,af=0.0,sphrad=0.0,spharea=0.0;
    /*double xmin=142.50;*/
    double u,v,rv[ORDER],cv[ORDER],x,y,z,oldinc,inc=0.05,error=0.005;
    cls();
    do{
    printf("\nprint curvatures ?  y/n :");
    pause=getch();
    printf("%c\n",pause);
    printf("\ninput contour interval in mm [%lf] : ",inc);
    gets(str);
    oldinc=inc;
    inc=(double)atof(str);
    if(inc==0.0){inc=oldinc;
                 sprintf(str,"%6.1lf",inc*1000);}
    fprintf(stdprn,"IN;SP1;SC0,1000,0,720;DT.;");
    /* draw box */
    /* total box size = 1000 x 720, map box = 720 x 720          */
    /* map box central coordinate = 640,360                      */
    /* every 100 pixels = 1mm, therefore every pixel = 10 microns*/
    fprintf(stdprn,"PU;PA0,0;PD;PA0,720;PA1000,720;PA1000,0;PA0,0;PU;");
    /*finish patient box*/
    fprintf(stdprn,"PA280,720;PD;PA280,0;PU;");
    /*print details*/
    strcpy(fname,name);
    fprintf(stdprn,"PA10,680;LBFILENAME: %s.;",fname);
    fprintf(stdprn,"PA450,680;LBCORNEAL TOPOGRAPHY:EDMUND MODEL.;");
    /*PLOT POINTS*/
    fprintf(stdprn,"DT*;SR0.2,0.5;");
       /*build patch for different values of u and v*/
       u=v=0.0;
       for(uaxis=0;uaxis<200;uaxis++){
          u=0.0+(double)uaxis*0.005;
          for(vaxis=0;vaxis<200;vaxis++){
             v=0.0+(double)vaxis*0.005;

             /*construct surface patch*/
             /*column and row vectors*/
             rv[0]=u*u;
             rv[1]=u;
             rv[2]=1.0;
             cv[0]=v*v;
             cv[1]=v;
             cv[2]=1.0;

             /*calculate x,y,z*/
             x=matrix(rv,px,cv);  /*pxrect*/
             y=matrix(rv,py,cv);
             z=matrix(rv,pz,cv);
             if((uaxis==apex_u)&&(vaxis==apex_v))
                 fprintf(stdprn,"PA%d,%d;DTA;SR0.8,1.0;LBA;DT*;SR0.2,0.5;"
,(int)(y*100.0+640.0),(int)(z*100.0+360.0));
```

```
            if(!((uaxis>apex_u-5)&&(uaxis<apex_u+5)&&(vaxis>apex_v-5)
&&(vaxis<apex_v+5)))
                for(i=0;i<8;i++){
                                if((x>((xmin-error)+inc*i)) &&
(x<((xmin+error)+inc*i)))
                                fprintf(stdprn,"PA%d,%d;LB*;",
(int)(y*100.0+640.0),(int)(z*100.0+360.0));
                                }
            }
         }

     u=v=curve15=curve3=0.0;
     count15=count3=0;
     for(uaxis=0;uaxis<200;uaxis+=20){
        uelement=uaxis/10;
        u=0.0+(double)uaxis*0.005;
        for(vaxis=0;vaxis<200;vaxis+=20){
           velement=vaxis/10;
           v=0.0+(double)vaxis*0.005;


           /*construct surface patch*/
           /*column and row vectors*/
           rv[0]=u*u;
           rv[1]=u;
           rv[2]=1.0;
           cv[0]=v*v;
           cv[1]=v;
           cv[2]=1.0;

           /*calculate x,y,z*/
           x=matrix(rv,px,cv);/*pxrect*/
           y=matrix(rv,py,cv);
           z=matrix(rv,pz,cv);
           ydist=(y-apex_y)*(y-apex_y);
           zdist=(z-apex_z)*(z-apex_z);
           dist=sqrt(ydist+zdist);
           if((pause=='y')&&(dist<1.5)){
             sprintf(cstr,"%6.2lf",curve[uelement][velement]+CONSTA-
R_FACTOR*dist);
             fprintf(stdprn,"PA%d,%d;SR0.7,1.0;DTX;LBX;SR0.2,0.5;DT*;",
(int)(y*100.0+640.0),(int)(z*100.0+360.0));
             fprintf(stdprn,"PA%d,%d;SR0.3,0.6;DT);LB(%s);SR0.2,0.5;DT*;
",(int)(y*100.0+630.0),(int)(z*100.0+350.0),cstr);
             }

          if(dist<1.5){count3++;
                       curve3+=curve[uelement][velement];}
          if(dist<0.75){count15++;
                       curve15+=curve[uelement][velement];}

          }
       }
     apex=sqrt(apex_y*apex_y+apex_z*apex_z);
     curve3=curve3/count3;
     curve15=curve15/count15;
     flatt=((curve3+CONST3)-(curve15+CONST15))/1.5;
```

```
        if(flatt<0.01)flatt=0.001;
        sphrad=0.1/flatt;
        spharea=sphrad*sphrad*3.142;
        apow=337.5/(curve15+CONST15);
        af=337.5*((1.0/(curve15+CONST15))-(1.0/(curve15+CONST15+flatt*1.5)));
        sprintf(apexstr,"%7.2lf",apex);
        sprintf(cur3,"%6.2lf",curve3+CONST3);
        sprintf(cur15,"%6.2lf",curve15+CONST15);
        sprintf(flatstr,"%7.2lf",flatt);
        sprintf(radstr,"%7.2lf",sphrad);
        sprintf(sphstr,"%7.2lf",spharea);
        sprintf(apowstr,"%7.2lf",apow);
        sprintf(afstr,"%7.2lf",af);
      fprintf(stdprn,"DT.;SR;");
      fprintf(stdprn,"PA10,620;LBCONTOURS microns:%s.;",str);
      fprintf(stdprn,"DT;PA10,560;LBApex dist(mm):%s;",apexstr);
      fprintf(stdprn,"PA10,500;LBApex R.of.C (mm):%s;",cur15);
      fprintf(stdprn,"PA10,440;LBApex Power (D):%s;",apowstr);
      fprintf(stdprn,"PA10,380;LBA.F.(D/mm):%s;",afstr);
        /*PLOT LOCATION GRID*/
      fprintf(stdprn,"PA340,360;PD;PA940,360;PU;");
      fprintf(stdprn,"PA340,360;XT;PA440,360;XT;PA540,360;XT;");
      fprintf(stdprn,"PA640,360;XT;PA740,360;XT;PA840,360;XT;PA940,360;XT;");
      fprintf(stdprn,"PA640,360;CI100;CI300;");
      /*change test size*/
      fprintf(stdprn,"SR0.5,0.8;DT;");
      fprintf(stdprn,"PA940,330;LB3.0 mm%c;DT.;",3);

      fprintf(stdprn,"PG;");
      fflush(stdprn);
      printf("\nplot another map with different contour increment y/n? : ");
      opt=getch();

    }while(opt=='y');
}

surface(){
     int i,j;
     short pixelvalue=15;
     char pause,fname[10],depth[10];
     /* matrices of points saved on disk */
     static double trx_pos[POINTS][PLANES],try_pos[POINTS][PLANES],
trz_pos[POINTS][PLANES];
     static double tlx_pos[POINTS][PLANES],tly_pos[POINTS][PLANES],
tlz_pos[POINTS][PLANES];
     /*wing coordinates*/
     double wu_xpos=0.0,wu_ypos,wu_zpos,wl_xpos=0.0,wl_ypos,wl_zpos;
     /*input data file*/
     _registerfonts("*.FON");
     cls();
     printf("\n file name : ");
     gets(name);
     strcpy(fname,name);
     /*input left side points*/
     strcat(fname,".dal");
     fp=fopen(fname,"rb");
     fread(tlx_pos,sizeof(tlx_pos),1,fp);
```

```
fread(tly_pos,sizeof(tly_pos),1,fp);
fread(tlz_pos,sizeof(tlz_pos),1,fp);
fclose(fp);
/*input right point file */
strcpy(fname,name);
strcat(fname,".dar");
fp=fopen(fname,"rb");
fread(trx_pos,sizeof(trx_pos),1,fp);
fread(try_pos,sizeof(try_pos),1,fp);
fread(trz_pos,sizeof(trz_pos),1,fp);
fclose(fp);
/*input upper wing point*/
strcpy(fname,name);
strcat(fname,".dwu");
if((fp=fopen(fname,"r"))!=NULL){
   fscanf(fp,"%lf,%lf,%lf",&wu_xpos,&wu_ypos,&wu_zpos);
   fclose(fp);}
/*input lower wing point*/
strcpy(fname,name);
strcat(fname,".dwl");
if((fp=fopen(fname,"r"))!=NULL){
   fscanf(fp,"%lf,%lf,%lf",&wl_xpos,&wl_ypos,&wl_zpos);
   fclose(fp);}
_setvideomode(_VRES16COLOR);
_setviewport(0,0,640,480);
_setwindow(TRUE,-5.0,-5.0,5.0,5.0);
_setbkcolor(_BRIGHTWHITE);
_remappalette(15,_BLACK);
_setcolor(15);
_setfont("courbh16w12");
_moveto_w(-5.0,0.0);
_lineto_w(5.0,0.0);
_moveto_w(0.0,5.0);
_lineto_w(0.0,-5.0);
_moveto_w(0.1,5.0);
_outgtext("z");
_moveto_w(-5.0,-0.1);
_outgtext("y");
_moveto_w(-5.0,5.0);
_outgtext("POINT PLOT");
_setfont("courbh8w6");
for(i=0;i<POINTS;i++)
   for(j=0;j<PLANES;j++)
       {

      if(tlx_pos[i][j]!=0.0){
         _setpixel_w(tly_pos[i][j],tlz_pos[i][j]);
          sprintf(depth,"%8.3f",tlx_pos[i][j]-142.0);
          _moveto_w(tly_pos[i][j]-0.25,tlz_pos[i][j]-0.1);
          _outgtext(depth);
          }
      if(trx_pos[i][j]!=0.0){
          _setpixel_w(try_pos[i][j],trz_pos[i][j]);
          sprintf(depth,"%8.3f",trx_pos[i][j]-142.0);
          _moveto_w(try_pos[i][j]-0.25,trz_pos[i][j]-0.1);
          _outgtext(depth);}
       }
```

```
        /*plot wing points if they exist*/
             if(wu_xpos!=0.0){
                 _setpixel_w(wu_ypos,wu_zpos);
               sprintf(depth,"%8.3f",wu_xpos-142.0);
               _moveto_w(wu_ypos-0.25,wu_zpos-0.1);
               _outgtext(depth);
               }
             if(wl_xpos!=0.0){
                 _setpixel_w(wl_ypos,wl_zpos);
               sprintf(depth,"%8.3f",wl_xpos-142.0);
               _moveto_w(wl_ypos-0.25,wl_zpos-0.1);
               _outgtext(depth);
               }

     pause=getch();

     _setvideomode(_DEFAULTMODE);
     }
write_cross(last,i,line_step,sl,colour)
     int last,i,line_step,sl,colour;{
     ofg_cline(0,last-2,i*line_step+sl,last+2,i*line_step+sl,colour);
     ofg_cline(0,last,i*line_step+sl-2,last,i*line_step+sl+2,colour);
     }

centre(){
      char pause,fname[12];
      int l=5,r=760,m=5,centre_x,centre_y;
      fcx=fcy=0.0;
      /*set lut 15 to 255*/
      ofg_clearlut(OUTPUT,15,255);
      /*set dynamic lut mode*/
      ofg_dlutmode(DYNAMIC);
      ofg_setframe(0);
      cls();
      printf("\n Press 'i' to move in, 'o' to move out, any key to end");
      ofg_cline(0,l,0,l,512,15);
      do{
         pause=getch();
         ofg_cvline(gaoi,l,0,512,0);
         if(pause=='i')l+=2;
         if(pause=='o')l-=2;
         ofg_cline(0,l,0,l,512,15);
         }while(pause=='i' | pause=='o');
      ofg_cline(0,r,0,r,512,15);
      do{
         pause=getch();
         ofg_cvline(gaoi,r,0,512,0);
         if(pause=='i')r-=2;
         if(pause=='o')r+=2;
         ofg_cline(0,r,0,r,512,15);
         }while(pause=='i' | pause=='o');
       cls();
       printf("\npress 'd' to move down, 'u' to move up, any key to end");
       ofg_cline(0,0,m,768,m,15);
      do{
         pause=getch();
         ofg_chline(gaoi,0,m,1,0);
```

```
        ofg_chline(gaoi,l+1,m,r-l-1,0);
        ofg_chline(gaoi,r+1,m,767-r,0);
        if(pause=='d')m+=2;
        if(pause=='u')m-=2;
        ofg_cline(0,0,m,768,m,15);
        }while(pause=='d' | pause=='u');
      /*calculate distance shift from centre*/
      /*centre x = (r+l)/2, centre y = m */
      centre_x=((r+l)/2)-384;
      centre_y=m-256;
      /*calculate value in mm*/
      fcx=(float)centre_x*dist_per_pixel*0.001;
      fcy=(float)centre_y*dist_per_pixel*0.001;
      printf("\nx = %f mm, y = %f mm",fcx,fcy);
      pause=getch();
      clear_overlay();
      }/*end proc*/


limits(){
      char pause;
      ofg_clearlut(OUTPUT,15,255);
      ofg_dlutmode(DYNAMIC);
      ofg_setframe(0);
      do{
         cls();
         printf("\npress 'd' to move down, 'u' to move up, any key to end");
         printf("\n\ntop_mark = %d",ftop);
         ofg_cline(0,0,ftop,768,ftop,15);
         pause=getch();
         ofg_chline(gaoi,0,ftop,768,0);
         if(pause=='d' && ftop<500)ftop+=2;
         if(pause=='u'&& ftop>2)ftop-=2;
         ofg_cline(0,0,ftop,768,ftop,15);
         }while(pause=='d' | pause=='u');
      do{
         cls();
         printf("\npress 'd' to move down, 'u' to move up, any key to end");
         printf("\n\nbottom_mark = %d",fbottom);
         ofg_cline(0,0,fbottom,768,fbottom,15);
         pause=getch();
         ofg_chline(gaoi,0,fbottom,768,0);
         if(pause=='d' && fbottom<508)fbottom+=2;
         if(pause=='u'&& fbottom>(ftop+1))fbottom-=2;
         ofg_cline(0,0,fbottom,768,fbottom,15);
         }while(pause=='d' | pause=='u');
         clear_overlay();
}


l_extract(){
      int x[POINTS][PLANES],y[POINTS][PLANES],i,j,last_point,last;
      int line_step=50,sl=100;
      char choose,pause,line[100],wfname[10],fname[10];
      double l_cam=-7743.11,l;
      static double lx_pos[POINTS][PLANES],ly_pos[POINTS][PLANES],
lz_pos[POINTS][PLANES];
```

```
double tan_phi=1.84,sep=118.5,deltah=0.0,syserror=0.0;
double w_xpos,w_ypos,w_zpos;
float Lminusl[13];
float yp=0.0,zp=0.0,t=0.0,x_coord=0.0,y_coord=0.0,z_coord=0.0;
/*read calibrated values*/
       strcpy(fname,name);
       strcat(fname,".lcb");
       if( (fp = fopen(fname,"r"))==NULL){
     printf("\ncannot open %s file",fname);
     printf("\npress any key to continue");
     pause=getch();
     return(-1);}
fscanf(fp,"%lf,%lf,%lf,%lf,%lf",&tan_phi,&sep,&l,&deltah,&syserror);
fclose(fp);
l_cam=l;
ofg_clearlut(OUTPUT,15,255);
ofg_dlutmode(DYNAMIC);
ofg_setframe(0);
line_step=(fbottom-ftop)/(POINTS-1);
sl=ftop-line_step;
cls();
printf("\nf..forward, b..back, p..place point, every %1d bands",BAND);
for(i=1;i<POINTS+1;i++){
    last_point=390;
    last=390;
    write_cross(last,i,line_step,sl,15);
    for(j=0;j<PLANES;j++){
    do{
        choose=getch();
                switch(choose){
                       case 'f':write_cross(last,i,line_step,sl,0);
                                last--;
                                write_cross(last,i,line_step,sl,15);
                                break;
                       case 'b':if(last>=last_point-1)break;
                                write_cross(last,i,line_step,sl,0);
                                last++;
                                write_cross(last,i,line_step,sl,15);
                                break;
                       }/*end case*/
                }while(choose!='p');
                write_cross(last,i,line_step,sl,15);
                last_point=last;
                x[i-1][j]=last;
                y[i-1][j]=i*line_step+sl;
                last-=4;
           }/*next j*/
      }/*next i*/

      /*****calibration factors*****/
      printf("\n press any key");
      pause=getch();
      Lminusl[0]=150000.0;/*distance to first plane in microns*/
      cls();
      printf("\n\nvalues as stored");
      printf("\ntan phi = %lf",tan_phi);
      printf("\nl=%lf,l_cam=%lf",l,l_cam);
```

```
            printf("\nseperation along axis in mm= %lf",sep*0.001);
            printf("\ndistance to first plane in mm= %f",Lminusl[0]*0.001);
            for(i=1;i<13;i++)Lminusl[i]=Lminusl[0]-(float)sep*(float)i;

            /*print results on screen*/

            /*set of POINTS for each plane*/
            for(j=0;j<PLANES;j++){/******planes******/
               for(i=0;i<POINTS;i++){/******points on plane*****/
                   yp=(float)(384-x[i][j]);
                   zp=(float)(256-y[i][j]);
                   /*all plane measured*/
                   t=Lminusl[j*BAND]*tan_phi/((float)l_cam*tan_phi+yp);
                   x_coord=(float)l_cam*(t+1.0)*0.001;
                   y_coord=yp*t*0.001;
                   z_coord=-zp*t*0.001;
                   lx_pos[i][j]=(double)x_coord+deltah+(syserror*
(double)yp);
                   ly_pos[i][j]=(double)y_coord-(double)fcx;
                   lz_pos[i][j]=(double)z_coord+(double)fcy;
                   /*coords multiplied by 0.001 to convert to mm*/
                   }/*next point*/
               }/*next plane*/

            /**correct for lost points**/
            /**these are set to zero **/
            /** and correct for depth error **/

            for(i=0;i<POINTS;i++)
                for(j=0;j<PLANES;j++)
                   lx_pos[i][j]=lx_pos[i][j]+(centrx-
lx_pos[centline][0]);
            for(i=0;i<POINTS;i++){
                for(j=1;j<PLANES;j++){
                   lx_pos[i][j]=lx_pos[i][j]+(centrx-
lx_pos[centline][0]);
                   if(x[i][j]>(x[i][j-1]-5)){
                                       lx_pos[i][j]=0.0;
                                       ly_pos[i][j]=0.0;
                                       lz_pos[i][j]=0.0;}

                   }
                }

           printf("\n press any key");
           pause=getch();
           cls();
                      for(i=0;i<POINTS;i++){printf("\n");
                          for(j=PLANES-1;j>=0;j--){
                              printf("x");
                              printf("[%7.3lf] ",lx_pos[i][j]);}
                              printf("\n");

                      }
           printf("\nsend to printer? y/n");
           pause=getch();
           if(pause=='y'){
```

```
                            fprintf(stdprn,"\r\n\n\n");
                            fprintf(stdprn,"\r\n              LEFT
measurement results");
                            for(i=0;i<POINTS;i++){fprintf(stdprn,"\n\n\n\r");
                                for(j=PLANES-1;j>=0;j--){
                                    fprintf(stdprn,"x");
                                    fprintf(stdprn,"[%7.3lf] ",lx_pos[i][j]);}
                                    fprintf(stdprn,"\n\r");

                                }
                      fprintf(stdprn,"\f");
                      fflush(stdprn);
                      }
                    ofg_dlutmode(STATIC);
                    ofg_setframe(I);

/*save matrix on file*/
printf("\n%csave matrix on file ? y/n",7);
pause=getch();
if(pause=='y'){
    printf("\n file name : ");
    gets(wfname);
    strcpy(fname,wfname);
    strcat(fname,".dal");
    fp=fopen(fname,"wb");
    fwrite(lx_pos,sizeof(lx_pos),1,fp);
    fwrite(ly_pos,sizeof(ly_pos),1,fp);
    fwrite(lz_pos,sizeof(lz_pos),1,fp);
    fclose(fp);
    }

  if(pause=='y'){
    /*interpolation for upper wing points*/
    w_xpos=(lx_pos[0][0]+rightup[0])/2.0;
    w_ypos=(ly_pos[0][0]+rightup[1])/2.0;
    w_zpos=(lz_pos[0][0]+rightup[2])/2.0;
    strcpy(fname,wfname);
    strcat(fname,".dwu");
    fp=fopen(fname,"w");
    fprintf(fp,"%lf,%lf,%lf ",w_xpos,w_ypos,w_zpos);
    close(fp);
    /*interpolation for lower wing points*/
    w_xpos=(lx_pos[POINTS-1][0]+rightdown[0])/2.0;
    w_ypos=(ly_pos[POINTS-1][0]+rightdown[1])/2.0;
    w_zpos=(lz_pos[POINTS-1][0]+rightdown[2])/2.0;
    strcpy(fname,wfname);
    strcat(fname,".dwl");
    fp=fopen(fname,"w");
    fprintf(fp,"%lf,%lf,%lf ",w_xpos,w_ypos,w_zpos);
    close(fp);
    }/*end if*/
        }/*end proc*/
r_extract(){
     int x[POINTS][PLANES],y[POINTS][PLANES],i,j,last_point,last;
     int line_step=50,sl=100,u_wing_yp,u_wing_zp,l_wing_yp,l_wing_zp;
     char choose,pause,line[100],fname[10];
     double l_cam=-7743.11,l;
```

```
      static double rx_pos[POINTS][PLANES],ry_pos[POINTS][PLANES],
rz_pos[POINTS][PLANES];
      double tan_phi=1.84,sep=118.5,deltah=0.0,syserror=0.0;
      double Lminusl[13];
      double yp=0.0,zp=0.0,t=0.0,x_coord=0.0,y_coord=0.0,z_coord=0.0;
      double uwyp,uwzp,lwyp,lwzp,wuxc,wuyc,wuzc;
      double wlxc,wlyc,wlzc;
      double ul,ll,ut,lt;
      double wnum_u,wnum_l;
      double wu_xpos,wu_ypos,wu_zpos,wl_xpos,wl_ypos,wl_zpos;
      /*read calibrated values*/
      cls();
         strcpy(fname,name);
         strcat(fname,".rcb");
         if( (fp = fopen(fname,"r"))==NULL){
           printf("\ncannot open %s file",fname);
           printf("\npress any key to continue");
           pause=getch();
           return(-1);}
      fscanf(fp,"%lf,%lf,%lf,%lf,%lf",&tan_phi,&sep,&l,&deltah,&syserror);
      fclose(fp);
      l_cam=l;
      ofg_clearlut(OUTPUT,15,255);
      ofg_dlutmode(DYNAMIC);
      ofg_setframe(0);
      line_step=(fbottom-ftop)/(POINTS-1);
      sl=ftop-line_step;
      cls();
      printf("\nf..forward, b..back, p..place point, every %1d bands",BAND);
      for(i=1;i<POINTS+1;i++){
          last_point=380;
          last=380;
          write_cross(last,i,line_step,sl,15);
          for(j=0;j<PLANES;j++){
          do{
              choose=getch();
                       switch(choose){
                             case 'f':write_cross(last,i,line_step,sl,0);
                                      last++;
                                      write_cross(last,i,line_step,sl,15);
                                      break;
                             case 'b':if(last==last_point+1)break;
                                      write_cross(last,i,line_step,sl,0);
                                      last--;
                                      write_cross(last,i,line_step,sl,15);
                                      break;
                             }/*end case*/
                       }while(choose!='p');
                       write_cross(last,i,line_step,sl,15);
                       last_point=last;
                       x[i-1][j]=last;
                       y[i-1][j]=i*line_step+sl;
                       last+=5;
                  }/*next j*/
             }/*next i*/

             /*****calibration factors*****/
```

```
                printf("\n press any key");
                pause=getch();
                Lminus1[0]=150000.0;/*distance to first plane in microns*/
                cls();
                printf("\n\nvalues as stored");
                printf("\ntan phi = %lf",tan_phi);
                printf("\nl=%lf,l_cam=%lf",l,l_cam);
                printf("\nseperation along axis in mm= %lf",sep*0.001);
                printf("\ndistance to first plane in mm= %lf",Lminus1[0]*
0.001);
                for(i=1;i<13;i++)Lminus1[i]=Lminus1[0]-(double)sep*(double)i;

