Ray Tracing in a 3-D Atmosphere Model

by

Wesley J. Friesen

A thesis
presented to the University of Manitoba
in partial fulfillment of the requirements
for the degree of
Master of Science

Department of
Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada 1991

December 7, 1991

ISBN 0-315-78012-6

RAY TRACING IN A 3-D ATMOSPHERE MODEL

BY

WESLEY J. FRIESEN

A thesis submitted to the Faculty of Graduate Studies of
the University of Manitoba in partial fulfillment of the requirements
of the degree of

MASTER OF SCIENCE

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Abstract

The history of mirages and mirage simulation is reviewed. A mirage is a distorted view of a distant object. Mirage simulation can be broken into two parts. The first part is simulating the atmospheric effects on light ray paths and the second is simulating human perception of the affected ray paths. An atmosphere model consisting of confocal ellipses defined by temperature and height is described. A program which implements ray tracing in the atmosphere is developed and verified. In the ray tracing program a predictor-corrector method is applied to improve ray path accuracy.

Next a program is described which simulates the mirage view by using the ray tracing results and a photo of the normal view. Several comparisons to previous results are made to check the simulation process. Finally some new simulations are presented which explore the impact of different model parameters on the resulting mirage.

Acknowledgements

It is with great enthusiasm that I thank Professor Waldemar Lehn for his faith in me and for all the counsel and guidance given during the course of this research work. Together we have overcome gaps of distance and time to accomplish what you are about to read. I would also like to thank Professor Donald Trim for his assistance in helping sort out various mathematical confusions that I had during my thesis work.

I gratefully acknowledge the financial support of the University of Manitoba fellowship program and grant funding provided by Professor Lehn. I have greatly appreciated the opportunity to pursue graduate work. Thank You.

Dedication

This work is dedicated to those people who have significantly impacted my life. It is because of the time and energy they invested in me that I have come as far as here. I hope that they are proud of this accomplishment.

I also wish to thank a GOD who so graciously granted me the skill and ability to do such as this. He continues to amaze me with His patience towards me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Historical Background

Awareness of atmospheric phenomena dates back to antiquity. During the reign of Alexander the Great (336-323 B.C.), an era of Greek enlightenment, the Alexandrians in Egypt made a pillar depicting a rising sun. The carving shows the upper rim colored blue with a green band below it; this effect is now known as "the Green Flash". [31]

Later, Ptolemy of Alexandria, a 2nd century astronomer, studied refraction between air, glass, and water. This was an attempt to explain the illusion of a bent stick protruding from a body of water. [39, p. 244]

The other major cultures of these times also had words for the idea of "a distorted view of a distant object resulting from the passage of light through a nonuniform medium" [1]. In our language the word which characterizes this concept is mirage.

The word mirage made its appearance in the English language in 1837. It comes from the French word *mirer* which means to look at or to be reflected. *Mirer* has its

[1] *Webster's Ninth New Collegiate Dictionary*, Thomas Allan & Son Ltd, Markham, Ontario, 1983.

roots in the Latin word *mirare*. People believed that the phenomena observed were due to "mirror like" properties of the earth.

Modern scientists continue to study mirages. The "Novaya-Zemlya phenomenon" was first noted in 1597 when Captain Willem Barents [4], while wintering in the high artic, observed the sun appearing fourteen days before it was expected. Another type of mirage, the Fata Morgana, was described by Father Angelucci in the seventeenth century [7]. In this phenomenon the ground appears to rise up and form mountains or other shapes on the horizon. It was also observed by Robert Peary in 1906 [7] while traveling to the North Pole. Peary observed "snow-clad summits above the ice horizon". It was Donald B. MacMillan who discovered on his expedition that the lands he saw, and which Peary had seen, were a mirage. He traveled toward the land but never reached it.

Johannes Kepler was interested in the "Novaya-Zemlya" and suggested that this phenomenon was the result of total reflection of the sun by an upper air layer [15]. Further observations made by Nansen in 1897 [30] and Shackleton on the 1914-1917 Antarctic Expedition [32] helped substantiate Kepler's hypothesis.