                /*print results on screen*/

                /*set of POINTS for each plane*/
                for(j=0;j<PLANES;j++){/******planes******/
                   for(i=0;i<POINTS;i++){/******points on plane*****/
                        yp=(double)(x[i][j]-384);
                        zp=(double)(256-y[i][j]);
                        /*all plane measured*/
                        t=Lminus1[j*BAND]*tan_phi/((double)l_cam*tan_phi+yp);
                        x_coord=(double)l_cam*(t+1.0)*0.001;
                        y_coord=-yp*t*0.001;
                        z_coord=-zp*t*0.001;
                        rx_pos[i][j]=(double)x_coord+deltah+(syserror*
(double)yp);
                        ry_pos[i][j]=(double)y_coord-(double)fcx;
                        rz_pos[i][j]=(double)z_coord+(double)fcy;
                        /*coords multiplied by 0.001 to convert to mm*/
                        }/*next point*/
                   }/*next plane*/

                /** save central point for calibration correction **/
                /** with left points (central depth) **/
                centrx=rx_pos[centline][0];

                /**correct for lost points**/
                /**these are set to zero**/
                for(i=0;i<POINTS;i++){
                    for(j=1;j<PLANES;j++){
                        if(x[i][j]<(x[i][j-1]+6)){
                                        rx_pos[i][j]=0.0;
                                        ry_pos[i][j]=0.0;
                                        rz_pos[i][j]=0.0;}
                        }
                    }

                /*save top and bottom (0,0) points for central 'wing'
interpolation*/
                rightup[0]=rx_pos[0][0];
                rightup[1]=ry_pos[0][0];
                rightup[2]=rz_pos[0][0];
                rightdown[0]=rx_pos[POINTS-1][0];
                rightdown[1]=ry_pos[POINTS-1][0];
                rightdown[2]=rz_pos[POINTS-1][0];

                printf("\n press any key");
```

```
                pause=getch();
                cls();
                        for(i=0;i<POINTS;i++){printf("\n");
                            for(j=0;j<PLANES;j++){
                                printf("x");
                                printf("[%7.3lf] ",rx_pos[i][j]);}
                                printf("\n");

                        }
                printf("\nsend to printer? y/n");
                pause=getch();
                if(pause=='y'){
                        fprintf(stdprn,"\r\n\n\n");
                        fprintf(stdprn,"\r\n          RIGHT measurement
results");
                        for(i=0;i<POINTS;i++){fprintf(stdprn,"\n\n\n\r");
                            for(j=0;j<PLANES;j++){
                                fprintf(stdprn,"x");
                                fprintf(stdprn,"[%7.3lf] ",
rx_pos[i][j]);}
                                fprintf(stdprn,"\n\r");

                            }
                  fprintf(stdprn,"\f");
                  fflush(stdprn);
                  }

/*save matrix on file*/
printf("\n%csave matrix on file ? y/n",7);
pause=getch();
if(pause=='y'){
    printf("\n file name : ");
    gets(fname);
    strcat(fname,".dar");
    fp=fopen(fname,"wb");
    fwrite(rx_pos,sizeof(rx_pos),1,fp);
    fwrite(ry_pos,sizeof(ry_pos),1,fp);
    fwrite(rz_pos,sizeof(rz_pos),1,fp);
    fclose(fp);
    }
    ofg_clf(gaoi,0);
    cls();
    printf("\n get upper wing point ? y/n");
    pause=getch();
    if (pause=='y'){

      u_wing_zp=2;
      u_wing_yp=300;
         cls();
         printf("\n Press 'i' to move in, 'o' to move out, any key to end");
         ofg_cline(0,u_wing_yp,0,u_wing_yp,512,15);
         do{
            pause=getch();
            ofg_cvline(gaoi,u_wing_yp,0,512,0);
            if(pause=='i')u_wing_yp++;
            if(pause=='o')u_wing_yp--;
            ofg_cline(0,u_wing_yp,0,u_wing_yp,512,15);
```

```
            }while(pause=='i' | pause=='o');
         do{
            cls();
            printf("\npress 'd' to move down, 'u' to move up, any key to
end");
            printf("\n\nmark = %d",u_wing_zp);
            ofg_cline(0,0,u_wing_zp,768,u_wing_zp,15);
            pause=getch();
            ofg_chline(gaoi,0,u_wing_zp,768,0);
            if(pause=='d' && u_wing_zp<500)u_wing_zp++;
            if(pause=='u'&& u_wing_zp>2)u_wing_zp--;
            ofg_cline(0,0,u_wing_zp,768,u_wing_zp,15);
          }while(pause=='d' | pause=='u');
          printf("\ninput wing number , counting right to left ");
          gets(line);
          wnum_u=atof(line);
         /*calculate x,y,z*/
         uwyp=(double)(u_wing_yp-384);
         uwzp=(double)(256-u_wing_zp);
         ul=Lminusl[0]+(double)sep*(double)wnum_u;
         ut=ul*tan_phi/((double)l_cam*tan_phi+uwyp);
         wuxc=(double)l_cam*(ut+1.0)*0.001;
         wuyc=-uwyp*t*0.001;
         wuzc=-uwzp*t*0.001;
         wu_xpos=(double)wuxc+deltah+(syserror*(double)uwyp);
         wu_ypos=(double)wuyc-(double)fcx;
         wu_zpos=(double)wuzc+(double)fcy;
         printf("\n%csave point on file ? y/n",7);
         pause=getch();
         if(pause=='y'){
             printf("\n file name : ");
             gets(fname);
             strcat(fname,".dwu");
             fp=fopen(fname,"w");
         fprintf(fp,"%lf,%lf,%lf ",wu_xpos,wu_ypos,wu_zpos);
         close(fp);}
         }/*end if*/
     ofg_clf(gaoi,0);
     cls();
     printf("\n get lower wing point ? y/n");
     pause=getch();
     if (pause=='y'){

       l_wing_zp=506;
       l_wing_yp=300;
          cls();
          printf("\n Press 'i' to move in, 'o' to move out, any key to
end");
          ofg_cline(0,l_wing_yp,0,l_wing_yp,512,15);
          do{
             pause=getch();
             ofg_cvline(gaoi,l_wing_yp,0,512,0);
             if(pause=='i')l_wing_yp++;
             if(pause=='o')l_wing_yp--;
             ofg_cline(0,l_wing_yp,0,l_wing_yp,512,15);
             }while(pause=='i' | pause=='o');
        do{
```

```
            cls();
            printf("\npress 'd' to move down, 'u' to move up, any key to
 end");
            printf("\n\nmark = %d",l_wing_zp);
            ofg_cline(0,0,l_wing_zp,768,l_wing_zp,15);
            pause=getch();
            ofg_chline(gaoi,0,l_wing_zp,768,0);
            if(pause=='d' && l_wing_zp<500)l_wing_zp++;
            if(pause=='u'&& l_wing_zp>2)l_wing_zp--;
            ofg_cline(0,0,l_wing_zp,768,l_wing_zp,15);
          }while(pause=='d' | pause=='u');
          printf("\ninput wing number , counting right to left ");
          gets(line);
          wnum_l=atof(line);
         /*calculate x,y,z*/
         lwyp=(double)(l_wing_yp-384);
         lwzp=(double)(256-l_wing_zp);
         ll=Lminusl[0]+(double)sep*wnum_l;
         lt=ll*tan_phi/((double)l_cam*tan_phi+lwyp);
         wlxc=(double)l_cam*(lt+1.0)*0.001;
         wlyc=-lwyp*t*0.001;
         wlzc=-lwzp*t*0.001;
         wl_xpos=(double)wlxc+deltah+(syserror*(double)lwyp);
         wl_ypos=(double)wlyc-(double)fcx;
         wl_zpos=(double)wlzc+(double)fcy;
         printf("\n%csave point on file ? y/n",7);
         pause=getch();
         if(pause=='y'){
             printf("\n file name : ");
             gets(fname);
             strcat(fname,".dwl");
             fp=fopen(fname,"w");
         fprintf(fp,"%lf,%lf,%lf ",wl_xpos,wl_ypos,wl_zpos);
         close(fp);}
         }/*end if*/
         clear_overlay();
    }/*end proc*/

m_calibration(){
      char pause;
      int l=100,r=668;
      /*set lut 15 to 255*/
      ofg_clearlut(OUTPUT,15,255);
      /*set dynamic output lut mode - lut depends on pixel values in
      overlay*/
      ofg_dlutmode(DYNAMIC);
      ofg_setframe(0);
      /*in overlay frame, draw line down centre of screen*/
      ofg_cline(0,384,0,384,512,15);
      /*draw lines at equal distances each side of middle line*/
      do{ ofg_cline(0,l,0,l,512,15);
          ofg_cline(0,r,0,r,512,15);
          ofg_grab(CAMERA,gaoi);
          cls();
          printf("\n1/ direction of translation along camera axis");
          printf("\n press 'i' to move in, 'o' to move out");
          pause=getch();
```

```
            ofg_cvline(gaoi,l,0,512,0);
            ofg_cvline(gaoi,r,0,512,0);
            if(pause=='i'){l++;r--;}
            if(pause=='o'){l--;r++;}
         }while(pause=='i' | pause=='o');
         clear_overlay();      }

systemerr(){/*adjust for systematic error*/
int lx[13],rx[13],i,ldist,rdist;
double ltan_phi,rtan_phi,lsep,rsep,ll,rl,deltah=0.0,lsyserror,rsyserror;
double lx_pos[13],rx_pos[13],yp,t,Lminusl[13];
char pause;

/*open right calibration file*/
fp=fopen("right.cal","r");
fscanf(fp,"%lf,%lf,%lf",&rtan_phi,&rsep,&rl);
for(i=0;i<13;i++)fscanf(fp,",%d",&rx[i]);
/*open left calibration file*/
fp=fopen("left.cal","r");
fscanf(fp,"%lf,%lf,%lf",&ltan_phi,&lsep,&ll);
for(i=12;i>=0;i--)fscanf(fp,",%d",&lx[i]);
ldist=lx[1]-lx[11];
rdist=rx[11]-rx[1];
/*calculate depths for right*/
Lminusl[0]=150000.0;
for(i=1;i<13;i++)Lminusl[i]=Lminusl[0]-rsep*(double)i;
for(i=0;i<13;i++){
          yp=(double)(rx[i]-384);
          t=Lminusl[i]*rtan_phi/(rl*rtan_phi+yp);
          rx_pos[i]=rl*(t+1.0)*0.001;
          }
/*calculate depths for left*/
Lminusl[0]=150000.0;
for(i=1;i<13;i++)Lminusl[i]=Lminusl[0]-lsep*(double)i;
for(i=0;i<13;i++){
          yp=(double)(384-lx[i]);
          t=Lminusl[i]*ltan_phi/(ll*ltan_phi+yp);
          lx_pos[i]=ll*(t+1.0)*0.001;
          }

deltah=(lx_pos[2]-rx_pos[2])/2.0;
lsyserror=(lx_pos[1]-lx_pos[11])/(double)ldist;
rsyserror=(rx_pos[1]-rx_pos[11])/(double)rdist;
printf("\n deltah = %lf microns",deltah);
printf("\n left system error = %lf microns",lsyserror);
printf("\n right system error = %lf microns",rsyserror);
pause=getch();
fp=fopen("right.cal","w");
fprintf(fp,"%lf,%lf,%lf,%lf,%lf",rtan_phi,rsep,rl,-1.0*deltah,rsyserror);
                    fclose(fp);
fp=fopen("left.cal","w");
fprintf(fp,"%lf,%lf,%lf,%lf,%lf",ltan_phi,lsep,ll,deltah,lsyserror);
                    fclose(fp);
}

win_calib(side)char side;
     {
```

```
        int ascend1[13],ascend2[13],y=256,x,i,z=0;
        WORD threshold=50;
        double l_line,l=0.0,h=0.0,angle=0.0,phi=0.0,mean_tan=0.0;
        double mean_phi=0.0,sep=0.0,base,Lminusl[13],p=0.0;
        double cal_dist,meas_dist,lcal,left,right,plane0;
        char pause,line[256];
        ofg_freeze();
        if(side=='l')ofg_im_restore(gaoi,"l1");
           else ofg_im_restore(gaoi,"r1");
        cls();
        printf("\nenter threshold [50] : ");
        gets(line);
        threshold=(WORD)atoi(line);
        if(threshold==0)threshold=50;
        if(side=='l'){
        x=n_upper_thresh(1,y,threshold);
        for(i=0;i<13;i++){
                   ascend1[i]=l_lower_thresh(x,y,threshold);
                   if(i!=0)if(ascend1[i]==ascend1[i-1]){printf("\nerror in
 plane calibration");
                                                printf("\npress any key");
                                                pause=getch();
                                                return(-1);}

           x=n_upper_thresh(ascend1[i]+1,y,threshold);
           }

        }
        else{
          for(i=0;i<13;i++){
           if(i==0)ascend1[i]=l_upper_thresh(1,y,threshold);
             else {ascend1[i]=l_upper_thresh(x,y,threshold);
                   if(ascend1[i]==ascend1[i-1]){printf("\nerror in plane
 calibration");
                                                printf("\npress any key");
                                                pause=getch();
                                                return(-1);}
                   }
           x=n_lower_thresh(ascend1[i]+1,y,threshold);
           }
        }/*end if-else*/

        if(side=='l')ofg_im_restore(gaoi,"l2");
           else ofg_im_restore(gaoi,"r2");

        if(side=='l'){
        x=n_upper_thresh(1,y,threshold);
        for(i=0;i<13;i++){
                   ascend2[i]=l_lower_thresh(x,y,threshold);
                   if(i!=0)if(ascend2[i]==ascend2[i-1]){printf("\nerror in
 plane calibration");
                                                printf("\npress any key");
                                                pause=getch();
                                                return(-1);}

           x=n_upper_thresh(ascend2[i]+1,y,threshold);
           }
```

```
      }
      else{
      for(i=0;i<13;i++){
         if(i==0)ascend2[i]=l_upper_thresh(1,y,threshold);
           else {ascend2[i]=l_upper_thresh(x,y,threshold);
                 if(ascend2[i]==ascend2[i-1]){printf("\nerror in plane
calibration");
                                              printf("\npress any key");
                                              pause=getch();
                                              return(-1);}
                 }
         x=n_lower_thresh(ascend2[i]+1,y,threshold);
         }
      }/*end if-else*/
      printf("\nenter distance moved in microns [1500]:");
      gets(line);
      h=(double)atof(line);
      if(h==0.0)h=1500.0;
      /*do not use end two planes*/
      for(i=2;i<11;i++){
           angle=((double)ascend1[i]-(double)ascend2[i])*(double)dist_
per_pixel/h;
           phi=2.0*90.0*(double)atan(angle)/3.142;
           printf("\n1=%d,2=%d,tan=%lf,angle = %lf",ascend1[i],ascend2[i],
angle,phi);
           mean_tan+=angle;
           mean_phi+=phi;
           }

      mean_tan=mean_tan/9.0;
      phi=mean_phi/9.0;
      printf("\nmean tan %lf",mean_tan);
      printf("\nmean phi= %lf",phi);
      sep=(((double)ascend1[9]-(double)ascend1[3])/6.0)*(double)dist_per
_pixel/mean_tan;
      printf("\n seperation along axis mm = %lf",sep*0.001);

      /*seperation along axis for each plane*/
      printf("\n input distance to plane 0 in microns [150000]:");
      gets(line);
      plane0=(double)atof(line);
      if(plane0==0)plane0=150000.0;
      Lminus1[0]=plane0;
      for(i=1;i<13;i++){Lminus1[i]=plane0-sep*(double)i;
                        printf("\n dist to plane %d = %lf",i,Lminus1[i]);}
      meas_dist=dist_per_pixel*(double)(ascend1[11]-ascend1[1]);
      l=-7635.000000;
      pause=getch();
      /*************************************************************/
      /*                                                           */
      /*     camera calibration code is held in file camcode       */
      /*     deltah is the difference in height between the        */
      /*     left and the right side planes                        */
      /*     syserror is the systematic error in depth per         */
      /*     pixel                                                 */
      /*                                                           */
```

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

        ofg_dlutmode(STATIC);
        if(side=='r')
                    {
                     fp=fopen("right.cal","w");
                     fprintf(fp,"%lf,%lf,%lf",mean_tan,sep,l);
                     for(i=0;i<13;i++)fprintf(fp,",%d",ascend2[i]);
                     fclose(fp);}
        if(side=='l')
                    {
                     fp=fopen("left.cal","w");
                     fprintf(fp,"%lf,%lf,%lf",mean_tan,sep,l);
                     for(i=0;i<13;i++)fprintf(fp,",%d",ascend1[i]);
                     fclose(fp);}
        }


 upper_thresh(){
        char choose;
        do{
           cls();
           printf("\nupper threshold = %d",top);
           printf("\npress u - threshold up, d - threshold down, e - end");
           choose=getch();
           if(choose=='d'){if(top==bottom)break;
             ofg_wval(OUTPUT,0,top,0);
             top--;}
           if(choose=='u'){if(top==255)break;
             top++;
             ofg_wval(OUTPUT,0,top,top);}
           }while(choose!='e');
        }

 lower_thresh(){
        char choose;
        do{
           cls();
           printf("\nlower threshold = %d",bottom);
           printf("\npress u - threshold up, d - threshold down, e - end");
           choose=getch();
           if(choose=='d'){if(bottom==1)break;
             bottom--;
             ofg_wval(OUTPUT,0,bottom,bottom);}
           if(choose=='u'){if(bottom==top)break;
             ofg_wval(OUTPUT,0,bottom,0);
             bottom++;}
           }while(choose!='e');
        }


 real_t(){
       /*set lut 15 to 255*/
        ofg_clearlut(OUTPUT,15,255);
        ofg_dlutmode(STATIC);
        ofg_setframe(I);
```

```
      ofg_grab(CAMERA,gaoi);
}

grab(){
       char pause;
       /*draw line down centre of screen*/
       ofg_clearlut(OUTPUT,15,255);
       ofg_dlutmode(DYNAMIC);
       ofg_setframe(O);
       ofg_cline(0,384,0,384,512,15);
       ofg_setframe(I);
       cls();
       printf("\npress any key to grab image");
       pause=getch();
       clear_overlay();
       ofg_snap(CAMERA,gaoi);
       ofg_dlutmode(STATIC);
}

clear_overlay(){
      ofg_setframe(O);
      ofg_clf(gaoi,0);
      ofg_setframe(I);
}

sharpen(){
       ofg_sharpen(gaoi,gaoi,POSITIVE);
       /*sharpen edges in right half of image*/
       ofg_envert(gaoir,gaoir,POSITIVE);
       /*sharpen edges in left half of image*/
       ofg_convolve(gaoil,gaoil,3,5,v_kernel,0,0,POSITIVE);
}

sobel(){
       ofg_sobel(gaoi,gaoi,32);
}

save(){
      double tan_phi,sep,l,deltah,syserror;
      char pause,fname[10];
      cls();
      printf("%cfilename to save image:",7);
      gets(name);
      ofg_im_save(gaoi,EIGHT_BIT,name);
      strcpy(fname,name);
      /* save left calibration file*/
      if( (fp = fopen("left.cal","r"))==NULL){
           printf("\n%c cannot find left calibration file",7);
           printf("\npress any key to continue");
           pause=getch();
           return(-1);}
      fscanf(fp,"%lf,%lf,%lf,%lf,%lf",&tan_phi,&sep,&l,&deltah,&syserror);
      fclose(fp);
      strcat(fname,".lcb");
      fp=fopen(fname,"w");
      fprintf(fp,"%lf,%lf,%lf,%lf,%lf",tan_phi,sep,l,deltah,syserror);
      fclose(fp);
```

```
        /* save right calibration file*/
        if( (fp = fopen("right.cal","r"))==NULL){
             printf("\n%c cannot find right calibration file",7);
             printf("\npress any key to continue");
             pause=getch();
             return(-1);}
        fscanf(fp,"%lf,%lf,%lf,%lf,%lf",&tan_phi,&sep,&l,&deltah,&syserror);
        fclose(fp);
        strcpy(fname,name);
        strcat(fname,".rcb");
        fp=fopen(fname,"w");
        fprintf(fp,"%lf,%lf,%lf,%lf,%lf",tan_phi,sep,l,deltah,syserror);
        fclose(fp);
}

restore(){
        cls();
        printf("%cfilename to restore image:",7);
        gets(name);
        ofg_freeze();
        ofg_im_restore(gaoi,name);
}

equalise(){
DWORD histvals[256];
        ofg_histogram(gaoi,2,2,0,histvals);
        ofg_eq_lut(OUTPUT,0,histvals);
        ofg_maplut(gaoi,gaoi,OUTPUT,0);
        ofg_linlut(OUTPUT,0);
        }

auto_equalise(){
DWORD histvals[256];
        ofg_histogram(gaoi,2,2,0,histvals);
        ofg_eq_lut(ADC,0,histvals);
        }

w_auto_equalise(){
static char fname[12]="enhance.pic";
DWORD histvals[256];
        ofg_freeze();
        ofg_im_restore(gaoi,fname);
        ofg_histogram(gaoi,2,2,0,histvals);
        ofg_eq_lut(ADC,0,histvals);
        }

median(){
        static int kernel[]={
        1,1,1,
        1,1,1,
        1,1,1
        };
        ofg_median(gaoi,gaoi,3,3,kernel);
        }

mul_mat(){          /*called by contour map plot*/
       char str[12],pause,side,elem,fname[10];
```

```
      static char element[3]=" ";
      int i,j,k=0.0,uaxis,vaxis,pont,plne;
      int upos,vpos,inc,uflag=0,dflag=0;


     /*row and column vector*/
     double rv[ORDER],cv[ORDER];

     /* matrices of points saved on disk */
     static double trx_pos[POINTS][PLANES],try_pos[POINTS][PLANES],
trz_pos[POINTS][PLANES];
     static double tlx_pos[POINTS][PLANES],tly_pos[POINTS][PLANES],
tlz_pos[POINTS][PLANES];
     /*wing points*/
     double wu_xpos=0.0,wu_ypos,wu_zpos,wl_xpos=0.0,wl_ypos,wl_zpos;

     /*co-ordinate matrices*/
     /*used as 5x5 matrix which moves over the surface to average*/
     /*curvature for a small area*/
     static double x_patch[5][5],y_patch[5][5],z_patch[5][5],
r1_patch[5][5],r2_patch[5][5];
     double x,y,z;
     double u,v;

     double r,theta,phi,step,xmin=1000.0;
     static double contlx[9],contly[9],contlz[9];

     static double deltax[9],deltay[9],deltaz[9];
     double lefty,topz,righty,bottomz,catch=0.025;
     double pxcentre=0.0,pycentre=0.0,pzcentre=0.0;

    /*initilise matrices*/
    for(i=0;i<ORDER;i++) rv[i]=cv[i]=0.0;
    for(i=0;i<3;i++)
        for(j=0;j<3;j++)pxrect[i][j]=pyrect[i][j]=pzrect[i][j]=0.0;
    for(i=0;i<9;i++)deltax[i]=deltay[i]=deltaz[i]=0.0;
    cls();
    printf("\nis the patch theoretical ? y/n ");
    pause=getch();
    if(pause=='y'){
       printf("\n  r = ");
       gets(str);
       r=atof(str);
       printf("\n  step = ");
       gets(str);
       step=atof(str);


       for(i=0;i<ORDER;i++){  /*phi varies*/
         phi=3.142*((step*i)-12.5)/180.0;
         for(j=0;j<ORDER;j++){  /*theta varies*/
             theta=3.142*5.0*((step*j)-12.5)/180.0;
             px[i][j]=r*sin(phi)*cos(theta);
             py[i][j]=r*sin(phi)*sin(theta);
             pz[i][j]=r*cos(phi);
             }/*next j*/
```

```
        } /*next i*/
    }
    else{
        /*data points read from disk and placed in            */
        /*px[ORDER][ORDER],py[ORDER][ORDER], pz[ORDER][ORDER] */

        cls();
        printf("\n file name : ");
        gets(name);
        strcpy(fname,name);
        /*input left side points*/
        strcat(fname,".dal");
        fp=fopen(fname,"rb");
        fread(tlx_pos,sizeof(tlx_pos),1,fp);
        fread(tly_pos,sizeof(tly_pos),1,fp);
        fread(tlz_pos,sizeof(tlz_pos),1,fp);
        fclose(fp);

         /*input right point file */
      strcpy(fname,name);
      strcat(fname,".dar");
      fp=fopen(fname,"rb");
      fread(trx_pos,sizeof(trx_pos),1,fp);
      fread(try_pos,sizeof(try_pos),1,fp);
      fread(trz_pos,sizeof(trz_pos),1,fp);
      fclose(fp);
             cls();
             printf("\noutput points to printer? y/n");
             pause=getch();
             if(pause=='y'){
                          printf("\n%ccheck left file matrix ",7);
                          for(i=0;i<POINTS;i++){fprintf(stdprn,"\n");
                            for(j=PLANES-1;j>=0;j--){
                               fprintf(stdprn,"x");
                               fprintf(stdprn,"[%7.3lf] ",tlx_pos[i]
[j]);}
                               fprintf(stdprn,"\n\r");

                           for(j=PLANES-1;j>=0;j--){
                               fprintf(stdprn,"y");
                               fprintf(stdprn,"[%7.3lf] ",tly_pos[i]
[j]);}
                               fprintf(stdprn,"\n\r");

                           for(j=PLANES-1;j>=0;j--){
                               fprintf(stdprn,"z");
                               fprintf(stdprn,"[%7.3lf] ",tlz_pos[i]
[j]);}
                               fprintf(stdprn,"\n\r");
                    }
                    fprintf(stdprn,"\f");
                    fflush(stdprn);
      pause=getch();

             cls();
                        printf("\n%ccheck right file matrix ",7);
                        for(i=0;i<POINTS;i++){fprintf(stdprn,"\n");
```

```
                                for(j=0;j<PLANES;j++){
                                    fprintf(stdprn,"x");
                                    fprintf(stdprn,"[%7.3lf] ",trx_pos[i]
[j]);}
                                    fprintf(stdprn,"\n\r");

                                for(j=0;j<PLANES;j++){
                                    fprintf(stdprn,"y");
                                    fprintf(stdprn,"[%7.3lf] ",try_pos[i]
[j]);}
                                    fprintf(stdprn,"\n\r");

                                for(j=0;j<PLANES;j++){
                                    fprintf(stdprn,"z");
                                    fprintf(stdprn,"[%7.3lf] ",trz_pos[i]
[j]);}
                                    fprintf(stdprn,"\n\r");
                            }
                        fprintf(stdprn,"\f");
                        fflush(stdprn);
                        }/*end if 'print points' */
        /*input upper wing point*/
        strcpy(fname,name);
        strcat(fname,".dwu");
        if((fp=fopen(fname,"r"))!=NULL){
           fscanf(fp,"%lf,%lf,%lf",&wu_xpos,&wu_ypos,&wu_zpos);
           fclose(fp);}
        /*input lower wing point*/
        strcpy(fname,name);
        strcat(fname,".dwl");
        if((fp=fopen(fname,"r"))!=NULL){
           fscanf(fp,"%lf,%lf,%lf",&wl_xpos,&wl_ypos,&wl_zpos);
           fclose(fp);}
         pause=getch();
         /*place points in matrix patch file*/
         for(i=0;i<3;i++){
             for(j=0;j<3;j++){
                cls();
                printf("enter for point (%d,%d) ",i,j);
                printf("\nleft/right/up/down (l/r/u/d) : ");
                side=getch();
                if(side=='l' || side=='r'){
                      printf("\n point : ");
                      elem=getch();
                      element[0]=elem;
                      pont=atoi(element);
                      printf("\n plane : ");
                      elem=getch();
                      element[0]=elem;
                      plne=atoi(element);
                      }/*end if*/
             switch(side){
                    case 'l':px[i][j]=tlx_pos[pont][plne];
                             py[i][j]=tly_pos[pont][plne];
                             pz[i][j]=tlz_pos[pont][plne];
                             break;
                    case 'r':px[i][j]=trx_pos[pont][plne];
```

```
                    py[i][j]=try_pos[pont][plne];
                    pz[i][j]=trz_pos[pont][plne];
                    break;
            case 'u':px[i][j]=wu_xpos;
                    py[i][j]=wu_ypos;
                    pz[i][j]=wu_zpos;
                    uflag=1;
                    break;
            case 'd':px[i][j]=wl_xpos;
                    py[i][j]=wl_ypos;
                    pz[i][j]=wl_zpos;
                    dflag=1;
                    break;
            }/*end case*/
        }
    }

     cls();
                printf("\n%ccheck patch matrix ",7);
                for(i=0;i<ORDER;i++){printf("\n");
                    for(j=0;j<ORDER;j++){
                        printf("x");
                        printf("[%7.3lf] ",px[i][j]);}
                        printf("\n");
                    for(j=0;j<ORDER;j++){
                        printf("y");
                        printf("[%7.3lf] ",py[i][j]);}
                        printf("\n");
                    for(j=0;j<ORDER;j++){
                        printf("z");
                        printf("[%7.3lf] ",pz[i][j]);}
            }
lefty=py[0][0];
righty=py[0][2];
for(i=1;i<3;i++)
       {
        if(py[i][0]>lefty)lefty=py[i][0];
        if(py[i][2]<righty)righty=py[i][2];
        }
topz=pz[0][0];
bottomz=pz[2][0];
for(j=1;j<3;j++)
       {
        if(pz[0][j]<topz)topz=pz[0][j];
        if(pz[2][j]>bottomz)bottomz=pz[2][j];
       }
printf("\nleft=%lf",lefty);
printf("\nright=%lf",righty);
printf("\ntop=%lf",topz);
printf("\nbottom=%lf",bottomz);

/*  adjust matrix */
if((uflag==1) && (dflag==1)){/*only move centre point*/
printf("\ncalculating matrix ");
  u=v=0.0;
  for(uaxis=50;uaxis<200;uaxis++){
     u=0.0+(double)uaxis*0.004;
```

```
            for(vaxis=50;vaxis<200;vaxis++){
               v=0.0+(double)vaxis*0.004;
                /*construct surface patch*/
                /*column and row vectors*/
                rv[0]=u*u;
                rv[1]=u;
                rv[2]=1.0;
                cv[0]=v*v;
                cv[1]=v;
                cv[2]=1.0;

                /*calculate x,y,z*/
                x=matrix(rv,px,cv);
                y=matrix(rv,py,cv);
                z=matrix(rv,pz,cv);

                if(pxcentre==0.0){
                  if((y>(lefty+righty)/2.0-catch)&&(y<(lefty+righty)/
2.0+catch)&&(z>(topz+bottomz)/2.0-catch)&&(z<(topz+bottomz)/2.0+catch))
                     {pxcentre=x;pycentre=y;pzcentre=z;
                      printf("\ncorrection made");}
                  }

                }
             }
          if(pxcentre>0.0){px[1][1]=pxcentre;py[1][1]=pycentre;pz[1][1]=
pzcentre;
             printf("\ncorrection implemented");
             printf("\npress any key");}
          pause=getch();
          }
          else
          {/*rebuild rectangular matrix*/
          printf("\ncalculating matrix ");
          do{
            /*  build rectangular matrix */
            u=v=0.0;
            for(uaxis=0;uaxis<250;uaxis++){
               u=0.0+(double)uaxis*0.004;
               for(vaxis=0;vaxis<250;vaxis++){
                  v=0.0+(double)vaxis*0.004;

                  /*construct surface patch*/
                  /*column and row vectors*/
                  rv[0]=u*u;
                  rv[1]=u;
                  rv[2]=1.0;
                  cv[0]=v*v;
                  cv[1]=v;
                  cv[2]=1.0;

                  /*calculate x,y,z*/
                  x=matrix(rv,px,cv);
                  y=matrix(rv,py,cv);
                  z=matrix(rv,pz,cv);

                  if((y>lefty-catch)&&(y<lefty+catch)&&(z>topz-catch)&&
```

```
(z<topz+catch))
                        {pxrect[0][0]=x;pyrect[0][0]=y;pzrect[0][0]=z;}
                    if((y>(lefty+righty)/2.0-catch)&&(y<(lefty+righty)/
2.0+catch)&&(z>topz-catch)&&(z<topz+catch))
                        {pxrect[0][1]=x;pyrect[0][1]=y;pzrect[0][1]=z;}
                    if((y>righty-catch)&&(y<righty+catch)&&(z>topz-catch)&&
(z<topz+catch))
                        {pxrect[0][2]=x;pyrect[0][2]=y;pzrect[0][2]=z;}
                    if((y>lefty-catch)&&(y<lefty+catch)&&(z>(topz+bottomz)/
2.0-catch)&&(z<(topz+bottomz)/2.0+catch))
                        {pxrect[1][0]=x;pyrect[1][0]=y;pzrect[1][0]=z;}
                    if((y>(lefty+righty)/2.0-catch)&&(y<(lefty+righty)/
2.0+catch)&&(z>(topz+bottomz)/2.0-catch)&&(z<(topz+bottomz)/2.0+catch))
                        {pxrect[1][1]=x;pyrect[1][1]=y;pzrect[1][1]=z;}
                    if((y>righty-catch)&&(y<righty+catch)&&(z>(topz+bottomz)
/2.0-catch)&&(z<(topz+bottomz)/2.0+catch))
                        {pxrect[1][2]=x;pyrect[1][2]=y;pzrect[1][2]=z;}
                    if((y>lefty-catch)&&(y<lefty+catch)&&(z>bottomz-catch)&&
(z<bottomz+catch))
                        {pxrect[2][0]=x;pyrect[2][0]=y;pzrect[2][0]=z;}
                    if((y>(lefty+righty)/2.0-catch)&&(y<(lefty+righty)/
2.0+catch)&&(z>bottomz-catch)&&(z<topz+bottomz+catch))
                        {pxrect[2][1]=x;pyrect[2][1]=y;pzrect[2][1]=z;}
                    if((y>righty-catch)&&(y<righty+catch)&&(z>bottomz-catch)
&&(z<bottomz+catch))
                        {pxrect[2][2]=x;pyrect[2][2]=y;pzrect[2][2]=z;}
                    }
                }
                                printf("\n%ccheck rectangular matrix ",7);
                                for(i=0;i<ORDER;i++){printf("\n");
                                    for(j=0;j<ORDER;j++){
                                        printf("x");
                                        printf("[%7.3lf] ",pxrect[i][j]);}
                                        printf("\n");
                                    for(j=0;j<ORDER;j++){
                                        printf("y");
                                        printf("[%7.3lf] ",pyrect[i][j]);}
                                        printf("\n");
                                    for(j=0;j<ORDER;j++){
                                        printf("z");
                                        printf("[%7.3lf] ",pzrect[i][j]);}
                            }
            printf("\nif elements = [0], rebuild surface matrix y/n ? ");
            pause=getch();
            printf("%c",pause);
            if(pause=='y'){
                            printf("\n input new error for matrix position
[%5.3lf]:",catch);
                            gets(str);
                            if(str=="")catch==catch;
                               else catch=(double)atof(str);}
            }while(pause=='y');
            for(i=0;i<3;i++)
               for(j=0;j<3;j++) {px[i][j]=pxrect[i][j];
                                 py[i][j]=pyrect[i][j];
                                 pz[i][j]=pzrect[i][j];}
          }/*end if (uflag && dflag) */
```

```
        }/*end if-else*/

      cls();
      printf("\nplot contours y/n ? ");
      pause=getch();
      printf("%c\n",pause);
      if(pause=='y'){
      /*calculate highest (closest) point of surface*/
            u=v=0.0;
            for(uaxis=0;uaxis<100;uaxis++){
               u=0.0+(double)uaxis*0.01;
               for(vaxis=0;vaxis<100;vaxis++){
                  v=0.0+(double)vaxis*0.01;

                  /*construct surface patch*/
                  /*column and row vectors*/
                  rv[0]=u*u;
                  rv[1]=u;
                  rv[2]=1.0;
                  cv[0]=v*v;
                  cv[1]=v;
                  cv[2]=1.0;

                  /*calculate x,y,z*/
                  x=matrix(rv,px,cv);
                  y=matrix(rv,py,cv);
                  z=matrix(rv,pz,cv);
                  if(x<xmin)xmin=x;
                  }
            }
      contours(xmin);}


}

double matrix(rv,points,cv)double rv[ORDER],points[ORDER][ORDER],cv[ORDER];{
       int i,j,k;
       char pause;

       static double new_m[ORDER][ORDER],new_mT[ORDER][ORDER],result_
a[ORDER][ORDER],result[ORDER][ORDER];
       double sum=0.0;
       static double m[3][3]={
                              {2.0,-4.0,2.0},
                              {-3.0,4.0,-1.0},
                              {1.0,0.0,0.0}};
       static double mT[3][3]={
                               {2.0,-3.0,1.0},
                               {-4.0,4.0,0.0},
                               {2.0,-1.0,0.0}};

      /*initialise matrices*/
       for(i=0;i<ORDER;i++){
          for(j=0;j<ORDER;j++){
              new_m[i][j]=new_mT[i][j]=result_a[i][j]=result[i][j]=0.0;
          }/*next j*/
```

```
      }/*next i*/


      /* multiply row and column vectors */
      /* second suffix = row */
      /*  ___________________  */
      /* |         ->j       | */
      /* |                   | */
      /* |  ^i               | */
      /* |                   | */
      /*  -------------------  */


      for(i=0;i<ORDER;i++){
         for(j=0;j<ORDER;j++){
             new_m[i][j]=m[i][j]*rv[i];
         }/*next j*/
      }/*next i*/

   for(j=0;j<ORDER;j++){
         for(i=0;i<ORDER;i++){
             /*indicies reversed for column multiplication*/
             new_mT[i][j]=mT[i][j]*cv[j];
         }/*next i*/
     }/*next j*/


/* multiply new_m by point matrix */
/* sum row of first matrix by column of second*/
      for(i=0;i<ORDER;i++){
         for(j=0;j<ORDER;j++){
           for(k=0;k<ORDER;k++)result_a[i][j]+=new_m[i][k]*points[k][j];
         }
      }

/* multiply result by new_mT matrix */
      for(i=0;i<ORDER;i++){
         for(j=0;j<ORDER;j++){
            for(k=0;k<ORDER;k++)result[i][j]+=result_a[i][k]*new_mT[k][j];

         }
      }
  /*sum matrix*/
  for(i=0;i<ORDER;i++){
     for(j=0;j<ORDER;j++){
       sum+=result[i][j];
     }
  }

   return(sum);
}

double matsp(rv,points,cv)double rv[ORDER],points[ORDER][ORDER],cv[ORDER];{
      int i,j,k;
      char pause;
```

```
      static double new_m[ORDER][ORDER],new_mT[ORDER][ORDER],result_a
 [ORDER][ORDER],result[ORDER][ORDER];
      double sum=0.0;
      static double m[3][3]={
                             {1.0,-2.0,1.0},
                             {-2.0,2.0,0.0},
                             {1.0,1.0,0.0}};
      static double mT[3][3]={
                              {1.0,-2.0,1.0},
                              {-2.0,2.0,1.0},
                              {1.0,0.0,0.0}};

     /*initialise matrices*/
      for(i=0;i<ORDER;i++){
         for(j=0;j<ORDER;j++){
             new_m[i][j]=new_mT[i][j]=result_a[i][j]=result[i][j]=0.0;
         }/*next j*/
      }/*next i*/


      /* multiply row and column vectors */
      /* second suffix = row */
      /*  _________________  */
      /* |        ->j      | */
      /* |                 | */
      /* |  ^i             | */
      /* |                 | */
      /*  -----------------  */


      for(i=0;i<ORDER;i++){
         for(j=0;j<ORDER;j++){
             new_m[i][j]=m[i][j]*rv[i];
         }/*next j*/
      }/*next i*/

   for(j=0;j<ORDER;j++){
         for(i=0;i<ORDER;i++){
             /*indicies reversed for column multiplication*/
             new_mT[i][j]=mT[i][j]*cv[j];
         }/*next i*/
     }/*next j*/



/* multiply new_m by point matrix */
/* sum row of first matrix by column of second*/
      for(i=0;i<ORDER;i++){
         for(j=0;j<ORDER;j++){
           for(k=0;k<ORDER;k++)result_a[i][j]+=new_m[i][k]*points[k][j];
         }
      }

/* multiply result by new_mT matrix */
      for(i=0;i<ORDER;i++){
         for(j=0;j<ORDER;j++){
            for(k=0;k<ORDER;k++)result[i][j]+=result_a[i][k]*new_mT[k][j];
```

```
        }
      }
  /*sum matrix*/
  for(i=0;i<ORDER;i++){
     for(j=0;j<ORDER;j++){
       sum+=result[i][j];
     }
  }

 sum=sum/(double)4.0;
   return(sum);
}

mul4_mat(){
      char str[12],pause,side,elem,fname[10];
      static char element[3]=" ";
      int i,j,k=0.0,uaxis,vaxis,pont,plne;
      int upos,vpos,inc=1;
      int apex_u,apex_v,uelement,velement;
     /*row and column vector*/
     double rv[ORDER],cv[ORDER];

     /* matrices of points saved on disk */
     static double trx_pos[POINTS][PLANES],try_pos[POINTS][PLANES],
trz_pos[POINTS][PLANES];
     static double tlx_pos[POINTS][PLANES],tly_pos[POINTS][PLANES],
tlz_pos[POINTS][PLANES];
     /*wing points*/
     double wu_xpos=0.0,wu_ypos,wu_zpos,wl_xpos=0.0,wl_ypos,wl_zpos;

     /*co-ordinate matrices*/
     static double pxcurve[ORDER][ORDER],pycurve[ORDER][ORDER],pzcurve
[ORDER][ORDER];
     static double x_patch[5][5],y_patch[5][5],z_patch[5][5],r1_patch
[5][5],r2_patch[5][5];
     /*first and second partial derivative variables*/
     double x,y,z,xu,yu,zu,xv,yv,zv,xuu,yuu,zuu,xvv,yvv,zvv,xuv,yuv,zuv;
     double u,v;
     /*fundamental form variables*/
     double E_,F_,G_,e,f,g;

     double denom=0.0,lmax=0.0,lmin=0.0,kmax=0.0,kmin=0.0;
     double complex=0.0,a=0.0,b=0.0,r1=0.0,r2=0.0,eq1=0.0,eq2=0.0;
     double r,theta,phi,step,mean_r1=0.0,mean_r2=0.0;
     double av_mean=0.0,gauss_mean=0.0,length,xmin=1000.0;
     static double contlx[9],contly[9],contlz[9];

     static double deltax[9],deltay[9],deltaz[9];
     double lefty,topz,righty,bottomz,catch=0.012;
     double apex_r=20.0,apex_y,apex_z;
    /*initilise matrices*/
    for(i=0;i<ORDER;i++) rv[i]=cv[i]=0.0;
    for(i=0;i<9;i++)deltax[i]=deltay[i]=deltaz[i]=0.0;
    for(i=0;i<3;i++)
        for(j=0;j<3;j++)pxrect[i][j]=pyrect[i][j]=pzrect[i][j]=0.0;
    cls();
```

```
printf("\nis the patch theoretical ? y/n ");
pause=getch();
if(pause=='y'){
   printf("\n  r = ");
   gets(str);
   r=atof(str);
   printf("\n  step = ");
   gets(str);
   step=atof(str);


   for(i=0;i<ORDER;i++){  /*phi varies*/
     phi=3.142*((step*i)-12.5)/180.0;
     for(j=0;j<ORDER;j++){  /*theta varies*/
         theta=3.142*5.0*((step*j)-12.5)/180.0;
         px[i][j]=r*sin(phi)*cos(theta);
         py[i][j]=r*sin(phi)*sin(theta);
         pz[i][j]=r*cos(phi);
         }/*next j*/
      } /*next i*/
}
else{
     /*data points read from disk and placed in            */
     /*px[ORDER][ORDER],py[ORDER][ORDER], pz[ORDER][ORDER] */

     cls();
     printf("\n file name : ");
     gets(name);
     strcpy(fname,name);
     /*input left side points*/
     strcat(fname,".dal");
     fp=fopen(fname,"rb");
     fread(tlx_pos,sizeof(tlx_pos),1,fp);
     fread(tly_pos,sizeof(tly_pos),1,fp);
     fread(tlz_pos,sizeof(tlz_pos),1,fp);
     fclose(fp);
          cls();
                    printf("\n%ccheck left file matrix ",7);
                    for(i=0;i<POINTS;i++){printf("\n");
                        for(j=PLANES-1;j>=0;j--){
                            printf("x");
                            printf("[%7.3lf] ",tlx_pos[i][j]);}

                }
   pause=getch();

      /*input right point file */
   strcpy(fname,name);
   strcat(fname,".dar");
   fp=fopen(fname,"rb");
   fread(trx_pos,sizeof(trx_pos),1,fp);
   fread(try_pos,sizeof(try_pos),1,fp);
   fread(trz_pos,sizeof(trz_pos),1,fp);
   fclose(fp);
          cls();
                    printf("\n%ccheck right file matrix ",7);
```

```
                for(i=0;i<POINTS;i++){printf("\n");
                    for(j=0;j<PLANES;j++){
                        printf("x");
                        printf("[%7.3lf] ",trx_pos[i][j]);}
                }
/*input upper wing point*/
strcpy(fname,name);
strcat(fname,".dwu");
if((fp=fopen(fname,"r"))!=NULL){
   fscanf(fp,"%lf,%lf,%lf",&wu_xpos,&wu_ypos,&wu_zpos);
   fclose(fp);}
/*input lower wing point*/
strcpy(fname,name);
strcat(fname,".dwl");
if((fp=fopen(fname,"r"))!=NULL){
   fscanf(fp,"%lf,%lf,%lf",&wl_xpos,&wl_ypos,&wl_zpos);
   fclose(fp);}
 pause=getch();
 /*place points in matrix patch file*/
 for(i=0;i<3;i++){
     for(j=0;j<3;j++){
        cls();
        printf("enter for point (%d,%d) ",i,j);
        printf("\nleft/right/up/down (l/r/u/d) : ");
        side=getch();
        if(side=='l' || side=='r'){
              printf("\n point : ");
              elem=getch();
              element[0]=elem;
              pont=atoi(element);
              printf("\n plane : ");
              elem=getch();
              element[0]=elem;
              plne=atoi(element);
              }/*end if*/
      switch(side){
             case 'l':px[i][j]=tlx_pos[pont][plne];
                      py[i][j]=tly_pos[pont][plne];
                      pz[i][j]=tlz_pos[pont][plne];
                      break;
             case 'r':px[i][j]=trx_pos[pont][plne];
                      py[i][j]=try_pos[pont][plne];
                      pz[i][j]=trz_pos[pont][plne];
                      break;
             case 'u':px[i][j]=wu_xpos;
                      py[i][j]=wu_ypos;
                      pz[i][j]=wu_zpos;
                      break;
             case 'd':px[i][j]=wl_xpos;
                      py[i][j]=wl_ypos;
                      pz[i][j]=wl_zpos;
                      break;
             }/*end case*/
        }
     }

      cls();
```

```
                        printf("\n%ccheck patch matrix ",7);
                        for(i=0;i<ORDER;i++){printf("\n");
                            for(j=0;j<ORDER;j++){
                                printf("x");
                                printf("[%7.3lf] ",px[i][j]);}
                                printf("\n");
                            for(j=0;j<ORDER;j++){
                                printf("y");
                                printf("[%7.3lf] ",py[i][j]);}
                                printf("\n");
                            for(j=0;j<ORDER;j++){
                                printf("z");
                                printf("[%7.3lf] ",pz[i][j]);}
                    }
        lefty=py[0][0];
        righty=py[0][2];
        for(i=1;i<3;i++)
               {
                if(py[i][0]>lefty)lefty=py[i][0];
                if(py[i][2]<righty)righty=py[i][2];
                }
        topz=pz[0][0];
        bottomz=pz[2][0];
        for(j=1;j<3;j++)
               {
                if(pz[0][j]<topz)topz=pz[0][j];
                if(pz[2][j]>bottomz)bottomz=pz[2][j];
               }
        printf("\nleft=%lf",lefty);
        printf("\nright=%lf",righty);
        printf("\ntop=%lf",topz);
        printf("\nbottom=%lf",bottomz);
        printf("\ncalculating matrix ");
        do{
          /*  build rectangular matrix */
          u=v=0.0;
          for(uaxis=0;uaxis<250;uaxis++){
             u=0.0+(double)uaxis*0.004;
             for(vaxis=0;vaxis<250;vaxis++){
                v=0.0+(double)vaxis*0.004;

                /*construct surface patch*/
                /*column and row vectors*/
                rv[0]=u*u;
                rv[1]=u;
                rv[2]=1.0;
                cv[0]=v*v;
                cv[1]=v;
                cv[2]=1.0;

                /*calculate x,y,z*/
                x=matrix(rv,px,cv);
                y=matrix(rv,py,cv);
                z=matrix(rv,pz,cv);

                if((y>lefty-catch)&&(y<lefty+catch)&&(z>topz-catch)&&
(z<topz+catch))
```

```
                    {pxrect[0][0]=x;pyrect[0][0]=y;pzrect[0][0]=z;}
                if((y>(lefty+righty)/2.0-catch)&&(y<(lefty+righty)/2.0+
catch)&&(z>topz-catch)&&(z<topz+catch))
                    {pxrect[0][1]=x;pyrect[0][1]=y;pzrect[0][1]=z;}
                if((y>righty-catch)&&(y<righty+catch)&&(z>topz-catch)&&
(z<topz+catch))
                    {pxrect[0][2]=x;pyrect[0][2]=y;pzrect[0][2]=z;}
                if((y>lefty-catch)&&(y<lefty+catch)&&(z>(topz+bottomz)/
2.0-catch)&&(z<(topz+bottomz)/2.0+catch))
                    {pxrect[1][0]=x;pyrect[1][0]=y;pzrect[1][0]=z;}
                if((y>(lefty+righty)/2.0-catch)&&(y<(lefty+righty)/2.0+
catch)&&(z>(topz+bottomz)/2.0-catch)&&(z<(topz+bottomz)/2.0+catch))
                    {pxrect[1][1]=x;pyrect[1][1]=y;pzrect[1][1]=z;}
                if((y>righty-catch)&&(y<righty+catch)&&(z>(topz+bottomz)
/2.0-catch)&&(z<(topz+bottomz)/2.0+catch))
                    {pxrect[1][2]=x;pyrect[1][2]=y;pzrect[1][2]=z;}
                if((y>lefty-catch)&&(y<lefty+catch)&&(z>bottomz-catch)&&
(z<bottomz+catch))
                    {pxrect[2][0]=x;pyrect[2][0]=y;pzrect[2][0]=z;}
                if((y>(lefty+righty)/2.0-catch)&&(y<(lefty+righty)/2.0+
catch)&&(z>bottomz-catch)&&(z<topz+bottomz+catch))
                    {pxrect[2][1]=x;pyrect[2][1]=y;pzrect[2][1]=z;}
                if((y>righty-catch)&&(y<righty+catch)&&(z>bottomz-catch)&&
(z<bottomz+catch))
                    {pxrect[2][2]=x;pyrect[2][2]=y;pzrect[2][2]=z;}
                }
            }
                        printf("\n%ccheck rectangular matrix ",7);
                        for(i=0;i<ORDER;i++){printf("\n");
                            for(j=0;j<ORDER;j++){
                                printf("x");
                                printf("[%7.3lf] ",pxrect[i][j]);}
                                printf("\n");
                            for(j=0;j<ORDER;j++){
                                printf("y");
                                printf("[%7.3lf] ",pyrect[i][j]);}
                                printf("\n");
                            for(j=0;j<ORDER;j++){
                                printf("z");
                                printf("[%7.3lf] ",pzrect[i][j]);}
                    }
        printf("\nif elements = [0.0] rebuild surface matrix y/n ? ");
        pause=getch();
        printf("%c",pause);
        if(pause=='y'){
                    printf("\n input new error for matrix position
[%5.3lf]:",catch);
                    gets(str);
                    if(str=="")catch==catch;
                       else catch=(double)atof(str);
                    printf("\ncalculating matrix");}
        }while(pause=='y');
      }/*end if-else*/
/*iteration for centre spline point to pass through centre point*/
/*     contlx[0]=px[0][0];*/
/*     contlx[1]=px[0][1];*/
/*     contlx[2]=px[0][2];*/
```

```
/*      contlx[3]=px[1][0];*/
/*      contlx[4]=pxrect[1][1];*/
/*      contlx[5]=px[1][2];*/
/*      contlx[6]=px[2][0];*/
/*      contlx[7]=px[2][1];*/
/*      contlx[8]=px[2][2];*/

/*      do{*/
/*         px[0][0]=px[0][0]-deltax[0];*/
/*         px[0][1]=px[0][1]-deltax[1];*/
/*         px[0][2]=px[0][2]-deltax[2];*/
/*         px[1][0]=px[1][0]-deltax[3];*/
/*         pxrect[1][1]=pxrect[1][1]-deltax[4];*/
/*         px[1][2]=px[1][2]-deltax[5];*/
/*         px[2][0]=px[2][0]-deltax[6];*/
/*         px[2][1]=px[2][1]-deltax[7];*/
/*         px[2][2]=px[2][2]-deltax[8];*/

        /*build patch for different values of u and v*/
        u=v=0.0;
        for(uaxis=0;uaxis<5;uaxis++){
          u=0.0+(double)uaxis*0.25;
          for(vaxis=0;vaxis<5;vaxis++){
             v=0.0+(double)vaxis*0.25;

             /*construct surface patch*/
             /*column and row vectors*/
             rv[0]=u*u;
             rv[1]=u;
             rv[2]=1.0;
             cv[0]=v*v;
             cv[1]=v;
             cv[2]=1.0;

             /*calculate x,y,z*/
             x=matrix(rv,pxrect,cv);
             y=matrix(rv,pyrect,cv);
             z=matrix(rv,pzrect,cv);

             x_patch[uaxis][vaxis]=x;
             y_patch[uaxis][vaxis]=y;
             z_patch[uaxis][vaxis]=z;
             }
          }


/*           deltax[0]=x_patch[0][0]-contlx[0];*/
/*           deltax[1]=x_patch[0][2]-contlx[1];*/
/*           deltax[2]=x_patch[0][4]-contlx[2];*/
/*           deltax[3]=x_patch[2][0]-contlx[3];*/
/*           deltax[4]=x_patch[2][2]-contlx[4];*/
/*           deltax[5]=x_patch[2][4]-contlx[5];*/
/*           deltax[6]=x_patch[4][0]-contlx[6];*/
/*           deltax[7]=x_patch[4][2]-contlx[7];*/
/*           deltax[8]=x_patch[4][4]-contlx[8];*/

/*      }while((x_patch[2][2]-contlx[4])>0.001);*/
```

```
printf("\ncalculating curvature\n");
printf("\npercent completed : ");
uelement=velement=0;
for(upos=1;upos<198;upos+=10){
 uelement=(upos-1)/10;
 for(vpos=1;vpos<198;vpos+=10){
   velement=(vpos-1)/10;
   printf("%d",uelement*5);
   if(uelement<2)printf("\b");
     else printf("\b\b");
   u=v=0.0;
   for(uaxis=0;uaxis<3;uaxis++){
      u=0.0+(double)(upos+inc*(uaxis-1))*0.005;
      for(vaxis=0;vaxis<3;vaxis++){
         v=0.0+(double)(vpos+inc*(vaxis-1))*0.005;

         /*construct surface patch*/
         /*column and row vectors*/
         rv[0]=u*u;
         rv[1]=u;
         rv[2]=1.0;
         cv[0]=v*v;
         cv[1]=v;
         cv[2]=1.0;

         /*calculate x,y,z*/
         pxcurve[uaxis][vaxis]=matrix(rv,pxrect,cv);
         pycurve[uaxis][vaxis]=matrix(rv,pyrect,cv);
         pzcurve[uaxis][vaxis]=matrix(rv,pzrect,cv);

         }
  }
      u=v=0.0;
      for(uaxis=0;uaxis<5;uaxis++){
        u=0.0+(double)uaxis*0.25;
        for(vaxis=0;vaxis<5;vaxis++){
           v=0.0+(double)vaxis*0.25;


         /*curvature calculations*/
         /*calculate partial derivatives*/
         /* first derivative*/

         /* partial dx/du, dy/du, dz/du */
         /* set up column and row vectors */

         rv[0]=2.0*u;
         rv[1]=1.0;
         rv[2]=0.0;
         cv[0]=v*v;
         cv[1]=v;
         cv[2]=1.0;

         xu=matrix(rv,pxcurve,cv);
         yu=matrix(rv,pycurve,cv);
         zu=matrix(rv,pzcurve,cv);
```

```
/* partial dx/dv, dy/dv, dz/dv */
/* set up column and row vectors */

rv[0]=u*u;
rv[1]=u;
rv[2]=1.0;
cv[0]=2.0*v;
cv[1]=1.0;
cv[2]=0.0;

xv=matrix(rv,pxcurve,cv);
yv=matrix(rv,pycurve,cv);
zv=matrix(rv,pzcurve,cv);

/* partial second derivatives d2x/du2, d2y/du2, d2z/du2 */
/* set up column and row vectors */

rv[0]=2.0;
rv[1]=0.0;
rv[2]=0.0;
cv[0]=v*v;
cv[1]=v;
cv[2]=1.0;

xuu=matrix(rv,pxcurve,cv);
yuu=matrix(rv,pycurve,cv);
zuu=matrix(rv,pzcurve,cv);

/* partial second derivatives d2x/dv2, d2y/dv2, d2z/dv2 */
/* set up column and row vectors */

rv[0]=u*u;
rv[1]=u;
rv[2]=1.0;
cv[0]=2.0;
cv[1]=0.0;
cv[2]=0.0;

xvv=matrix(rv,pxcurve,cv);
yvv=matrix(rv,pycurve,cv);
zvv=matrix(rv,pzcurve,cv);

/* partial second derivatives d2x/duv, d2y/duv, d2z/duv */
/* set up column and row vectors */

rv[0]=2.0*u;
rv[1]=1.0;
rv[2]=0.0;
cv[0]=2.0*v;
cv[1]=1.0;
cv[2]=0.0;

xuv=matrix(rv,pxcurve,cv);
yuv=matrix(rv,pycurve,cv);
zuv=matrix(rv,pzcurve,cv);

E_=xu*xu+yu*yu+zu*zu;
```

```
            F_=xu*xv+yu*yv+zu*zv;
            G_=xv*xv+yv*yv+zv*zv;

            denom=(E_*G_-F_*F_);
            denom=sqrt(denom);

            e=(xuu*(yu*zv-zu*yv))-(yuu*(xu*zv-zu*xv))+(zuu*(xu*yv-yu*xv));
            if(e!=0.0 && denom!=0.0)e=e/denom;
            f=(xuv*(yu*zv-zu*yv))-(yuv*(xu*zv-zu*xv))+(zuv*(xu*yv-yu*xv));
            if(f!=0.0 && denom!=0.0)f=f/denom;
            g=(xvv*(yu*zv-zu*yv))-(yvv*(xu*zv-zu*xv))+(zvv*(xu*yv-yu*xv));
            if(g!=0.0 && denom!=0.0)g=g/denom;

            a=((E_*g-e*G_)*(E_*g-e*G_));
            b=(4*(E_*f-e*F_)*(F_*g-f*G_));
            complex=a-b;
            eq1=e*G_-E_*g;
            eq2=2.0*(F_*g-f*G_);

/*            if(eq2!=0.0)lmax=(eq1+sqrt(complex))/eq2;*/
/*            if(eq2!=0.0)lmin=(eq1-sqrt(complex))/eq2;*/
    /*      lmin=tan((3.142/2.0)+atan(lmax));*/

            lmax=1.0;
            lmin=-1.0;


            if((E_+2.0*F_*lmax+G_*lmax*lmax)!=0.0)
                kmax=(e+2.0*f*lmax+g*lmax*lmax)/(E_+2.0*F_*lmax+G_*lmax*lmax);
            if((E_+2.0*F_*lmin+G_*lmin*lmin)!=0.0)
                kmin=(e+2.0*f*lmin+g*lmin*lmin)/(E_+2.0*F_*lmin+G_*lmin*lmin);


            if(kmax!=0.0)r1=1.0/kmax;
            if(kmin!=0.0)r2=1.0/kmin;
            r1_patch[uaxis][vaxis]=r1;
            r2_patch[uaxis][vaxis]=r2;

           }/*next v*/
        }/*next u*/

        mean_r1=mean_r2=0.0;
        for(uaxis=0;uaxis<5;uaxis++)
            for(vaxis=0;vaxis<5;vaxis++){
                      mean_r1+=r1_patch[uaxis][vaxis];
                      mean_r2+=r2_patch[uaxis][vaxis];}
        mean_r1=mean_r1/25.0;
        mean_r2=mean_r2/25.0;
        av_mean=(mean_r1+mean_r2)/2.0;
        curve[uelement][velement]=av_mean;
        gauss_mean=mean_r1*mean_r2;
        /*apex position */
        if(av_mean>0.0)
          if(apex_r>av_mean){apex_r=av_mean;
                             apex_u=upos;
                             apex_v=vpos;}
        }/*next upos*/
```

```
        }/*next vpos*/

       cls();
       printf("\napex position = %d,%d",apex_u,apex_v);
       printf("\nplot contours y/n ? ");
       pausè=getch();
       printf("%c\n",pause);
       if(pause=='y'){
       /*calculate highest (closest) point of surface*/
             u=v=0.0;
             for(uaxis=0;uaxis<100;uaxis++){
                u=0.0+(double)uaxis*0.01;
                for(vaxis=0;vaxis<100;vaxis++){
                   v=0.0+(double)vaxis*0.01;

                   /*construct surface patch*/
                   /*column and row vectors*/
                   rv[0]=u*u;
                   rv[1]=u;
                   rv[2]=1.0;
                   cv[0]=v*v;
                   cv[1]=v;
                   cv[2]=1.0;

                   /*calculate x,y,z*/
                   x=matrix(rv,pxrect,cv);
                   y=matrix(rv,pyrect,cv);
                   z=matrix(rv,pzrect,cv);
                   if(x<xmin)xmin=x;

                   }
             }
       /*calculate apex y,z position*/
       u=apex_u*0.005;
       v=apex_v*0.005;

       /*construct surface patch*/
       /*column and row vectors*/
       rv[0]=u*u;
       rv[1]=u;
       rv[2]=1.0;
       cv[0]=v*v;
       cv[1]=v;
       cv[2]=1.0;

       /*calculate y,z*/
       apex_y=matrix(rv,pyrect,cv);/*pxrect*/
       apex_z=matrix(rv,pzrect,cv);
       con4(xmin,apex_u,apex_v,apex_y,apex_z);}
}



int l_upper_thresh(start_x,y,thresh)int start_x,y;WORD thresh;{
/*line drawn at upper threshold*/
    WORD pix_val;
```

```
    int x;
    for(x=start_x;x<768;x++){
       pix_val=ofg_rpixel(gaoi,x,y);
       if(pix_val>thresh){ofg_cvline(gaoi,x,0,512,0XFF);
                          break;}
       }
    return(x);
    }

int l_lower_thresh(start_x,y,thresh)int start_x,y;WORD thresh;{
/*line drawn at lower threshold*/
    WORD pix_val;
    int x;
    for(x=start_x;x<768;x++){
       pix_val=ofg_rpixel(gaoi,x,y);
       if(pix_val<thresh){ofg_cvline(gaoi,x,0,512,0XFF);
                          break;}
       }
    return(x);
    }

int p_upper_thresh(start_x,y,thresh)int start_x,y;WORD thresh;{
/*point drawn at upper threshold*/
    WORD pix_val;
    int x;
    for(x=start_x;x<768;x++){
       pix_val=ofg_rpixel(gaoi,x,y);
       if(pix_val>thresh){ofg_wpixel(gaoi,x,y,0XFF);
                          break;}
       }
    return(x);
    }

int p_lower_thresh(start_x,y,thresh)int start_x,y;WORD thresh;{
/*point drawn at lower threshold*/
    WORD pix_val;
    int x;
    for(x=start_x;x<768;x++){
       pix_val=ofg_rpixel(gaoi,x,y);
       if(pix_val<thresh){ofg_wpixel(gaoi,x,y,0XFF);
                          break;}
       }
    return(x);
    }

int n_upper_thresh(start_x,y,thresh)int start_x,y;WORD thresh;{
/*nothing drawn at upper threshold*/
    WORD pix_val;
    int x;
    for(x=start_x;x<768;x++){
       pix_val=ofg_rpixel(gaoi,x,y);
       if(pix_val>thresh)break;
       }
    return(x);
    }
int n_lower_thresh(start_x,y,thresh)int start_x,y;WORD thresh;{
/*nothing drawn at lower threshold*/
```

```
    WORD pix_val;
    int x;
    for(x=start_x;x<768;x++){
       pix_val=ofg_rpixel(gaoi,x,y);
       if(pix_val<thresh)break;
       }
    return(x);
    }


calib_pix(){
   int x1=0,x2=0,y=256;
   double size;
   WORD threshold=100;
   char line[10];
   cls();
   printf("\nthreshold = ");
   gets(line);
   threshold=(WORD)atoi(line);
        x1=l_lower_thresh(x1+1,y,threshold);
        x2=l_upper_thresh(x1+1,y,threshold);
   printf("\ndistance = ");
   gets(line);
   size=atof(line);
   dist_per_pixel=size/(double)(x2-x1);
   printf("\nx2=%d",x2);
   printf("\nx1=%d",x1);
   printf("\ndist_per_pixel=%lf",dist_per_pixel);
   gets(line);
   }
```