By experimenting with refraction Willebrord Snellius von Roijen (1591-1626) was able to formulate the exact law of refraction. He observed that light bent after it struck the surface of water. By comparing the deflected path to the undeflected path he noticed a constant relationship. We can model this situation by drawing the ray paths as in Figure 1.1. [39] In Figure 1.1 the lines BA and AF form the side of a container holding water. Path ECB represents the path of a light ray when the container is empty. If the container is filled with the appropriate amount of water

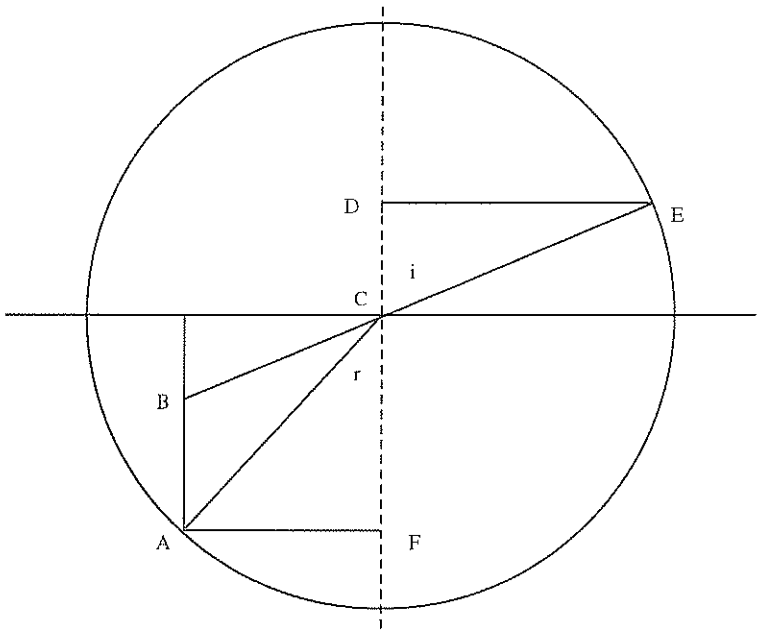


Figure 1.1: **The Angle of Refraction of Light**

then the light ray bends at C, the water surface, and hits the container side at point A. Snell observed that for water the ratio of CA over CB was constant.

$$\frac{CA}{CB} = constant$$

By using modern trigonometry it is fairly simple to derive the law of refraction from this relationship. Let us label ∠ECD as the incident $\angle i$ and ∠ACF as the refracted $\angle r$. In addition recognize ∠BCF is the same as $\angle i$.

If we multiply Snell's equation by $\frac{AF}{AF} = 1$ we get

$$\frac{AF}{AF} * \frac{CA}{CB} = constant$$

or

$$\frac{AF}{CB} * \frac{CA}{AF} = constant$$

Notice that for $\angle r$ that CA is the hypotenus and AF is the length of the opposite side. Thus $\frac{AF}{CA} = \sin r$ and similarly $\frac{AF}{CB} = \sin i$. Substituting into the above equation

$$\sin i * \frac{1}{\sin r} = \frac{\sin i}{\sin r} = constant$$

Thus we have derived the law of refraction. Snell did not publish his result but showed several people his manuscript.

Descartes attempted to prove the laws of reflection and refraction by mechanical means. Using an analogue between light and a ball he could explain that velocity of the ball parallel to the surface was unaffected by the reflection while the perpendicular velocity was reversed. Similarly for refraction he theorized that the perpendicular speed of the ball changed in different media. In order for his theory to work the speed of the ball had to increase in denser media, something few people could accept.

Fermat approached the problem from yet another angle by applying a method for determining the maximum and minimum values of a variable quantity. By assuming time traveled to be minimum it was possible for Fermat to deduce Snell's law.

Fermat's Principle

The path taken by light in traveling from one point to another point is such that the time of travel is a minimum when compared with nearby paths. [36]

Other contributors to the expanding understanding of optics included Francesco Maria Grimaldi (1618-1663), a Jesuit Professor of Mathematics, Robert Hooke, Ole Römer (1644-1710), Christian Huygens (1629-1695), and Isaac Newton (1642-1727). Robert Hooke introduced the idea of layered atmosphere with slowly varying refractive indices for analyzing astronomical refraction.