Appendix B.

Topography Shape Results

	|---------Topography system---------			|---------- PEK----------		
cornea	apex radius of curvature mm	apex distance mm	A.F. (flattening) mm/mm	apex radius of curvature mm	apex distance mm	A.F. (flattening) mm/mm
1	7.97	0.25	0.06	8.04	0.29	0.00
2	7.81	0.17	0.07	8.00	0.02	0.00
3	7.93	0.71	0.09	8.34	0.51	0.01
4	8.03	0.29	0.03	8.42	0.54	0.00
5	8.06	0.13	0.00	8.12	0.07	0.00
6	7.03	0.27	0.02	7.39	0.15	0.01
7	8.03	0.24	0.01	8.38	0.29	0.00
8	7.74	1.23	0.01	7.66	0.13	0.00
9	8.21	0.38	0.03	8.49	0.36	0.07
10	7.66	0.68	0.02	7.89	0.68	0.00
11	7.67	0.14	0.06	7.90	0.12	0.00
12	7.87	0.36	0.05	8.18	0.20	0.01
13	7.46	0.31	0.01	7.37	0.41	0.01
14	7.09	0.52	0.06	7.56	0.21	0.07
15	7.33	0.58	0.07	8.12	0.10	0.00
16	8.25	0.74	0.00	8.06	0.35	0.04
17	7.46	0.38	0.01	7.70	0.29	0.00
18	8.03	0.54	0.10	8.28	0.24	0.05
19	8.10	1.08	0.10	8.22	0.08	0.13
20	7.89	0.53	0.04	7.90	0.11	0.01
21	7.99	0.53	0.03	8.15	0.36	0.00
22	7.90	0.71	0.05	8.16	0.32	0.00
23	8.35	0.65	0.11	8.51	0.11	0.02
24	7.90	0.06	0.06	8.06	0.35	0.00
25	8.07	0.77	0.06	8.28	0.05	0.00
26	7.92	0.05	0.04	8.13	0.44	0.00
27	7.51	0.70	0.07	7.87	0.13	0.04
28	7.67	0.64	0.01	7.97	0.45	0.00
29	7.66	0.53	0.04	8.03	0.47	0.00
30	8.13	0.33	0.00	8.25	0.31	0.02

Appendix C.

config.sys

```
files=30
buffers=30
shell=c:\command.com /E:512 /p
device=c:\dos\ansi.sys
device=c:\drivers\hpglpltr.sys
device=c:\drivers\laserjet.sys
device=c:\drivers\ibmega.sys
device=c:\drivers\gsscgi.sys
country=044,,c:\dos\country.sys
```

autoexec.bat

```
@echo off
cls
set comspec=c:\command.com
path = c:\;c:\dos;c:\lc;
keyb uk,,c:\dos\keyboard.sys
ver
prompt $p $g
set INCLUDE=c:\lc;
MODE COM1:300,N,8,2,P
MODE LPT1:,,P
C:\MOUSE\MOUSE
PGI_PLS 127
cd c:\cfold
scar
```

C:\cfold\remove.bat

```
del %1.*
```

C:\cfold\restore.bat

```
copy b:%1.* c:\cfold
```

c:\cfold\backup.bat

```
copy %1.* b:
```

c:\lc\sc.bat to compile scar.c and link

```
rem Batch file(SCAR.BAT) for compiling and linking SCAR.C
programme
lc -ml -ccdmsuw -k2  c:\cfold\scar.c
Plink86 @SCAR.LNK
```

c:\lc\scar.lnk

```
OUTPUT C:\CFOLD\SCAR.EXE
MAP = C:\CFOLD\SCAR.LST
F                    I                    L                    E
C:\LC\L\C,C:\CFOLD\LIB,C:\IBMDAC\DACC,C:\LC\PGIF_CLL,C:\CFOLD\MIC
KEY,C:\CFOLD\TITLES,C:\CFOLD\S_PROCS,C:\CFOLD\SCAR
LIB C:\LC\L\LCM, C:\LC\L\LC
```

c:\lc\s_procs.bat to compile procedure file s_procs

```
lc -ml -ccdmsuw -k2 c:\cfold\s_procs.c
```

c:\lc\titles.c to compile titles.c

```
rem Batch file(TITLES.BAT) for compiling and linking TITLES.C
module
lc -ml -ccdmsuw -k2 c:\cfold\titles.c
```

```
/*description - mouse driver interscepts bios calls by placing*/
/*vector at bios address. Thus when screen mode is changed, mouse*/
/*mode is changed. But need mouse mode at resolution of 1 pixel*/
/*In text mode resolution 8 pixels. Therefore find bios address*/
/*when mouse driver not loaded. Then run scar program when mouse*/
/*driver loaded, and set mouse mode to graphics(bios graphics call)*/
/*Replace bios and then (in factors();) reset bios to text mode with*/
/*mouse still in graphics mode. Then replace mouse driver.          */

/*screen.c to display screen mode and bios interrupt 10 address*/
#include<stdio.h>
#include<dos.h>
#define BIOS_INT 0X10
union REGS in,out;
unsigned char far *ptr;
main(){
 in.h.ah=15;
 in.h.al=0;
 int86(BIOS_INT,&in,&out);
 printf("\ncurrent screen mode=%d",out.h.al);
 ptr=0x00000040;
 printf("\n bios 10 address=%x:%x:%x:%x",*ptr,*(ptr+1),*(ptr+2),*(ptr+3));
 }
```

```
/*cell.h file*/
#define Cxmin 0 /*cursor*/
#define Cxmax 632
#define Cymin 0
#define Cymax 192
#define Xmin 40 /*viewport*/
#define Xmax 500
#define Ymin 0
#define Ymax 200
#define Xdmin 505 /*data*/
#define Xdmax 700
#define Ydmin 0
#define Ydmax 270
#define Xgmin 40 /*graph*/
#define Xgmax 500
#define Ygmin 205
#define Ygmax 270
struct record{char name[20];
              char date[10];
              char eye[3];
              char sex[3];
              char age[4];
              char hosp[12];/*hospital number*/
              char vas[12];/*V.A. with specs*/
              char vac[12];/*V.A. with contact lenses*/
              char diags1[22];
              char diags2[22];
              char diags3[22];
              char diags4[22];
              };
```

```
/*titles.c file*/
title1(){
   char *t[6],*p;
   int x,y,s=255,i;
   t[0]="place boundry";
   t[1]="buttons";
   t[2]="L..end";
   t[3]="M..remove point";
   t[4]="R..place point";
   t[5]="Hold R..new boundry";
   tblank();
   sccol(&s);
   sfcol(&s);
   scsp(&s);
   x=540;y=20;
   moveto(&x,&y);
   icsp(&s);
   p=t[0];
   i=0;while(*(p+i)){ pchar(p+i);i++;}
   x=570;y=60;
   moveto(&x,&y);
   p=t[1];
   i=0;while(*(p+i)){pchar(p+i);i++;}
   x=520;y=90;
   moveto(&x,&y);
   p=t[2];
   i=0;while(*(p+i)){pchar(p+i);i++;}
   y=120;
   moveto(&x,&y);
   p=t[3];
   i=0;while(*(p+i)){pchar(p+i);i++;}
   y=150;
   moveto(&x,&y);
   p=t[4];
   i=0;while(*(p+i)){pchar(p+i);i++;}
   y=180;
   moveto(&x,&y);
   p=t[5];
   i=0;while(*(p+i)){pchar(p+i);i++;}
   }/*end proc*/
title2(){
   char *t[3],*p;
   int x,y,s=255,i;
   t[0]="blank area";
   t[1]="buttons";
   t[2]="Any button..end";
   tblank();
   sfcol(&s);
   scsp(&s);
   x=540;y=20;moveto(&x,&y);
   p=t[0];
   i=0;while(*(p+i)){pchar(p+i);i++;}
   x=570;y=60;moveto(&x,&y);
   p=t[1];
   i=0;while(*(p+i)){pchar(p+i);i++;}
   x=520;y=90;moveto(&x,&y);
   p=t[2];
   i=0;while(*(p+i)){pchar(p+i);i++;}
   }/*end proc*/
title3(){
   char *t[4],*p;
   int x,y,s=255,i;
   t[0]="intensity profile";
   t[1]="any button: profile";
```

```
    t[2]="then";
    t[3]="any button: end";
    tblank();
    sfcol(&s);
    scsp(&s);
    x=530;y=20;moveto(&x,&y);
    p=t[0];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=90;moveto(&x,&y);
    p=t[1];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=570;y=120;moveto(&x,&y);
    p=t[2];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=150;moveto(&x,&y);
    p=t[3];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    }/*end proc*/
title4(){
    char *t[3],*p;
    int x,y,s=255,i;
    t[0]="threshold";
    t[1]="L..set";
    t[2]="M & R..display";
    tblank();
    sfcol(&s);
    scsp(&s);
    x=540;y=20;moveto(&x,&y);
    p=t[0];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=90;moveto(&x,&y);
    p=t[1];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=120;moveto(&x,&y);
    p=t[2];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    }/*end proc*/
title5(){
    char *t[6],*p;
    int x,y,s=255,i;
    t[0]="stretching";
    t[1]="any button: lower limit";
    t[2]="any button: upper limit";
    t[3]="any button: central";
    t[4]="threshold";
    t[5]="then";
    tblank();
    sfcol(&s);
    scsp(&s);
    x=560;y=20;moveto(&x,&y);
    p=t[0];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=508;y=90;moveto(&x,&y);
    p=t[1];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=570;y=120;moveto(&x,&y);
    p=t[5];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=508;y=150;moveto(&x,&y);
    p=t[2];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=570;y=180;moveto(&x,&y);
    p=t[5];
    i=0;while(*(p+i)){pchar(p+i);i++;}
```

```
    x=508;y=210;moveto(&x,&y);
    p=t[3];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=600;y=225;moveto(&x,&y);
    p=t[4];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    }/*end proc*/
title6(){
    char *t[5],*p;
    int x,y,s=255,i;
    t[0]="stretching";
    t[1]="buttons";
    t[2]="L..cancel";
    t[3]="M..redo";
    t[4]="R ..accept";
    tblank();
    sccol(&s);
    sfcol(&s);
    scsp(&s);
    x=560;y=20;moveto(&x,&y);
    p=t[0];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=570;y=60;moveto(&x,&y);
    p=t[1];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=90;moveto(&x,&y);
    p=t[2];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    y=120;
    moveto(&x,&y);
    p=t[3];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    y=150;
    moveto(&x,&y);
    p=t[4];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    }/*end proc*/
title7(){
    char *t[5],*p;
    int x,y,s=255,i;
    t[0]="place square";
    t[1]="any button -";
    t[2]="top left point";
    t[3]="    then";
    t[4]="bottom right point";
    tblank();
    sfcol(&s);
    scsp(&s);
    x=560;y=20;moveto(&x,&y);
    p=t[0];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=90;moveto(&x,&y);
    p=t[1];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=120;moveto(&x,&y);
    p=t[2];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=560;y=150;moveto(&x,&y);
    p=t[3];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=180;moveto(&x,&y);
    p=t[1];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=210;moveto(&x,&y);
```

```
    p=t[4];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    }/*end proc*/
 title8(){
    char *t[4],*p;
    int x,y,s=255,i;
    t[0]="place square";
    t[1]="buttons";
    t[2]="L..main menu";
    t[3]="R & M..draw box";
    tblank();
    sfcol(&s);
    scsp(&s);
    x=560;y=20;moveto(&x,&y);
    p=t[0];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=570;y=60;moveto(&x,&y);
    p=t[1];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=90;moveto(&x,&y);
    p=t[2];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=120;moveto(&x,&y);
    p=t[3];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    }/*end proc*/
 title9(){
    char *t[2],*p;
    int x,y,s=255,i;
    t[0]="pupil centre";
    t[1]="place with any button";
    tblank();
    sfcol(&s);
    scsp(&s);
    x=540;y=20;moveto(&x,&y);
    p=t[0];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=90;moveto(&x,&y);
    p=t[1];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    }/*end proc*/
 title10(){
    char *t[2],*p;
    int x,y,s=255,i;
    t[0]="background";
    t[1]="choose with any button";
    tblank();
    sfcol(&s);
    scsp(&s);
    x=540;y=20;moveto(&x,&y);
    p=t[0];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    x=520;y=90;moveto(&x,&y);
    p=t[1];
    i=0;while(*(p+i)){pchar(p+i);i++;}
    }/*end proc*/
```

```
/*s_procs.c object module for scar.c*/
#include<dos.h>
#include<fcntl.h>
#include<error.h>
#include<stdio.h>
#include<cell.h>
extern int x_centre,y_centre,background;
extern long int grey[5][14];
extern double dist_per_pixel,fraction[5];
extern struct record patient;
menu(){
   cls();
   printf("                    \033[7mCORNEAL SCAR MEASUREMENT SYSTEM\033[0m\n
   printf("      \033[7mTEST FUNCTIONS\033[0m                                  \03
   printf("  a..mouse  test                                  c..Real-time view\n
   printf("  b..cursor test                                  d..Grab image\n\n")
   printf("                                                  m..save image\n\n")
   printf("                                                  n..restore image\n"
   printf("   \033[7mFEATURE EXTRACTION\033[0m                                  \033
   printf("  f..Intensity profile                            i..Contrast stretch
   printf("  g..Grey level histogram                         j..Thresholding \n\
   printf("  h..Measurement of area                          k..Histogram equali
   printf("                                                  l..reverse contrast
   printf("  p..Redo pupil centre                            s..smoothing\n\n");
   printf("  q..Check partitions                             r..original\n\n");
   printf("  w..enter patient details                        x..export data file
   printf("                                                 e..End\n");
   printf("             ENTER OPTION==>");
   printf("%c",7);
   }/*end proc*/
initpat(){
   strcpy(patient.name,"");
  strcpy(patient.date,"");
  strcpy(patient.eye,"");
  strcpy(patient.sex,"");
  strcpy(patient.age,"");
  strcpy(patient.hosp,"");
  strcpy(patient.vas,"");
  strcpy(patient.vac,"");
  strcpy(patient.diags1,"");
  strcpy(patient.diags2,"");
  strcpy(patient.diags3,"");
  strcpy(patient.diags4,"");
}
export(){
/*write grey level results to file in ASCII format*/

   FILE *fp;
   int i,j;
   long int histo[14];
   char namex[15],namez[15],pause;
   char namea[12],nameb[12],namec[12],named[12],namee[12];
   double pixel_area;
   pixel_area=dist_per_pixel*2*dist_per_pixel;
   for(i=0;i<14;i++)histo[i]=0;

   cls();
   printf("\n%cinput file name to export results :",7);
   gets(namex);
   strncpy(namez,namex,8);
   strcat(namez,".z");
   if((fp=fopen(namez,"w"))==NULL){
      printf("\n%ccannot open file %s",7,namez);
      printf("\npress any key to continue");
```

```
      pause=getch();
      return;}
    strcpy(namea,namex);
    strcat(namea,"_a");
    strcpy(nameb,namex);
    strcat(nameb,"_b");
    strcpy(namec,namex);
    strcat(namec,"_c");
    strcpy(named,namex);
    strcat(named,"_d");
    strcpy(namee,namex);
    strcat(namee,"_e");
    fprintf(fp,"%s %s %s %s %s \n",namea,nameb,namec,named,namee);
    for(i=0;i<14;i++){
      for(j=0;j<5;j++){histo[i]=histo[i]+grey[j][i];
                       fprintf(fp,"%f ",(double)histo[i]);}
      fprintf(fp,"\n");
      }
    fclose(fp);
    }

details(){
   char line[100];
   cls();
   printf("Patient:                    :[%s]\033[1;9H",patient.name);
   gets(line);
   if(strlen(line)>0)strncpy(patient.name,line,18);
   printf("\nDate:         :[%s]\033[3;6H",patient.date);
   gets(line);
   if(strlen(line)>0)strncpy(patient.date,line,8);
   printf("\nEye: :[%s]\033[5;5H",patient.eye);
   gets(line);
   if(strlen(line)>0)strncpy(patient.eye,line,1);
   printf("\nSex: :[%s]\033[7;5H",patient.sex);
   gets(line);
   if(strlen(line)>0)strncpy(patient.sex,line,1);
   printf("\nAge:  :[%s]\033[9;5H",patient.age);
   gets(line);
   if(strlen(line)>0)strncpy(patient.age,line,2);
   printf("\nHospital Number:        :[%s]\033[11;17H",patient.hosp);
   gets(line);
   if(strlen(line)>0)strncpy(patient.hosp,line,10);
   printf("\nV.A.(Spectacles):         :[%s]\033[13;18H",patient.vas);
   gets(line);
   if(strlen(line)>0)strncpy(patient.vas,line,10);
   printf("\nV.A.(Contact Lenses):         :[%s]\033[15;22H",patient.vac);
   gets(line);
   if(strlen(line)>0)strncpy(patient.vac,line,10);
   printf("\nComment line 1:                    :");
   printf("\n              [%s]\033[17;16H",patient.diags1);
   gets(line);
   if(strlen(line)>0)strncpy(patient.diags1,line,20);
   printf("\nComment line 2:                    :");
   printf("\n              [%s]\033[19;16H",patient.diags2);
   gets(line);
   if(strlen(line)>0)strncpy(patient.diags2,line,20);
   printf("\nComment line 3:                    :");
   printf("\n              [%s]\033[21;16H",patient.diags3);
   gets(line);
   if(strlen(line)>0)strncpy(patient.diags3,line,20);
   printf("\nComment line 4:                    :");
   printf("\n              [%s]\033[23;16H",patient.diags4);
   gets(line);
   if(strlen(line)>0)strncpy(patient.diags4,line,20);
```

```
}
pat_details(){
      lprintf("PA10,680;LBPATIENT.;");
    lprintf("CP;CP;LB   DATE.;CP;LB    EYE.;CP;LBDETAILS.;");
    lprintf("PA10,680;CP;LB  %s.;",patient.name);
    lprintf("CP;LB         %s.;",patient.date);
    lprintf("CP;LB         %s.;",patient.eye);
    lprintf("CP;CP;");
    lprintf("LBSex  %s.;",patient.sex);
    lprintf("CP;LBAge  %s.;",patient.age);
    lprintf("CP;LBHospital Number .;");
    lprintf("CP;LB%s.;",patient.hosp);
    lprintf("CP;LBV A (Spectacles).;");
    lprintf("CP;LB%s.;",patient.vas);
    lprintf("CP;LBV A (Contact Lenses).;");
    lprintf("CP;LB%s.;",patient.vac);
    lprintf("CP;LBComments.");
    lprintf("CP;LB%s.;",patient.diags1);
    lprintf("CP;LB%s.;",patient.diags2);
    lprintf("CP;LB%s.;",patient.diags3);
    lprintf("CP;LB%s.;",patient.diags4);
}
dataplot(){
char pause;
int i,j,start_pos,horiz,max_histo,g;
long int histo[14];
double pixel_area;
pixel_area=dist_per_pixel*2*dist_per_pixel;
    /*DRAW BOX*/
    lprintf("PU;PA0,0;PD;PA0,720;PA1000,720;PA1000,0;PA0,0;PU;");
    /*FINISH PATIENT BOX*/
    lprintf("PA280,720;PD;PA280,0;PU;");
    /*PRINT TITLES*/
    pat_details();
    lprintf("SR0.6,1.0;CP;CP;LBPUPIL RADIUS   SCAR FRACTION.;");
    lprintf("DT%;");
    lprintf("SR;CP;CP;LB   0.5mm    %6.2f%;",fraction[0]);
    lprintf("CP;LB   1.0mm    %6.2f%;",fraction[1]);
    lprintf("CP;LB   1.5mm    %6.2f%;",fraction[2]);
    lprintf("CP;LB   2.0mm    %6.2f%;",fraction[3]);
    lprintf("CP;LB   2.5mm    %6.2f%;",fraction[4]);
    lprintf("DT.;");
    lprintf("PA580,680;LBSCAR LOCATION.;");
    /*PLOT LOCATION GRID*/
    lprintf("PA340,360;PD;PA940,360;PU;");
    lprintf("PA340,360;XT;PA440,360;XT;PA540,360;XT;");
    lprintf("PA640,360;XT;PA740,360;XT;PA840,360;XT;PA940,360;XT;");
    lprintf("PA640,360;CI100;CI300;");
    /*change test size*/
    lprintf("SR0.4,0.7;PA740,330;DT;LB1.0 mm%c;",3);
    lprintf("PA840,330;LB2.0 mm%c;PA940,330;LB3.0 mm%c;DT.;",3,3);
    /*restore text size*/
    lprintf("SR;");
    lprintf("PA740,10;LBBACKGROUND=%d.;",background);
    lprintf("PG;"); /*PAGE FEED*/

    /*second page*/
    /*DRAW BOX*/
    lprintf("PU;PA0,0;PD;PA0,720;PA1000,720;PA1000,0;PA0,0;PU;");
    /*FINISH PATIENT BOX*/
    lprintf("PA280,720;PD;PA280,0;PU;");
    /*PRINT TITLES*/
    pat_details();
    lprintf("PA400,680;LBNORMALISED SCAR GREY LEVEL HISTOGRAM.;");
```

```
    lprintf("PA450,650;LB(PIXELS AGAINST GREY LEVELS).;");
    lprintf("SR0.6,1.0;PA290,600;LBPUPIL RADIUS .;");
    lprintf("DTq;PA600,600;LBAREA PER PIXEL=%8.6f mm sq;SR;DT.;",pixel_area);
    lprintf("PA400,550;PD;PA400,460;PA960,460;PU;");
    lprintf("PA400,440;PD;PA400,350;PA960,350;PU;");
    lprintf("PA400,330;PD;PA400,240;PA960,240;PU;");
    lprintf("PA400,220;PD;PA400,130;PA960,130;PU;");
    lprintf("PA400,110;PD;PA400,20;PA960,20;PU;");
    lprintf("PA300,510;DT;LB0.5mm%c;PA300,400;LB1.0mm%c;PA300,280;LB1.5mm%c;",3
    lprintf("PA300,170;LB2.0mm%c;PA300,60;LB2.5mm%c;DT.;",3,3);
    for(i=0;i<14;i++)histo[i]=0;
    lprintf("SR0.3,1.0;DT);");
    for(j=0;j<5;j++){
       for(i=0;i<14;i++)histo[i]=histo[i]+grey[j][i];
       start_pos=460-(j*110);
           max_histo=0.0;
       for(i=0;i<14;i++)if(histo[i]>max_histo)max_histo=histo[i];
       lprintf("PA400,%d;PD;",start_pos);
       for(i=0;i<14;i++){
        horiz=i*40;
        g=i+1;
        lprintf("PA%d,%d;",400+horiz,start_pos+(90*histo[i]/max_histo));
        lprintf("PA%d,%d;",440+horiz,start_pos+(90*histo[i]/max_histo));
        lprintf("PU;PA%d,%d;",400+horiz+1,start_pos-10);
        lprintf("LB%5ld(%d);",histo[i],g);
        lprintf("PA%d,%d;PD;",440+horiz,start_pos+(90*histo[i]/max_histo));
        }
       lprintf("PU;");
       }/*next radius*/
       lprintf("SR;DT.;");
   lprintf("PG;");
   flushall();
   }

pupil(){
   char pause;
   int x,y,s,xpos,ypos,xc,yc,b;
   int b1,b2,b3,b4,xl,xr,yt,yb;
   cls();
   printf("%c",7);
   printf("\n place cursor inside pupil");
   title9();
   cursor_block(&xpos,&ypos,&b1,&b2,&b3,&b4);
   x=xpos;
   s=255;sccol(&s);
   do{x--;
      rpix(&x,&ypos,&s);
      }while(s<50);
   xl=x;
   x=xpos;
   do{x++;
      rpix(&x,&ypos,&s);
      }while(s<50);
   xr=x;
   y=ypos;
   do{y--;
      rpix(&xpos,&y,&s);
      }while(s<50);
   yt=y;
   y=ypos;
   do{y++;
      rpix(&xpos,&y,&s);
      }while(s<50);
   yb=y;
```

```
    xc=xl+(xr-xl)/2;
    yc=yt+(yb-yt)/2;
    for(y=Ymin+1;y<=Ymax-1;y++)plot(&xc,&y);
    for(x=Xmin+1;x<=Xmax-1;x++)plot(&x,&yc);
    printf("%c",7);
    printf("\n\n is the pupil centre correct y/n?");
    pause=getch();
    if(pause=='n'){printf("%c",7);
                   printf("\n\n place cursor at pupil centre");
                   cursor_block(&xpos,&ypos,&b1,&b2,&b3,&b4);
                   xc=xpos;yc=ypos;
                   s=255;sccol(&s);
                   for(y=Ymin+1;y<=Ymax-1;y++)plot(&xc,&y);
                   for(x=Xmin+1;x<=Xmax-1;x++)plot(&x,&yc);
                   printf("%c",7);
                   printf("\n\npress any key to continue");
                   pause=getch();
                   }
    x_centre=xc;y_centre=yc;
    tblank();
    original();
    cls();
    printf("%c",7);
    title10();
    curs_point(&xpos,&ypos,&b1,&b2,&b3,&b4);
    rpix(&xpos,&ypos,&b);
    background=b;
    printf("\n background threshold=%d",background);
    printf("%c\nany key to continue",7);
    pause=getch();
    tblank();
    original();
    }/*end proc*/
save_image(){
    long int pos=0;
    int fp,fpx,x,y,s,xpos,ypos,s_line,num;
    char name[15],namex[15],pause,d[4][460];
    struct record{
                  double length_cal;
                 };
    struct record saved;
    cls();
    printf("\ninput file name to store data : ");
    gets(name);
    strcpy(namex,name);
    strcat(namex,".x");
    strcat(name,".dat");
    saved.length_cal=dist_per_pixel;
    fpx=dcreat(namex,0);
    if(fpx==-1){printf("\ncannot create file");
                printf("\npress any key to continue");
                pause=getch();
                return;}/*end if*/
    dseek(fpx,pos,2);
    dwrite(fpx,(char*)&saved,sizeof(saved));
    dclose(fpx);
    fp=dcreat(name,0);
    if(fp==-1){printf("\ncannot create file");
                printf("\npress any key to continue");
                pause=getch();
                return;}/*end if*/
    dseek(fp,pos,2);
    s_line=0;
    for(num=0;num<=49;num++){
```

```
        for(y=0;y<=3;y++){
           for(x=0;x<=459;x++){
              xpos=x+40;ypos=s_line+y;
              rpix(&xpos,&ypos,&s);
              d[y][x]=(char)s;
              }/*next x*/
           }/*next y*/
        dwrite(fp,(char*)d,1840);
        s_line=s_line+4;
        printf(".");
        }/*next num*/
     dclose(fp);
     }/*end proc*/
restore_image(){
     long int pos=0;
     int fp,fpx,count,x,y,s,xpos,ypos;
     int w,h,p1,x1,y1,p2,x2,y2,n,q;
     char name[15],namex[15],pause;
     char d[4][460];
     int s_line,num;
     struct record{
                   double length_cal;
                   };
     struct record saved;
     cls();
     printf("\ninput file name to restore data : ");
     gets(name);
     strcpy(namex,name);
     strcat(namex,".x");
     strcat(name,".dat");
     fpx=dopen(namex,O_RDONLY);
     if(fpx==-1){printf("\nfile %s cannot be opened",namex);
                 printf("\npress any key to continue");
                 pause=getch();
                 return;
                 }/*end if*/
     dseek(fpx,pos,0);
     count=dread(fpx,(char*)&saved,sizeof(saved));
     dclose(fpx);
     dist_per_pixel=saved.length_cal;
     fp=dopen(name,O_RDONLY);
     if(fp==-1){printf("\nfile %s cannot be opened",name);
                 printf("\npress any key to continue");
                 pause=getch();
                 return;
                 }/*end if*/
     cls();
     printf("dist_per_pixel=%f",dist_per_pixel);
     pinit();
     lutinitg();
     blank();
     boxes();
     s=1;scwp(&s);
     s=1;scdp(&s);
     s_line=0;
     dseek(fp,pos,0);
     for(num=0;num<=49;num++){
        dread(fp,(char*)d,1840);
        for(y=0;y<=3;y++){
           for(x=0;x<=459;x++){
              xpos=x+40;ypos=s_line+y;
              s=(int)d[y][x];
              sccol(&s);
              plot(&xpos,&ypos);
```

```
            }/*next x*/
          }/*next y*/
       s_line=s_line+4;
       }/*next num*/
       dclose(fp);
       istyle(&q);
       n=0;sstyle(&n);
       p1=1;x1=41;y1=1;
       w=459;h=199;
       p2=2;x2=155;y2=45;
       copy(&w,&h,&p1,&x1,&y1,&p2,&x2,&y2);
       sstyle(&q);
       pupil();
   }/*end proc*/
cam_test(){
   int t;
   char pause;
   pinit();
   lutinitg();
   t=1;preview(&t);
   cls();
   printf("\npress any key to return to main menu");
   pause=getch();
   t=0;preview(&t);
   }/*end proc*/
joy_test(){
   int jx,jy,s1,s2,s3,s4;
   int stat,adapt=0;
   long int count=50;
   char pause;
   do{
      joy_pos(&jx,&jy);
      buttons(&s1,&s2,&s3,&s4);
      cls();
      printf("\nx=%d  y=%d",jx,jy);
      printf("\n%d%d%d%d",s1,s2,s3,s4);
      delay(adapt,count,&stat);
     }while(s1==1);
      printf("\npress any key to return to main menu");
      pause=getch();
  }/*end proc*/
viewport(){
     int x,y,s;
     x=Xmin;
     y=Ymin;
     s=255;sccol(&s);
     moveto(&x,&y);
     x=Xmin;y=Ymax;
     lineto(&x,&y);
     x=Xmax;y=Ymax;
     lineto(&x,&y);
     x=Xmax;y=Ymin;
     lineto(&x,&y);
     x=Xmin;y=Ymin;
     lineto(&x,&y);
     }/*end of viewport*/
boxes(){
     int x,y,s;
     viewport();
     s=255;sccol(&s);
     x=Xdmin;y=Ydmin;
     moveto(&x,&y);
     x=Xdmin;y=Ydmax;
     lineto(&x,&y);
```

```
        x=Xdmax;y=Ydmax;
        lineto(&x,&y);
        x=Xdmax;y=Ydmin;
        lineto(&x,&y);
        x=Xdmin;y=Ydmin;
        lineto(&x,&y);
        /*end dataport*/
        x=Xgmin;y=Ygmin;
        moveto(&x,&y);
        x=Xgmin;y=Ygmax;
        lineto(&x,&y);
        x=Xgmax;y=Ygmax;
        lineto(&x,&y);
        /*end graphport*/
        }/*end proc*/
blank(){
        int s,x,y,xl,yl;
        s=0;sccol(&s);
        x=0;y=0;
        moveto(&x,&y);
        xl=Xmin;
        yl=Ymax;
        rfill(&xl,&yl);
        x=0;y=Ymax;
        moveto(&x,&y);
        xl=768;yl=288-Ymax;
        rfill(&xl,&yl);
        x=Xmax;y=0;
        moveto(&x,&y);
        xl=768-Xmax;yl=288;
        rfill(&xl,&yl);
        }/*end proc*/
grab_image(){
        int t,w,h,p1,x1,y1,p2,x2,y2,n,q,;
        char pause,mag;
        pinit();
        lutinitg();/*initialise screen*/
        t=1;preview(&t);
        cls();
        printf("\nwhen ready - press any key to grab image");
        pause=getch();
        t=0;preview(&t);
        istyle(&q);
        fgrab();
        n=0;sstyle(&n);
        p1=1;x1=0;y1=0;
        w=760;h=280;
        p2=2;x2=0;y2=0;
        copy(&w,&h,&p1,&x1,&y1,&p2,&x2,&y2);
        p1=1;x1=154;y1=44;
        w=460;h=200;
        p2=1;x2=40;y2=0;
        copy(&w,&h,&p1,&x1,&y1,&p2,&x2,&y2);
        blank();
        boxes();
        sstyle(&q);
        pupil();
        cls();
        printf("%c",7);
        printf("\n specify slit lamp magnification");
        printf("\n\n  1..30x");
        printf("\n\n  2..16x");
        printf("\n\n  3..10x");
        printf("\n\n\n magnification selection==>");
```

```
  mag=getch();
  printf("%c",mag);
  switch(mag){
            case '1':dist_per_pixel=0.0083;
                     break;
            case '2':dist_per_pixel=0.0144;
                     break;
            case '3':dist_per_pixel=0.0260;
                     break;
            }/*end case*/
   printf("\n%c any key to continue",7);
   pause=getch();
}/*end proc*/

check(){
   static int t=1;
   if(t==1)t=2;
     else t=1;
   scdp(&t);
   }

 original(){
   int w,h,p1,x1,y1,p2,x2,y2,n,q;
   istyle(&q);
   n=0;sstyle(&n);
   p1=2;x1=154;y1=44;
   w=460;h=200;
   p2=1;x2=40;y2=0;
   copy(&w,&h,&p1,&x1,&y1,&p2,&x2,&y2);
   blank();
   boxes();
   sstyle(&q);
   }
```

```
/*scar.c program*/
#include <dos.h>
#include <fcntl.h>
#include <error.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cell.h>
#include <mickey.h>
union REGS in_regs,out_regs;
int _STACK=32000;
int _HEAP=32000;
unsigned char *ptr;
unsigned char mouse_ptr[4];
char opt,pause;
float fjx,fjy,fgx,fgy;
int *boundry();
int x_centre,y_centre,background=0;
long int grey[5][14];
float *grey_levels();
double dist_per_pixel=1.0,fraction[5];
struct record patient;

main(){
    factors();
    while (opt!='e' && opt!='E'){
       menu();
       opt=getch();
       printf("\033[7m%c\033[0m",opt);
       switch(opt){
           case 'a':joy_test();
                    break;
           case 'b':cursor_test();
                    break;
           case 'c':cam_test();
                    break;
           case 'd':grab_image();
                    break;
           case 'f':profile();
                    break;
           case 'g':block_grey();
                    break;
           case 'h':cal_area();
                    break;
           case 'i':stretch();
                    break;
           case 'j':threshold();
                    break;
           case 'k':equalise();
                    break;
           case 'l':reverse();
                    break;
           case 'm':save_image();
                    break;
           case 'n':restore_image();
                    break;
           case 's':all_smooth();
                    break;
           case 'r':original();
                    break;
           case 'q':check();
                    break;
           case 'p':pupil();
                    break;
```

```
            case 'w':details();
                     break;
            case 'x':export();
                     break;
            }/*end case*/
         }/*end while*/
      cls();
      }/*end main*/

factors(){
   int i;
   fjx=(float)(Xmax-Xmin)/(float)(Cxmax-Cxmin);
   fjy=(float)(Ymax-Ymin)/(float)(Cymax-Cymin);
   fgx=(float)(Xgmax-Xgmin)/255.0;
   fgy=(float)(Ygmax-Ygmin)/255.0;
   in_regs.h.ah=0;in_regs.h.al=16;int86(0x10,&in_regs,&out_regs);
   mm_init();
   ptr=(unsigned char*)NULL+0x40;/*bios int 10 address*/
   for(i=0;i<4;i++)mouse_ptr[i]=*(ptr+i);
   *ptr=0x75;
   *(ptr+1)=0x07;
   *(ptr+2)=0x00;
   *(ptr+3)=0xc0;
   in_regs.h.al=3;int86(0x10,&in_regs,&out_regs);
   for(i=0;i<4;i++)*(ptr+i)=mouse_ptr[i];
   for(i=0;i<5;i++)fraction[i]=0.0;
   initpat();
   }/*end proc*/
stretch(){
   int x,y,shade;
   for(y=1;y<199;y++){
      for(x=41;x<=499;x++){
         rpix(&x,&y,&shade);
         if(shade<=127)shade=shade*2;
         sccol(&shade);
         plot(&x,&y);
         }/*next x*/
      }/*next y*/
   }/*end proc*/
reverse(){
   int x,y,shade;
   for(x=Xmin+1;x<=Xmax-1;x++){
      for(y=Ymin+1;y<=Ymax-1;y++){
         rpix(&x,&y,&shade);
         shade=220-shade;
         sccol(&shade);
         plot(&x,&y);
         }/*next y*/
      }/*next x*/
    }/*end proc*/
all_smooth(){
   int x1,y1,x2,y2;
   x1=42;y1=2;x2=498;y2=198;
   smoothing(&x1,&y1,&x2,&y2);
   }/*end proc*/
smoothing(x1,y1,x2,y2)int *x1,*y1,*x2,*y2;{
   int square[9],shade,x,y;
      for(y=*y1+2;y<=*y2-2;y++){
         for(x=*x1+2;x<=*x2-2;x++){
            neighbors(square,x,y);
            sort(square);
            shade=square[4];
            sccol(&shade);
            plot(&x,&y);
```

```
            }/*next x*/
          }/*next y*/
     }/*end proc*/
neighbors(square,xc,yc)int square[],xc,yc;{
   int x,y,shade,count=0;
   for(x=xc-1;x<=xc+1;x++){
       for(y=yc-1;y<=yc+1;y++){
          shade=0;
          rpix(&x,&y,&shade);
          square[count]=shade;
          count++;
          }/*next y*/
       }/*next x*/
    }/*end proc*/
sort(square)int square[];{
   int a,b,t;
   for(a=1;a<9;++a){
     for(b=9-1;b>=a;--b){
       if(square[b-1]>square[b]){
          t=square[b-1];
          square[b-1]=square[b];
          square[b]=t;
          }/*end if*/
       }/*next b*/
     }/*next a*/
   }/*end proc*/

place_corners(tlx,tly,brx,bry)int *tlx,*tly,*brx,*bry;{
   long int count;
   int pos[2],n,shade=230,b1,b2,b3,b4;
   title7();
   for(n=0;n<=1;n++){
      cursor_block(pos,pos+1,&b1,&b2,&b3,&b4);
      place_cursor(pos,&shade);
      for(count=0;count<=20000;count++);/*debounce*/
        if(n==0){*tlx=pos[0];
                 *tly=pos[1];}
          else{*brx=pos[0];
               *bry=pos[1];}/*end if-else*/
      }/*next n*/
    title8();
    do{
       buttons(&b1,&b2,&b3,&b4);
      }while(b1==1 && b2==1 && b3==1 && b4==1);
    if(b1==0)return(0);
    draw_square(*tlx,*tly,*brx,*bry);
    return(1);
    }/*end proc*/
draw_square(x1,y1,x2,y2)int x1,y1,x2,y2;{
    int s=230;
    sccol(&s);
    moveto(&x1,&y1);
    lineto(&x1,&y2);
    lineto(&x2,&y2);
    lineto(&x2,&y1);
    lineto(&x1,&y1);
    }/*end proc*/
mean(x,y)int x,y;{
   int total=0;
   int grey,xs,ys,xpos,ypos;
   for(xs=0;xs<=2;xs++){
      for(ys=0;ys<=2;ys++){
          xpos=x-1+xs;
          ypos=y-1+ys;
```

```
            rpix(&x,&y,&grey);
            total+=grey;
            }/*next y*/
         }/*next x*/
      total=total/9;
      return(total);
      }/*end proc*/

   adjust_lut(level) int level[3];{
      int i,value,a[3];
      for(i=0;i<=220;i++){
         value=i;
         if((i>level[0])&&(i<level[2]))value=level[0];
         if((i<level[1])&&(i>=level[2]))value=level[1];
         a[0]=a[1]=a[2]=value;
         wlut(&i,a);
         }/*next i*/
      }/*end proc*/
   threshold(){
      int x,y,x_grey,shade,s,xpos;
      grey_levels();
      title4();
      cursor1_line(&x_grey);
      xpos=x_grey+40;
      s=200;
      place_line(&xpos,&s);
      for(y=1;y<=199;y++){
         for(x=41;x<=499;x++){
            rpix(&x,&y,&shade);
            if(shade<=x_grey){
               s=0;
               sccol(&s);
               plot(&x,&y);
               }/*end if*/
            }/*next x*/
         }/*next y*/
         tblank();
      }/*end proc*/

   equalise(){
      float *value,sum=0,weight[256];
      int x,y,shade,i,new_shade;
      value=grey_levels();
      for(i=0;i<=255;i++) weight[i]=(sum+=*(value+i));
      for(y=1;y<=199;y++){
         for(x=41;x<=499;x++){
            rpix(&x,&y,&shade);
            if(weight[shade]>=1) new_shade=220;/*220 max grey shade*/
              else new_shade=220*weight[shade];/*254&255 used for area measure*/
            sccol(&new_shade);
            if(shade!=0)plot(&x,&y);
            }/*next x*/
         }/*next y*/
         tblank();
      }/*end proc*/

   cal_area(){
      int n,s,x,y,*b;
      int g_level,element,plotx,ploty;
      long int circle_r[5],x_dist,y_dist;
      double r,pixel_area,area,area_sum=0.0;
   static double total_area[5]={0.785,3.142,7.06,12.56,19.63};
     /*  area in mm for radius=  0.5,  1.0, 1.5,  2.0,  2.5*/
        /*circle_r= 0.5, 1, 1.5, 2, 2.5mm*/
```

```
  /*diameter=  1,  2,  3,  4,  5 mm*/
  for(x=0;x<5;x++){circle_r[x]=0;
                   for(y=0;y<14;y++)grey[x][y]=0;}
  pixel_area=dist_per_pixel*dist_per_pixel*2;
cls();
printf("\ndist_per_pixel=%f",dist_per_pixel);
printf("\n                    ISOLATE SCAR");
printf("\n                    -------------");
printf("\n\n1..mark boundry with cursor");
printf("\n2..place cursor inside area to be blanked, then EXIT cursor");
b=boundry();
printf("\n\nIs the printer switched on?. If not, please switch it on,");
printf("\n%c AND are the boundries correct?",7);
printf("\npress any key to continue, or 'n' to end");
pause=getch();
if(pause=='n'){original();return(-1);}
if(b!=NULL){
lprintf("%c&l32259.1057J",27);
lprintf("IN;SP1;SC0,1000,0,720;DT.;");/*INITIALISE PLOTTER*/
  place_markers(&n);
  s=255;spcol(&s);
  s=254;sccol(&s);
  bfill();
  s=0;sccol(&s);
  printf("\n.");
  for(x=41;x<=499;x++){
     for(y=1;y<=199;y++){
        rpix(&x,&y,&s);
        if(s>=254)plot(&x,&y);
          else{
                   if((s-background)>0){
                      x_dist=abs(x-x_centre);
                      y_dist=2*abs(y-y_centre);
                      r=x_dist*x_dist+y_dist*y_dist;
                      r=sqrt(r);
                      r=r*dist_per_pixel;
                      element=floor(r*2);
                   /*          g_level=(s-background)/10;   */
                      g_level=s/10;
                      if(element>4)element=4;
                      circle_r[element]++;
                      grey[element][g_level]++;
                      plotx=((x-x_centre)*dist_per_pixel*100)+640;
                      ploty=(-2*(y-y_centre)*dist_per_pixel*100)+360;
                      lprintf("PA%d,%d;PD;PU;",plotx,ploty);
                      if(g_level>4)lprintf("PA%d,%d;PD;PU;",plotx+1,ploty+1
                      if(g_level>5)lprintf("PA%d,%d;PD;PU;",plotx-1,ploty-1
                      if(g_level>6)lprintf("PA%d,%d;PD;PU;PD;PU;",plotx,plo
                      if(g_level>7)lprintf("PA%d,%d;PD;PU;PD;PU;",plotx+1,p
                      if(g_level>8)lprintf("PA%d,%d;PD;PU;PD;PU;PD;PU;",plo
                      if(g_level>9)lprintf("PA%d,%d;PD;PU;PD;PU;PD;PU;",plo
                      }
                }/*end if-else*/
        }/*next y*/
     }/*next x*/
  for(x=0;x<5;x++){area_sum=area_sum+circle_r[x];
                   area=area_sum*pixel_area;
                   fraction[x]=((double)area/total_area[x])*100.0;}
  }/*end if*/
  dataplot();
  tblank();
  if(b==NULL) return(-1); else return(0);
/*****************************************************************************/
/*                              Variables
```

```
    global double  dist_per_pixel=side length of pixel in mm. This is assigned
                                  in grab_image in module s_procs.c.
                                  The value assigned is hard coded in the
                                  program from a calibration of the slit lamp.
    global int x_centre,y_centre=pupil centre coordinates in pixels, assigned
                                 in grab image by pupil().
    double r=distance in mm to each pixel from centre.
    int element=array element for pixel count.

    arrays for count are:
    long_int circle_r[5]=the number of pixels within each diameter band of
                         1,2,3,4,5 mm
    long int grey[5][14]=the number of pixels of different grey levels
                         (14 levels) in each diameter band.

    double area_sum=total number of pixels of scar from centre.
    double area=total area of scar in mm from centre.
    double fraction= total area of scar as % of total area of pupil
                     at each different radius band.
    double total_area=area in mm for pupil of each radius(hard coded)
    double length[5]=radius in mm for each step
    int plotx,ploty = coordinates of pixel to plot on
                      page 1 of plotter output - see s_procs          */
/*****************************************************************************/

   }/*end proc*/
profile(){
   int x,xpos,ypos,s,i=0,b1,b2,b3,b4;
   float value[458];
   b1=b2=b3=b4=1;
   title3();
   cursor_block(&xpos,&ypos,&b1,&b2,&b3,&b4);
   for(x=Xgmin+1;x<=Xgmax-1;x++){
      rpix(&x,&ypos,&s);
      *(value+i)=(float)s;
      s=255;sccol(&s);plot(&x,&ypos);/*draw line across profile*/
      i++;
      }/*next x*/
   graph(value);
   do{
      buttons(&b1,&b2,&b3,&b4);
      }while(b1==1 && b2==1 && b3==1 && b4==1);
   i=0;
   for(x=Xgmin+1;x<=Xgmax-1;x++){  /*restore profile*/
      s=(int)*(value+i);
      sccol(&s);
      plot(&x,&ypos);
      i++;
      }/*next x*/
   tblank();
   }/*end proc*/
cursor_line(x)int *x;{
   int old_cursor,new_cursor,under_cursor[65],yd;
   int shade,b1,b2,b3,b4;
   old_cursor=-1;
   shade=200;
   do{
     if (old_cursor>=0)line_replace(&old_cursor,under_cursor);
     joy_pos(&new_cursor,&yd);
     if(new_cursor>260)new_cursor=260;/*220 max grey level allowed*/
     line_save(&new_cursor,under_cursor);
     old_cursor=new_cursor;
     place_line(&new_cursor,&shade);
     buttons(&b1,&b2,&b3,&b4);
```

```
      }while(b1==1 && b2==1 && b3==1 && b4==1);
    line_replace(&old_cursor,under_cursor);
    *x=new_cursor-40;
    }/*end proc*/
 cursor1_line(x)int *x;{
    int old_cursor,new_cursor,under_cursor[65],yd;
    int shade,b1,b2,b3,b4,a[3],i,value;
    old_cursor=-1;
    shade=200;
    do{
      do{
       if(old_cursor>=0)line_replace(&old_cursor,under_cursor);
       joy_pos(&new_cursor,&yd);
       if(new_cursor>260)new_cursor=260;/*220 max grey level allowed*/
       line_save(&new_cursor,under_cursor);
       old_cursor=new_cursor;
       place_line(&new_cursor,&shade);
       buttons(&b1,&b2,&b3,&b4);
       }while(b1==1 && b2==1 && b3==1 && b4==1);
       for(i=0;i<=220;i++){
          if(i<(new_cursor-40))value=0;
            else value=i;
          a[0]=a[1]=a[2]=value;
          wlut(&i,a);
          }/*next i*/
      }while(b1==1);
    lutinitg();
    line_replace(&old_cursor,under_cursor);
    *x=new_cursor-40;
    }/*end proc*/
line_replace(old_cursor,under_cursor)int *old_cursor,under_cursor[];{
     int y,xpos,ypos,shade;
     xpos=*old_cursor;
     for(y=0;y<=64;y++){
        shade=under_cursor[y];
        sccol(&shade);
        ypos=y+205;
        plot(&xpos,&ypos);
        }/*next y*/
     }/*end proc*/
line_save(new_cursor,under_cursor)int *new_cursor,under_cursor[];{
     int y,xpos,ypos,shade;
     xpos=*new_cursor;
     for(y=0;y<=64;y++){
        ypos=y+205;
        rpix(&xpos,&ypos,&shade);
        under_cursor[y]=shade;
        }/*next y*/
     }/*end proc*/
place_line(new_cursor,shade)int *new_cursor,*shade;{
     int y,ypos,xpos;
     sccol(shade);
     xpos=*new_cursor;
     for(y=0;y<=64;y++){
        ypos=y+205;
        plot(&xpos,&ypos);
        }/*next y*/
     }/*end proc*/
float *grey_levels(){
   int grey,x,y,black;
   float value[458];
   for(x=0;x<=457;x++)value[x]=0.0;
   blank_graph();
      for(x=41;x<=499;x+=2){ /*skip every second line and column to speed proc
```

```
          for(y=1;y<=199;y+=2){
              rpix(&x,&y,&grey);
              *(value+grey)=*(value+grey)+1;
              }/*next y*/
          }/*next x*/
      black=value[0];value[0]=1;
      for(x=0;x<=457;x++)value[x]=value[x]/23000;
      graph(value);
      tick();
      value[0]=(float)black/23000;
      return(value);
      }/*end proc*/
  block_grey(){
      int grey,x,y,black,i,group,start;
      float value[458],b_value[26];
      for(x=0;x<=457;x++)value[x]=0.0;
      for(x=0;x<26;x++)b_value[x]=0.0;
      blank_graph();
         for(x=41;x<=499;x++){
           for(y=1;y<=199;y++){
              rpix(&x,&y,&grey);
              if(grey==0)group=0;
                 else group=grey/10;
              *(b_value+group)=*(b_value+group)+1;
              }/*next y*/
           }/*next x*/
           for(start=0;start<26;start++){
              group=start*10;
              for(i=group;i<group+10;i++) *(value+i)=b_value[start];
            }
      black=value[0];value[0]=1;
      for(x=0;x<=457;x++)value[x]=value[x]/23000;
      graph(value);
      tick();
      value[0]=(float)black/23000;
      }/*end proc*/
  tick(){
      int x,y,s;
      x=256+Xgmin;
      y=Ygmax;
      moveto(&x,&y);
      s=255;
      sccol(&s);
      y=Ygmax-3;
      lineto(&x,&y);
      }/*end proc*/
  test_proc(value)float value[];{
      int x;
      cls();
      for(x=0;x<=255;x++)
         printf(" %d=%f ",x,value[x]);
      pause=getch();
      }/*end proc*/
  graph(value)float value[];{
      int xpos,ypos,shade=255,i;
      float max_value=0.0;
      /*calculate max value*/
      for(i=0;i<=457;i++) if(*(value+i)>max_value)max_value=*(value+i);
      /*plot histogram*/
      blank_graph();
      sccol(&shade);
      for(xpos=Xgmin+1;xpos<=Xgmax-1;xpos++){
         ypos=Ygmax;
         moveto(&xpos,&ypos);
```

```
      ypos=Ygmax-(value[xpos-(Xgmin+1)]*(Ygmax-Ygmin)/max_value);
      lineto(&xpos,&ypos);
      }/*next xpos*/
   }/*end proc*/
blank_graph(){
   int x,y,s,xl,yl;
   s=0;
   sccol(&s);
   xl=(Xgmax)-(Xgmin+1);
   yl=Ygmax-Ygmin;
   x=Xgmin+1;
   y=Ygmin;
   moveto(&x,&y);
   rfill(&xl,&yl);
   }/*end proc*/
cursor_test(){
   int points,*b,x=0;
   cls();
   printf("\n\nTHE FOLLOWING CURSOR FUNCTIONS ARE TESTED:");
   printf("\n1..place boundry");
   printf("\n2..place markers");
   printf("\n3..histogram line cursor");
   pinit();
   lutinitg();
   fgrab();
   blank();
   boxes();
   b=boundry();
   if(b!=NULL)place_markers(&points);
   title4();
   cursor_line(x);
   }/*end proc*/
tblank(){
   int x=506,y=1,s=0,xl=194,yl=268;
   sccol(&s);
   moveto(&x,&y);
   rfill(&xl,&yl);
   }/*end proc*/
cursor_block(x,y,b1,b2,b3,b4)int *x,*y,*b1,*b2,*b3,*b4;{
   int old_cursor[2],new_cursor[2],under_cursor[3][2];
   int shade;
   old_cursor[0]=-1;
   shade=255;
   do{
     if(old_cursor[0]>=0) replace_block(old_cursor,under_cursor);
     joy_pos(&new_cursor[0],&new_cursor[1]);
     save_block(new_cursor,under_cursor);
     old_cursor[0]=new_cursor[0];
     old_cursor[1]=new_cursor[1];
     place_cursor(new_cursor,&shade);
     buttons(b1,b2,b3,b4);
     }while(*b4==1 && *b3==1 && *b2==1 && *b1==1);
   replace_block(old_cursor,under_cursor);
   *x=new_cursor[0];
   *y=new_cursor[1];
   }/*end proc*/
curs_point(x,y,b1,b2,b3,b4)int *x,*y,*b1,*b2,*b3,*b4;{
   int old_cursor[2],new_cursor[2],under_cursor[3][2];
   int shade,threshold;
   old_cursor[0]=-1;
   shade=255;
   do{
     if(old_cursor[0]>=0) replace_block(old_cursor,under_cursor);
     joy_pos(&new_cursor[0],&new_cursor[1]);
```

```
      rpix(&new_cursor[0],&new_cursor[1],&threshold);
      save_block(new_cursor,under_cursor);
      old_cursor[0]=new_cursor[0];
      old_cursor[1]=new_cursor[1];
      place_cursor(new_cursor,&shade);
      buttons(b1,b2,b3,b4);
      cls();
      printf("\nchoose level=%d",threshold);
      }while(*b4==1 && *b3==1 && *b2==1 && *b1==1);
   replace_block(old_cursor,under_cursor);
   *x=new_cursor[0];
   *y=new_cursor[1];
   }/*end proc*/
place_markers(points)int *points;{
   long int count;
   int pos[1],n=0,b1,b2,b3,b4;
   title2();
   do{
     cursor_block(pos,pos+1,&b1,&b2,&b3,&b4);
     for(count=0;count<=20000;count++);/*debounce*/
     }while(b1==1 && b2==1 && b3==1 && b4==1);
     *points=n;
   }/*end proc*/
int *boundry(){
     int xpos,ypos,b1=1,b2=1,b3=1,b4=1,shade;
     int pos[2],*xval,*yval,x,n,*b;
     long int count;
     unsigned bytes;
     char *malloc(),*realloc();
     do{
       n=0;
       xval=(int*)malloc(sizeof(int));
       yval=(int*)malloc(sizeof(int));
       title1();
       do{
          cursor_block(&xpos,&ypos,&b1,&b2,&b3,&b4);
          if(b1==1 && b4==1){
            if(b2==1){
              shade=255;
              pos[0]=xpos;
              pos[1]=ypos;
              place_cursor(pos,&shade);
              n++;
              bytes=n*sizeof(int);
              xval=(int*)realloc((char*)xval,bytes);
              yval=(int*)realloc((char*)yval,bytes);
              if(xval==NULL || yval==NULL){
                cls();
                printf("Error in memory allocation\n");
                printf("press any key to return to main menu");
                b1=0;
                pause=getch();
                b=(int *)NULL;
                return(b);
                }/*end if*/
              xval[n-1]=xpos;
              yval[n-1]=ypos;}
             else{
              shade=0;
              pos[0]=xval[n-1];
              pos[1]=yval[n-1];
              place_cursor(pos,&shade);
              n--;
              if(n<0)n=0;
```

```
                bytes=n*sizeof(int);
                xval=(int*)realloc((char*)xval,bytes);
                yval=(int*)realloc((char*)yval,bytes);
                if(xval==NULL || yval==NULL){
                  cls();
                  printf("Error in memory allocation or last point was removed\n
                  printf("press any key to return to main menu");
                  b1=0;
                  pause=getch();
                  b=(int *)NULL;
                  return(b);
                }/*end if*/
              }/*end if-else*/
            }/*end if*/
           for(count=0;count<=20000;count++);/*debounce*/
           }while(b1==1 && b4==1);
       shade=255;sccol(&shade);
       moveto(xval,yval);
       for(x=1;x<n;x++)lineto(xval+x,yval+x);
       lineto(xval,yval);
       free((char*)xval);free((char*)yval);
       }while(b1==1);
       for(count=0;count<=20000;count++);/*debounce*/
    }/*end proc*/
save_block(new_cursor,under_cursor)int new_cursor[],under_cursor[][2];{
   int x,y,xpos,ypos,shade,xcursor,ycursor;
   xcursor=new_cursor[0];
   ycursor=new_cursor[1];
   for(y=0;y<=1;y++){
      for(x=0;x<=2;x++){
         xpos=xcursor+x;
         ypos=ycursor+y;
         rpix(&xpos,&ypos,&shade);
         under_cursor[x][y]=shade;
         }/*next x*/
      }/*next y*/
   }/*end proc*/
replace_block(old_cursor,under_cursor)
   int old_cursor[],under_cursor[][2];{
   int x,y,xpos,ypos,shade,xcursor,ycursor;
   xcursor=old_cursor[0];
   ycursor=old_cursor[1];
   for(y=0;y<=1;y++){
      for(x=0;x<=2;x++){
         xpos=xcursor+x;
         ypos=ycursor+y;
         shade=under_cursor[x][y];
         sccol(&shade);
         plot(&xpos,&ypos);
         }/*next x*/
      }/*next y*/
   }/*end proc*/
place_cursor(new_cursor,shade)int new_cursor[],*shade;{
   int x,y,xpos,ypos,xcursor,ycursor;
   xcursor=new_cursor[0];
   ycursor=new_cursor[1];
   sccol(shade);
   for(y=0;y<=1;y++){
     for(x=0;x<=2;x++){
       xpos=xcursor+x;
       ypos=ycursor+y;
       plot(&xpos,&ypos);
       }/*next x*/
     }/*next y*/
```

```
    }/*end proc*/

 joy_pos(xcpos,ycpos)int *xcpos,*ycpos;{
      /*xcpos,ycpos are actual cursor position in viewport*/
     int joyx,joyy,xcvalue,ycvalue;

     /* int adapt,device,chanlo,ctrl,stat;
      adapt=0;
      device=9;
      chanlo=0;
      ctrl=0;
      ains(adapt,device,chanlo,ctrl,&joyx,&stat);
      chanlo=1;
      ains(adapt,device,chanlo,ctrl,&joyy,&stat);*/

      mm_get(&joyx,&joyy);

      xcvalue=joyx-Cxmin;
      ycvalue=joyy-Cymin;
      *xcpos=xcvalue*fjx+Xmin;
      *ycpos=ycvalue*fjy+Ymin;
      if(*xcpos<Xmin) *xcpos=Xmin;
      if(*xcpos>(Xmax-3)) *xcpos=(Xmax-3);
      if(*ycpos<Ymin) *ycpos=Ymin;
      if(*ycpos>(Ymax-2)) *ycpos=(Ymax-2);
      }/*end proc*/
buttons(b1,b2,b3,b4)int *b1,*b2,*b3,*b4;{
long int count;
int x=0;
*b1=*b2=*b3=*b4=1;
x=mm_button();
switch(x){
      case 1:*b1=0;
             break;
      case 2:*b2=0;
             break;
      case 3:*b3=0;
             for(count=0;count<=40000;count++);
             if(mm_button()==3){*b4=0;
                                while(mm_button()==3);}
             break;
      }/*end case*/

/*      int adapt,device,bit1,bit2,bit3,bit4,stat;
      bit1=12;
      bit2=13;
      bit3=14;
      bit4=15;
      adapt=0;
      device=8;
      bitins(adapt,device,bit1,b1,&stat);
      bitins(adapt,device,bit2,b2,&stat);
      bitins(adapt,device,bit3,b3,&stat);
      bitins(adapt,device,bit4,b4,&stat);*/
      }/*end proc*/
```

Appendix D.

Measurement of corneal topography in keratoconus

D. A. de Cunha and E. G. Woodward

Applied Vision Research Centre, Department of Optometry and Visual Science, City University, Dame Alice Owen Building, 311–321 Goswell Road, London EC1V 7DD, UK

(Received 24th March 1993)

A new method is described for calculating the topography and curvature of an irregularly shaped cornea. Vertical planes of light are projected onto the cornea and points in x, y, z space are calculated from the light images on the corneal surface. A matrix of points is produced on the cornea and a mathematical surface is fitted to them. Using differential geometry theory, curvature values are calculated to find the corneal apex position, apex curvature and curvature change away from the apex. Typical results, in the form of a contour map, are shown for normal corneas and corneas with mild to severe keratoconus. The method gives an accurate quantitative measurement of keratoconic and other irregular corneas.

Keratoconus is an inherited eye disease in which localized thinning of the cornea causes it to distort from its normal regular shape. Initially this produces high astigmatism and later corneal protrusion and scarring. The management is usually by rigid contact lens wear but about 10–15% of eyes become so distorted that a corneal transplant is required. The progression of the disease is monitored by measuring the changing front curvature of the eye.

Different methods of measuring the topography and curvature of the anterior corneal surface have been developed over the last 50 years and these have been reviewed in the literature[1,2]. Although each method has its own unique features, two fundamentally different approaches have emerged in modelling the corneal surface.

The first approach uses the cornea as a reflecting convex mirror and measures the height of concentric ring images formed from the corneal surface. Standard paraxial ray equations are used to calculate the curvature at different points and build up a profile. Photokeratoscopes based on this principle[3,4] have been widely used over the last 20 years and have lately been superseded by sophisticated instruments[5,6] that photograph and analyse the ring images using a microcomputer system.

The second approach to mapping the cornea has been to measure actual points on the surface of the cornea and build a three-dimensional map. Methods using this approach looking directly at the corneal surface have included profile measurements[7], moiré fringe analysis[8] and stereophotogrammetry[9]. Within the last few years, the stereophotogrammetry method has been developed into rasterstereography[10,11] using a modified slit lamp and computer image grabbing system.

Keratoconus has been extensively photographed using photokeratoscopes but pictures[12,13] typically show highly distorted images produced from non-spherical surfaces. Many of the rings are out of focus and there is additional uncertainty in the alignment of the cornea with the instrument axis. The resulting pictures are difficult to analyse and curvature values are probably invalid in these cases because they are derived from ray equations that assume spherical surfaces. Stereographic techniques do not assume spherical surfaces, therefore they do not suffer from the problems outlined above when viewing distorted corneas. Previous systems have calculated curvature by fitting the best arc to the corneal profile along any meridian. The further the shape departs from sphericity, the more difficult and unreliable this method becomes.

We have developed a system of measuring the corneal topography and curvature for both normal and highly distorted eyes. This has lead to accurate and quantitative measurement of keratoconic and other irregular corneas. To overcome problems suffered by other instruments we needed to develop a system which could,

1. measure surface without any assumption about underlying shape,
2. reconstruct surface shape independent of alignment of eye,
3. extract parameters at any point on the surface and determine apex position, apex curvature and rate of flattening.

Method

Measurement of surface without previous assumption of surface shape

A system was set up to project vertical planes of light onto the eye and a picture of the planes on the cornea was taken by a computer image grabber. The direction of each plane is known and used in place of a second

0275-5408/93/040377-06

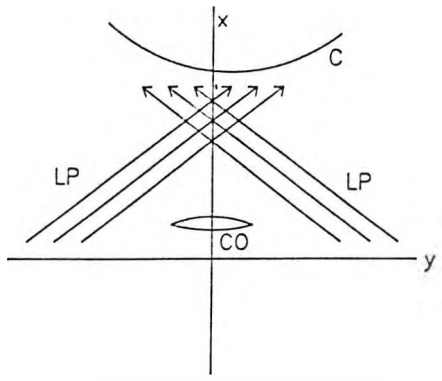


Figure 1 Projector and camera configuration with arbitrary x, y axes aligned with camera axes; CO is the camera objective and LP are the light planes projected onto the cornea C

camera in a stereographic system for calculating depth information[14].

The projection system was arranged to give maximum information over the central 4 mm of the cornea, which is the area of interest for vision. The configuration is shown in *Figure 1*.

Sodium fluorescein, 2%, was placed on the cornea to increase the brightness of the light plane images, which were very faint without fluorescein; also a histogram equalization image enhancement was applied to the camera input. A discrete point P is calculated on the cornea in x, y, z space by finding the intersection of a given light plane with a ray from an image point P of the light plane on the cornea as shown in *Figure 2*.

The general equation of the plane is

$$\hat{n}_p \bullet (\hat{r}_p - \hat{r}_{p0}) = 0 \tag{1}$$

where

$\hat{n}_p$ is a unit vector normal to the plane
$\hat{r}_{p0}$ is a vector to a point on the plane
$\hat{r}_p$ is a general vector in x, y, z space

The unit vector normal to the plane is found by measuring the angle ψ between the plane and the camera axis.

$$\hat{n}_p = \sin\psi\hat{x} - \cos\psi\hat{y} \tag{2}$$

If the intersection point of the plane with the x axis is at distance L, then $\hat{r}_{p0}$ can be taken along the x axis to this point.

$$\hat{r}_{p0} = L\hat{x} \tag{3}$$

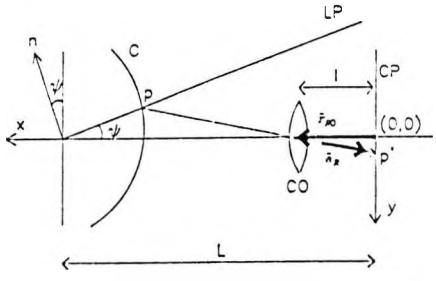


Figure 2 Intersection of ray and plane LP gives point P in x, y, z space on the corneal surface C. The camera objective is CO and CP is the camera image plane. The normal to the plane is given by n

When ψ and L are measured for each plane, the corresponding plane equation can be found from Equation (1)

$$(\sin\psi\hat{x} - \cos\psi\hat{y}) \cdot (x\hat{x} + y\hat{y} + z\hat{z} - L\hat{x}) = 0 \tag{4}$$

$$(\sin\psi\hat{x} - \cos\psi\hat{y}) \cdot ((x - L)\hat{x} + y\hat{y} + z\hat{z}) = 0 \tag{5}$$

$$\sin\psi(x - L) - y\cos\psi = 0 \tag{6}$$

Equation (6) is the equation for each plane in x, y, z space. The general equation of a light ray from P to P′ is given by

$$\hat{r}_R = \hat{r}_{R0} + \hat{a}_R t \tag{7}$$

where

$\hat{r}_{R0}$ is a vector to the ray

$\hat{a}_R$ is a unit vector along the ray

$\hat{r}_R$ is a general vector in x, y, z space

For each image point P′$(0, y'_p, z'_p)$, the ray equation (7) can be solved. If the distance from the camera focal plane CP to the centre of the camera objective CO = l, then $\hat{r}_{R0}$ can be taken along the x axis to CO.

$$\hat{r}_R = l\hat{x} + t(\hat{x}l - y'_p\hat{y} - z'_p\hat{z}) \tag{8}$$

$$x\hat{x} + y\hat{y} + z\hat{z} = (tl + l)\hat{x} - ty'_p\hat{y} - tz'_p\hat{z} \tag{9}$$

Equating components

$$x = l(t + 1) \tag{10}$$

$$y = -y'_p t \tag{11}$$

$$z = -z'_p t \tag{12}$$

t is a distance parameter along the light ray, and its value at the ray-plane intersection is found by inserting Equations (10), (11) and (12) into Equation (6).

$$\sin\psi(l(t + 1) - L) + y'_p t\cos\psi = 0 \tag{13}$$

$$t(l\sin\psi + y'_p\cos\psi) = (L - l)\sin\psi \tag{14}$$

$$t = \frac{(L - l)\sin\psi}{l\sin\psi + y'_p\cos\psi} \tag{15}$$

The x, y, z point P is found by substituting t in Equation (15) back into Equations (10), (11) and (12). This gives a unique point on the corneal surface in three-dimensional space dependent on y'_p and z'_p (for a given plane).

These equations are solved at chosen points on each plane; there are 26 light planes for which the data is collected producing a matrix of discrete points on the corneal surface.

Reconstruction of surface independent of x, y, z co-ordinate system

After interpolation between the discrete points to produce a set of points on a uniform matrix, a surface equation can be developed using u, v co-ordinate axes defined in the surface as shown in *Figure 3*.

The form of the surface equation and meaning of the u, v parameters can be illustrated by discussing the

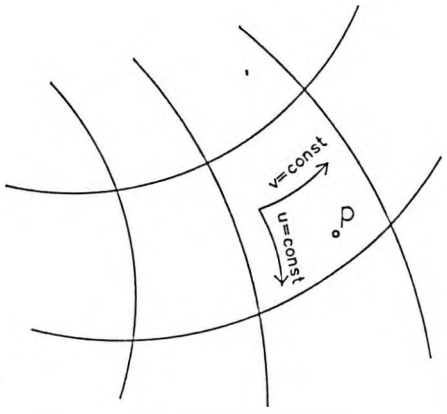


Figure 3 Co-ordinate axes u, v on the surface which uniquely define any point P on the surface

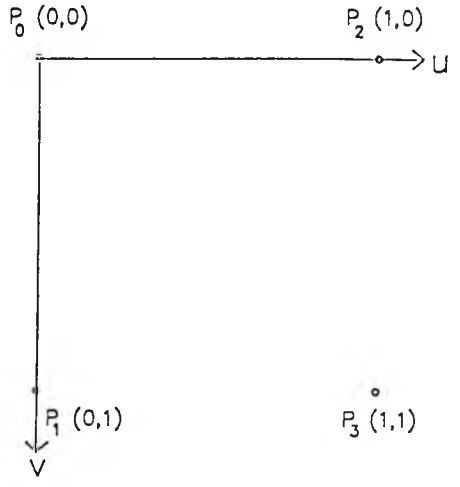


Figure 4 Four points on u, v parameter space

construction of a simple bilinear surface between four points[15]. This is a surface in which the boundaries between points are linear and the interpolation between boundaries are linear. Using four points in x, y, z space, a u, v co-ordinate system is constructed through the points with P_0, P_1, P_2, P_3 at u, v co-ordinates (0,0), (0,1), (1,0) and (1,1) respectively as shown in *Figure 4*.

Linear interpolation between P_0 and P_1 gives (in vector notation)

$$\vec{P}_0 + v(\vec{P}_1 - \vec{P}_0) = \vec{Q}_{0v} \tag{16}$$

Equation (16) represents three individual equations for the x, y, z components. Parameter v then represents the fractional distance along the line P_0 to P_1. e.g. $v = 0.5$ lies halfway between P_0 and P_1. Similar interpolation between P_2 and P_3 gives

$$\vec{Q}_{1v} = \vec{P}_2 + v(\vec{P}_3 - \vec{P}_2) \tag{17}$$

Equations (16) and (17) represent linear boundaries at constant u. We can now interpolate between the boundaries for any u at a given v as illustrated in *Figure 5*.

For a given v, a linear interpolation between the boundaries gives

$$\vec{Q}(u, v) = \vec{Q}_{0v} + u(\vec{Q}_{1v} - \vec{Q}_{0v}) \tag{18}$$

$$\vec{Q}(u, v) = (1 - u)\vec{Q}_{0v} + u\vec{Q}_{1v} \tag{19}$$

Substituting $\vec{Q}_{0v}$ and $\vec{Q}_{1v}$ from Equations (16) and (17) gives

$$\vec{Q}(u, v) = (1 - u)[\vec{P}_0 + v(\vec{P}_1 - \vec{P}_0)] + u[\vec{P}_2 + v(\vec{P}_3 - \vec{P}_2)] \tag{20}$$

expanding Equation (20) we obtain

$$\vec{Q}(u, v) = (1 - u)(1 - v)\vec{P}_0 + (1 - u)v\vec{P}_1 + u(1 - v)\vec{P}_2 + uv\vec{P}_3 \tag{21}$$

and in matrix notation Equation (21) gives

$$\vec{Q}(u, v) = \begin{bmatrix} (1 - v)(1 - u)\vec{P}_0 & v(1 - u)\vec{P}_1 \\ (1 - v)u\vec{P}_2 & uv\vec{P}_3 \end{bmatrix} \tag{22}$$

when the coefficients are separated from the point matrix, the surface equation takes the form

$$\vec{Q}(u, v) = [(1 - u)u]\begin{bmatrix} \vec{P}_0 & \vec{P}_1 \\ \vec{P}_2 & \vec{P}_3 \end{bmatrix}\begin{bmatrix} 1 - v \\ v \end{bmatrix} \tag{23}$$

The coefficients $(1 - u), u, (1 - v), v$ can be thought of as weighting functions for different points.

This method for development of a bilinear surface can be applied to a polynomial surface if the linear interpolation function is replaced by a polynomial function. A suitable form of surface equation, using a spline interpolation between 16 points, has been used. The equation for each component x, y, z of the surface between four points is[16]

$$\vec{Q}(u, v) = \left[\frac{1}{36}\right]\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}\begin{bmatrix} u^3 \\ u^2 \\ u \\ 1 \end{bmatrix}^T \times \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} \times \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix} \tag{24}$$

where

$$\begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$$

is a matrix of x, y or z values.

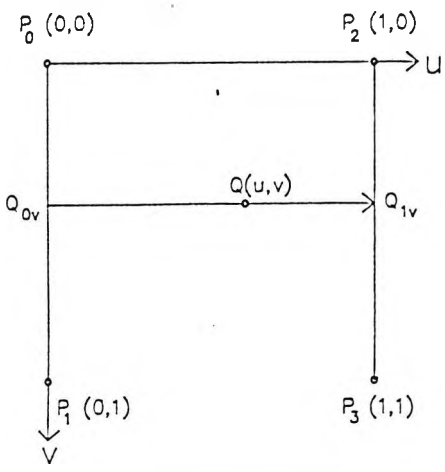


Figure 5 Co-ordinate axes u, v on the surface which uniquely define any point P on the surface

Using an iteration process, the point matrix is adjusted until the surface passes through all the original points within some acceptable error and degree of smoothing.

Extracting values from the surface

The most common parameter for describing the corneal surface is radius of curvature; radius of curvature can be calculated at any point on the surface using differential geometry theory[17].

At any point on the surface, two tangent vectors to the surface in the direction of the parameter axes are found by taking partial derivatives of the surface Equation (24). The vector cross product of these tangent vectors gives a vector normal to the surface at that point, as shown in *Figure 6*.

A plane containing the surface normal is constructed which intersects the surface as shown in *Figure 7*.

For a given normal, an infinite number of planes can be constructed for different directions dv/du on the surface. For any given direction dv/du the curvature of the line of intersection between the plane and surface can be calculated[18]. Using the surface Equation (24), the curvature is given by

$$k_n = \frac{\bar{Q}_{uu} \cdot \bar{N} + 2\bar{Q}_{uv} \cdot \bar{N}\left(\frac{dv}{du}\right) + \bar{Q}_{vv} \cdot \bar{N}\left(\frac{dv}{du}\right)^2}{\bar{Q}_u \cdot \bar{Q}_u + 2\bar{Q}_u \cdot \bar{Q}_v\left(\frac{dv}{du}\right) + \bar{Q}_v \cdot \bar{Q}_v\left(\frac{dv}{du}\right)^2} \qquad (25)$$

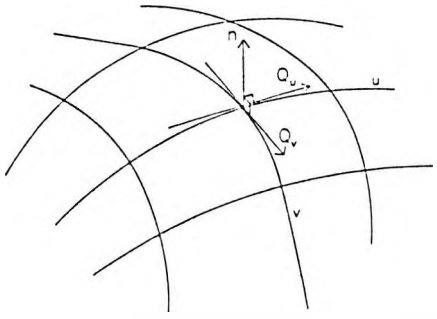


Figure 6 Surface tangent vectors Q_u and Q_v lie along co-ordinate axes, with surface normal vector n orthogonal to both Q_u and Q_v

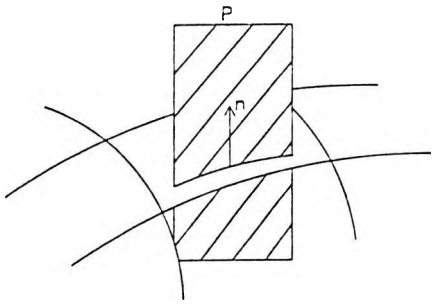


Figure 7 Plane P through surface containing normal n

Results

The resolution of the system depends on three factors,

1. the angle of projection of the light planes,
2. the magnification of the camera system and
3. the pixel resolution of the image grabbing system.

For the results shown below the projection angle was 60°, the camera magnification was ×20 and the pixel resolution was 768 × 512, giving a depth resolution of 10 μm.

The appearance of the light plane image on the eye is shown in *Figure 8*. Points on the cornea are extracted from these images and used to fit a surface model to the corneal shape.

Results are displayed as a contour map, with contours at 50 μm intervals and showing apex position, apex radius of curvature in mm and average rate of flattening from the apex in mm/mm (*Figure 9*).

Figure 9(a) shows a typical result from a normal cornea with a flatter than average apex radius of curvature of 8.06 mm and a central 3 mm region which is effectively spherical. The A on the map indicates the apex position. By contrast *Figure 9(b)* shows the results from a cornea with keratoconus. The measurements indicate a small apical radius of curvature and high rate of flattening. The contour map appearance gives a good visual impression of the actual corneal shape.

Figures 9(c) and *(d)* show results from the left and right eye of the same patient. The left eye *(Figure 9(c)* had a visual acuity of 6/5 when wearing a contact lens

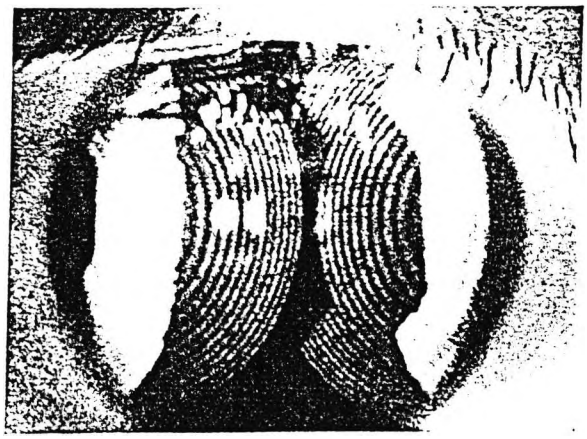


Figure 8 Appearance of normal cornea

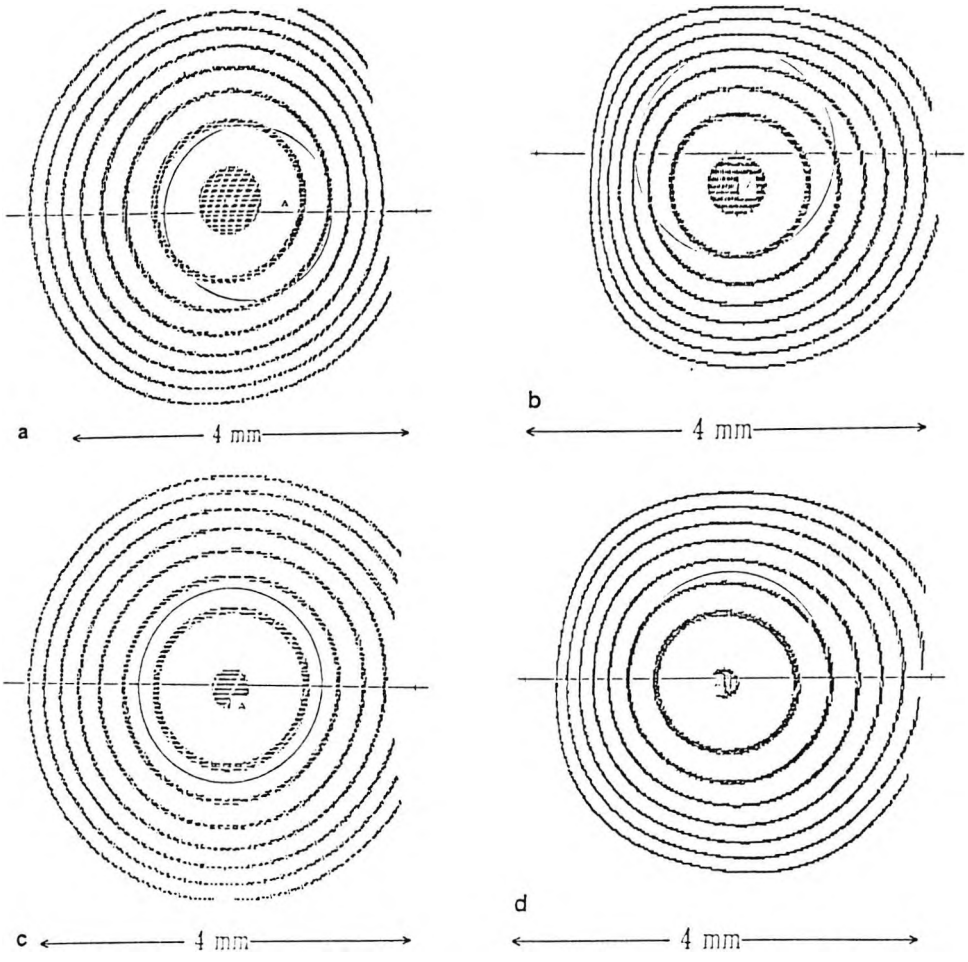


Figure 9 (*a*) Normal cornea, apex radius of curvature = 8.06 mm, average rate of flattening = 0.00 mm/mm. (*b*) Severe keratoconus, apex radius of curvature = 5.13 mm, average rate of flattening = 0.23 mm/mm. (*c*) Apex radius of curvature = 7.41 mm, average rate of flattening = 0.03 mm/mm. (*d*) Apex radius of curvature = 5.50 mm, average rate of flattening = 0.17 mm/mm

of back optic zone radius of 7.80 mm. Keratometry images were clear and single giving readings of 7.95 along 180°C, and 7.67 along 90°. The cornea, however, did have increased visibility of the corneal nerves and Vogt's straie, suggesting that keratoconus was present but had not yet produced any anterior corneal surface changes in topography.

The right eye (*Figure 9(d)*) had paracentral scarring and a visual acuity of 6/12 when wearing contact lens of back optic zone radius of 6.50 mm. Meaningful keratometry measurements could not be obtained owing to mire distortion. A Fleischers ring was present and the visibility of corneal nerves increased.

Discussion

The recent expansion of corneal surgery for the correction of refractive errors has generated a renewed interest in the investigation of corneal topography. New methods have been developed which produce copious computed data. However, the instruments still rely on eliciting a catoptric image from the cornea, measuring the height of concentric ring images and using standard paraxial ray equations to calculate the curvature at different points. The sophistication of the systems lie in the analysis and presentation of data, not in the method of obtaining information. They have not addressed the problem of the poor quality of the image produced from irregular surfaces, nor do they collect images from the centre of the cornea. Furthermore, the display of colour maps and contour graphs may actually mislead the clinician into thinking he has more accurate information than is actually the case. A recent paper[6] has shown that the precision of the values output from the EyeSys Corneal Topography System are beyond the capability of the instrument. The method described in this paper is ideally suited to the quantification of irregular surfaces such as is found in keratoconus. Systems currently commercially available are noticeably deficient in measuring such corneas. This is particularly the case where the cornea is not symmetric along a given meridian which is usually the case in keratoconus, i.e. that typically the superior portion of the cornea in any meridian flattens faster than the inferior portion.

Conclusions

Although contemporary methods of measuring corneal topography can give accurate and easily assimilable results on regular corneas, the new system described will give more accurate and valuable results on irregular

corneas. The monitoring of the progression of keratoconus has always been a problematical exercise, relying on such information as the back surface of the appropriate contact lens. The only alternative which had greater precision was that of serial topographical pachometry. Although this technique is accurate it is highly time consuming and needs an experienced operator. This new method of measuring corneal topography should permit more accurate and easier evaluation of the progression of the disease. It will also allow quantification of such factors as the tilting of a donor graft in a host cornea which is difficult to evaluate by other methods.

Acknowledgements

The authors are grateful to Mr Roger Buckley of Moorfields Eye Hospital for allowing access to his patients and to Mr Ernie Caswell for construction of the projection system.

References

1. Clark, B. A. Less common methods of measuring corneal topography. *Aust. J. Optom.* **56**, 182–192 (1973).
2. Yoon Phin Kon. Some methods of corneal measurement. *Contact Lens Journal* 19(8), Continuing Education No. 1 (1990).
3. Townsley, M. G. New equipment and methods for determining the contour of the human cornea. *Contacto* **11**, 72–81 (1967).
4. Bibby, M. M. Computer-assisted photokeratoscopy and contact lens design. *The Optician* **171**, 2-12 (1976).
5. Gormley, D. J., Gersten, M., Koplin, R. S. and Lubkin, V. Corneal modeling. *Cornea* **7**, 30–35 (1988).
6. McCarey, B. E., Zurawski, C. A. and O'Shea D. S. Practical aspects of a corneal topography system. *The CLAO Journal* **18**, 248–254 (1992).
7. McMonnies, C. W. Corneal curvature from profile measurements. *Aust. J. Optom.* **54**, 153–162 (1971).
8. Kawara, T. Corneal topography using moiré contour fringes. *Appl. Opt.* **18**, 3675–3678 (1979).
9. Bertotto, E. V. The stereophotogrammetric study of the anterior segment of the eye. *Am. J. Ophthal.* **31**, 573–579 (1948).
10. Warnicki, J. W., Rehkopf, P. G., Curtin, D. Y., Burns, S. A., Arffa, R. C. and Stuart J. C. Corneal topography using computer analyzed rasterstereographic images. *Appl. Opt.* **27**, 1135–1140 (1988).
11. Arffa, R. C., Warnicki, J. W. and Rehkopf, P. G. Corneal topography using rasterstereography. *Refractive and Corneal Surgery* **5**, 414–417 (1989).
12. Rowsey J. J., Reynolds A. E. and Brown R. Corneal topography. *Arch. Ophthalmol.* **99**, 1093–1100 (1981).
13. Rabinowitz, Y. S., Garbus, J. and McDonnell, P. J. Computer-assisted corneal topography in family members of patients with keratoconus. *Arch. Ophthalmol.* **108**, 365–371 (1990).
14. Sato, Y., Kitagawa, H. and Fujita, H. Shape measurements of curved objects using multiple slit-ray projections. *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI **4**, 641–646 (1982).
15. Rogers, D. F. and Adams, J. A. *Mathematical Elements for Computer Graphics*, McGraw-Hill, NY, USA (1976).
16. Beach, R. C. *Curves and Surfaces of Computer-aided Design*. Van Nostrand-Reinhold, NY, USA (1990).
17. Stoker, J. J. *Differential Geometry*. Wiley, NY, USA (1969).
18. Struik, D. J. *Lectures on Classical Differential Geometry*. Dover, NY, USA (1950).

METHODS FOR THE MEASUREMENT AND ANALYSIS OF LIGHT SCATTERED IN THE HUMAN EYE

Barbur J L, De Cuhna D, Harlow A J and Woodward E G. Applied Vision Research Centre, Department of Optometry and Visual Science, The City University, Northampton Square, London EC1V OHB.

INTRODUCTION Scattering of light is present in the human eye and ends up illuminating the retina away from the image of a source, although some of the light can be back-scattered through the pupil and this is particularly evident in patients with corneal scars or keratoconus. Scattered light is present in small amounts even in what one normally refers to as pure transparent materials. When the scattering centres are of equal or larger dimension than the wavelength of the incident beam, the angular distribution of the scattered beam tends to follow closely the direction of the incident beam. Scattered light in the eye has the effect of reducing visual acuity and contrast sensitivity and can also affect significantly the accuracy of psychophysical measurements of visual performance such as colour discrimination. When the level of scattered light is large, either as a result of changes in the structure of the dioptrics of the eye or the presence of intense sources of light, this results in significant impairment of vision, sometimes described as visibility glare (Vos, 1984; Vos and Bouman, 1959). Scatter in the human eye and in particular its angular dependence has been the subject of numerous studies (Holladay, 1927; Stiles and Crawford, 1937). In the case of small sources, the angular dependence of scattered light in the eye can be described adequately by a point spread function which decreases with the reciprocal of the square of the visual angle between the source and the point of interest on the retina (Stiles and Crawford, 1937). This relationship shows clearly that little or no Rayleigh scattering takes place in the eye since the very small particles involved in Rayleigh scattering would, in the extreme case, correspond to the ideal forward diffuser, with no angular dependence (Rayleigh, 1912). Measurement of overall scatter level and its angular dependence in the eye can provide valuable information on the number and the size of the particles involved. Monitoring changes in light scatter parameters may provide useful information on the underlying morphological changes in the dioptrics of the eye. Since the scattering of light is not uniform over the pupil and the formation of ocular opacities tends to start in the periphery of the lens, measuring the effect of pupil size on light scatter may provide an additional parameter worth investigating. Extended annular sources of light scatter are often required in order to increase the light flux level entering the eye. This is usually the case when the scatter source is of relatively low luminance and generated on visual display units. The use of extended sources makes it more difficult to extract accurately the angular dependence of light scatter in the eye since the scatter source can no longer be taken to have a single eccentricity. Errors in estimates of light scatter parameters can also be introduced when the light flux level entering the eye does not remain constant for different scatter source eccentricities. **In this paper we describe how the various problems mentioned above can be overcome and the light scattered in the human eye measured using a stable display and appropriate computational methods.**

EXPERIMENTAL METHODS The intraocular light scatter program used in this investigation has been implemented on the P_SCAN 100 pupillometer apparatus (Barbur, 1991) which allows the simultaneous, binocular measurement of pupil size and eye movements. The principle of the method employed is based on the flicker compensation technique described by van den Berg and Spekreijse (van den Berg, 1986; Van den Berg and Spekreijse, 1987). The program makes use of large annuli which vary in *effective eccentricity* (ie., the eccentricity of an equivalent, narrow annulus which would cause the same level of scattered light at the location of the test stimulus). The dimensions of the scattering source are adjusted for each eccentricity of interest so as to maintain a constant light flux level in the plane of the pupil. The basic diagram for the calculations involved is shown in figure 1 (a,b). Sinusoidal modulation of scatter source luminance is achieved at a frequency of 8.6 Hz and this is presented to the subject as a burst of flicker lasting for 1.2 s. The luminance of the central test stimulus is modulated in counterphase with the scattering source at the same temporal frequency. The subject is required to adjust the mean luminance of the test stimulus so as to null out the perceived flicker over the test stimulus using a modified staircase procedure. Six estimates of the equivalent luminance of the test stimulus which is sufficient to balance the retinal illuminance caused by scattered light are obtained and averaged for each scatter source eccentricity. Estimates of the standard errors involved provide appropriate weights for the regression analysis required to compute the light scatter model parameters. ie., n and Θ, in the empirical light scatter equation, $L_s = k.E/\Theta^n$, where k and n are constants and Θ represents the *effective eccentricity* of the annulus. Before the regression analysis can be

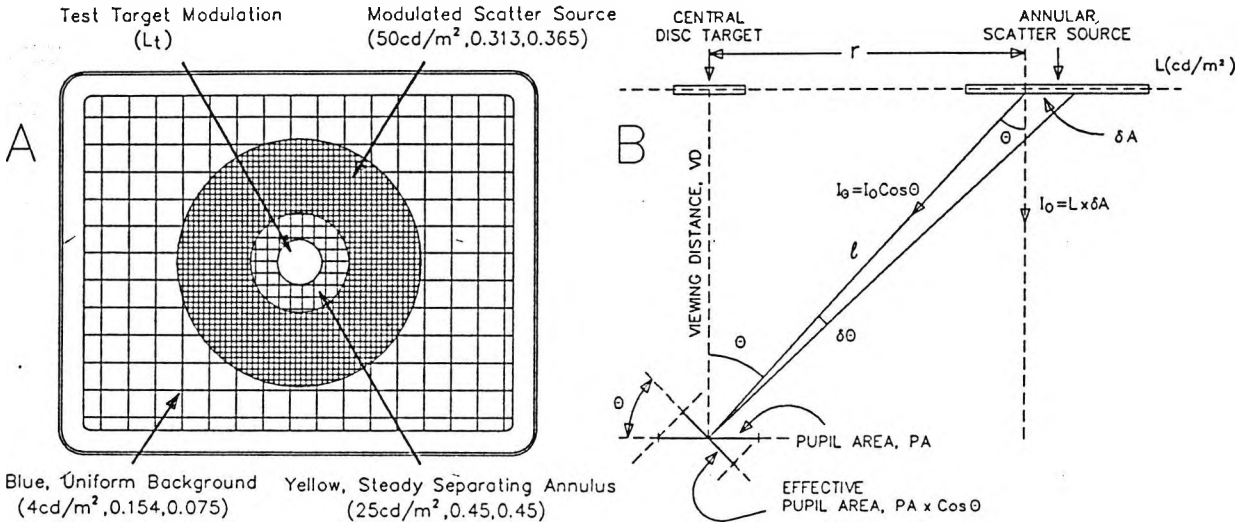


Figure 1. Diagrams showing a schematic representation of the stimulus configuration used for the measurement of scattered light on a visual display unit (A), and the geometry of the display required for the computation of annulus size as a function of eccentricity (B).

The light scatter program was implemented on the P_SCAN 100 system (Barbur, 1991) which provides simultaneous measurements of pupil diameter. The system incorporates facilities for automatic calibration of phosphor luminances over the centre of the display and makes use of an LMT 1002 luminance meter. The various sizes and the luminance and chromaticity parameters employed for the uniform background, the scatter annulus and the separating annulus were selected on the basis of preliminary experiments. The measurement technique is based on a flicker nulling method similar to that described by van den Berg (1986). The luminance of the scattering source was modulated sinusoidally at a frequency of 8.6 Hz. The luminance of the centre test stimulus was also modulated sinusoidally in counterphase with the scattering source and its mean luminance adjusted to obtain a null flicker point using a modified staircase procedure. Section (B) shows the basis for the computation of annulus size so as to ensure that the illuminance in the pupil plane was independent of annulus eccentricity.

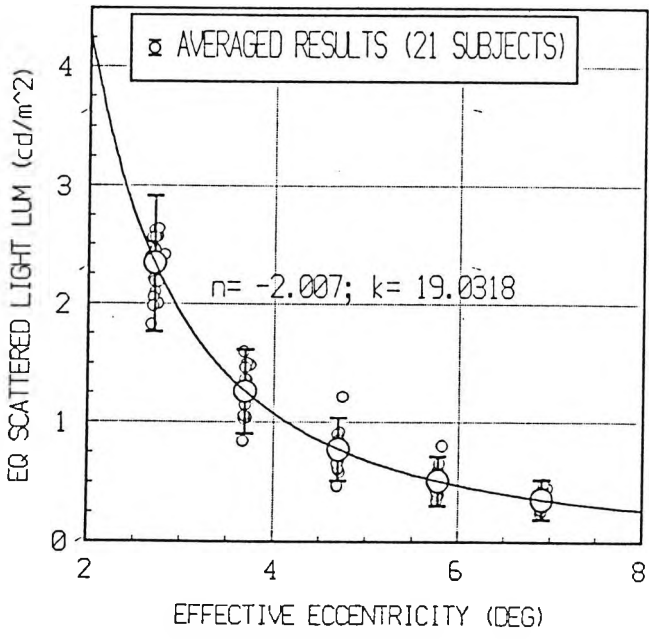


Figure 2. Averaged, best-fit model parameters for 21 normal subjects plotted as a continuous line together with individual data points. The error bars represent ± 2 standard errors calculated from the best fit data function for each object. The averaged data are described by the empirical light scatter function $L_s = k.E/\Theta^n$, where k and n are constants, Θ represents the *effective eccentricity* of the annulus (ie., a function of n and hence different for different observers) and, E, represents the illuminance level in the plane of the cornea.

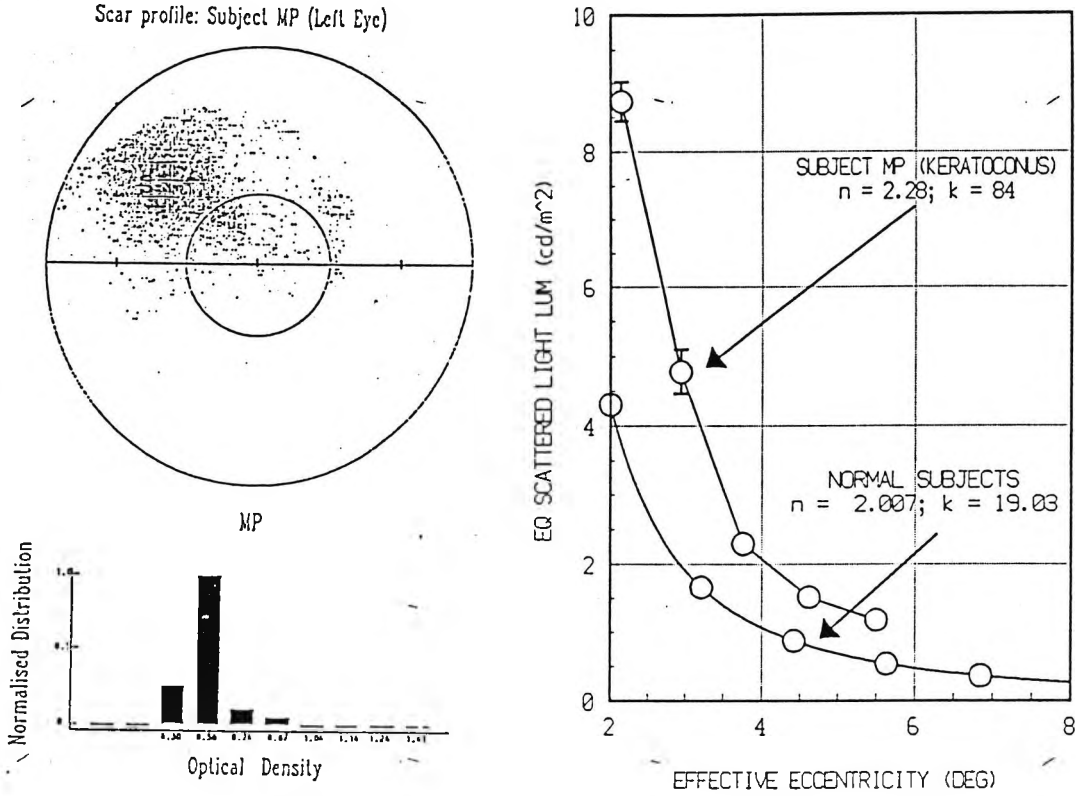


Figure 3. Typical scar profile obtained by measuring the back scatter from damaged corneal tissue in a patient with keratoconus (section on the left), and measurement of light scattered onto the retina using the stimulus configuration shown in figure 1 (section on the right). A radius of 3 mm measured with respect to the centre of the pupil is shown. The normalised frequency histogram plotted below the corresponding scar profile shows the distribution of optical density in the scar, measured with respect to background level. (ie., Log (L_s/L_b), where L_s represents the relative luminance of a given point in the image of the cornea and L_b represents the average background level). The absolute measurements of light scattered on the retina depend on the luminance and area of the scattering source and the viewing distance. The algorithm implemented in the program allows the iterative computation of the best model parameters which fit the measured experimental data.

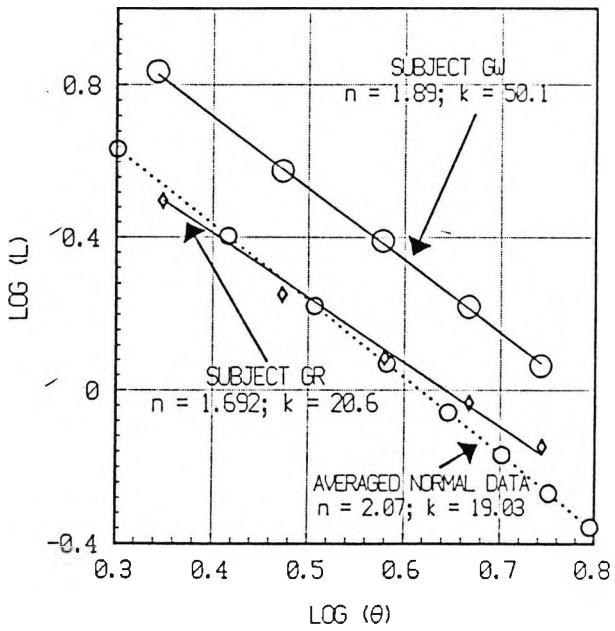


Figure 4. Measurements of scattered light in two subjects with cortical cataracts. The continuous lines represent best-fit model predictions for the parameters shown on the diagram for each subject. The averaged normal data are shown as a dotted line. In spite of the apparently *normal* k value for subject GR, the more gradual decrease in scattered light with eccentricity, as reflected by the significant decrease in n value for this subject, causes large impairment of vision and a significant loss of contrast sensitivity.

carried out, the effective eccentricities of each of the extended annuli have to be calculated. This is achieved by integrating the scatter equation over the extended source for each chosen value of n. The eccentricity of a very narrow annulus which contributes the same amount of scattered light over the measurement area as the extended scattering annulus can then be calculated for a given value of n. Iterative numerical methods are used to compute effective eccentricities for each value of n and to carry out weighted, linear regression analysis on the experimental data. The best-fit light scatter model parameters are obtained when the value of n extracted from the regression analysis matches that used in the computation of effective eccentricities.

Preliminary results on the measurement of corneal scar densities from back-scattered light are also presented. For such measurements, we developed an imaging system which is based on a modified Zeiss slit lamp which allows accurate alignment of the eye and capture of the corneal image by means of a CCD camera. The centre of the pupil is located automatically and graphical methods are used to isolate scarred areas and to eliminate specular images. The luminance level of the scar is quantised into 14 levels for ease of analysis. The area of the scar is then expressed as a percentage of the total pupil area for a range of pupil radii.

EXPERIMENTAL FINDINGS Preliminary results show that small changes in light scatter parameters can be measured accurately. For a given eye, the technique yields light scatter parameters which are relatively independent of stimulus size, viewing distance and scatter source luminance, provided changes in pupil size are minimised. The function displayed in figure 2 represents the averaged data for 21 normal subjects. The value of k provides a measure of the overall light scatter level and the parameter n determines its distribution away from the scattering source. The parameters obtained in this study are in close agreement to those reported by Fry and Alpern (1953) and Vos (1963). Typical results obtained in patients with keratoconus are shown in figure 3. The large increase in scattered light level correlates well with the measured scar profiles and corresponding densities. Patients with various forms of cataract have also been investigated. The results presented in figure 4 show why a single *light scattering factor* which does not take into account the eccentricity dependence of scattered light fails to identify and describe some cases of cataract. Subject GR (see figure 4) has normal light scatter levels according to the value of k measured, but his vision is grossly impaired by what can be described as visibility glare which causes a massive loss in the subject's contrast sensitivity. The n value measured for this subject is, however, significantly smaller and describes the more gradual decrease in scattered light with eccentricity. This suggests that smaller scattering centres are involved and they may account for this subject's glare disability. Measurements of scattered light in the eye were also carried out under natural and enlarged pupil conditions so as to investigate the effect of pupil diameter on such measurements. Preliminary observations show that changes in pupil size can cause significant changes in the best-fit light scatter model parameters in normal subjects. The results suggest that unless the size of the pupil is known at the time of measurement and its effects accounted for, small changes in light scatter parameters cannot be separated from the effects of pupil size and therefore they cannot be attributed to other factors.

REFERENCES

Barbur J L (1991). Methods for studying the behaviour of the pupil. Journal of Psychophysiology, **5**, pp 223-239.

Fry G A and Alpern M (1953). The effect of a peripheral glare source upon the apparent brightness of an object. J. Opt. Soc. Amer., **43**, pp 189-195.

Holladay L L (1927). Action of a light source in the field of view in lowering visibility. J. Opt. Soc. Am., **14**, pp 1-15.

Rayleigh (1912) The scientific papers of Lord Rayleigh, Vol. 1 and 4, Cambridge University Press, New York.

Stiles W S and Crawford B H (1937). The effect of a glaring light source on extra foveal vision. Proc. R. Soc. B. **122**, pp 255-280.

Van den Berg T J T P (1986). Importance of pathological intraocular scatter for visual disability. Documenta Ophthalmologica, **61**, pp 327-333.

Van den Berg T J T P and Spekreijse H (1987). Measurement of the straylight function of the eye in cataract and other optical media disturbances by means of a direct compensation method. Investigative Ophthalmology and Visual Science, (Suppl.), **28**, pp 397.

Vos J J (1984). Disability glare: A state of the art report. CIE Journal, **2**, pp 39-53.

Vos J J and Bouman M A (1959). Disability glare: theory and practice. Proc. CIE, Brussels, pp 298-306.