1.2 Mirages

Mirages take many forms and shapes. The purpose of this section is to introduce and briefly explain the mechanics of mirages.

Some mirages are hardly noticed; one of these is the watery appearance of highways on hot summer days. This sight is seldom the cause of much interest. What is happening here?

We see objects because light rays hit the object and are reflected away. Some of these rays are reflected in the direction of an observer. These rays will enter the eye and be focused on the retina. The image on the retina is transmitted to the brain

for processing. It is during the transmission of the light rays from the object to the observer that the ray paths are affected by the atmosphere. Light rays will travel in straight lines through media with constant index of refraction. Index of refraction is the ratio of the velocity of radiation in the first of two media to its velocity in the second as it passes from one into the other. For light this ratio compares the velocity of radiation in the media with respect to the velocity of light in a vacuum. Thus constant index of refraction means that the velocity of light is constant in that medium. The atmosphere does not possess this quality.

The atmosphere has varying temperature and pressure, two quantities which strongly affect the index of refraction. Thus the atmosphere has a varying index of refraction. This property will lead to light rays bending during passage through the atmosphere. Under normal conditions the object being seen is usually low to the ground where pressure and temperature are very nearly constant leaving the light rays to travel in essentially straight lines. These rays do actually curve slightly downward due to the natural temperature gradient of the atmosphere. This results in an extension of our vision past the normal horizon. Most of the mirages we are interested in have objects based on the surface of the earth. It is possible to approximate that the index of refraction is mainly dependent on temperature in these situations.

It is intuitive that the stronger the change in index of refraction the more light will bend. Similarly, since the index of refraction is dependent on temperature, the stronger the change in temperature the sharper light will bend. Let's consider temperature variations in the atmosphere. As mentioned earlier the part of the atmosphere we are interested in is close to the ground. Close to the surface it is possible to approximate pressure as constant since its variation will be relatively small compared

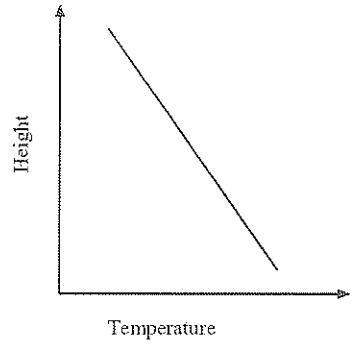


Figure 1.2: **Decreasing Temperature Profile**

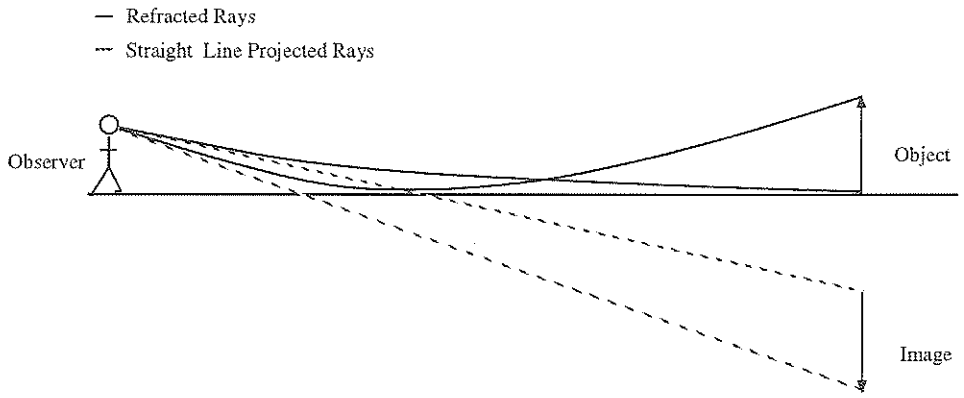


Figure 1.3: **Inferior Mirage Light Refraction**

to temperature variations. Temperature will definitely vary strongly with height and may also vary in lateral extent. Because of the continuous nature of the atmosphere it should be possible to connect points of the same temperature to form surfaces. These surfaces should form distinct layers of air at approximately the same temperature.

The next step is to determine what affect an increasing or decreasing temperature would have on a light ray path.

When temperature decreases with height as in Figure 1.2 then the index of refraction will increase with height. According to Snell's law when light travels into a

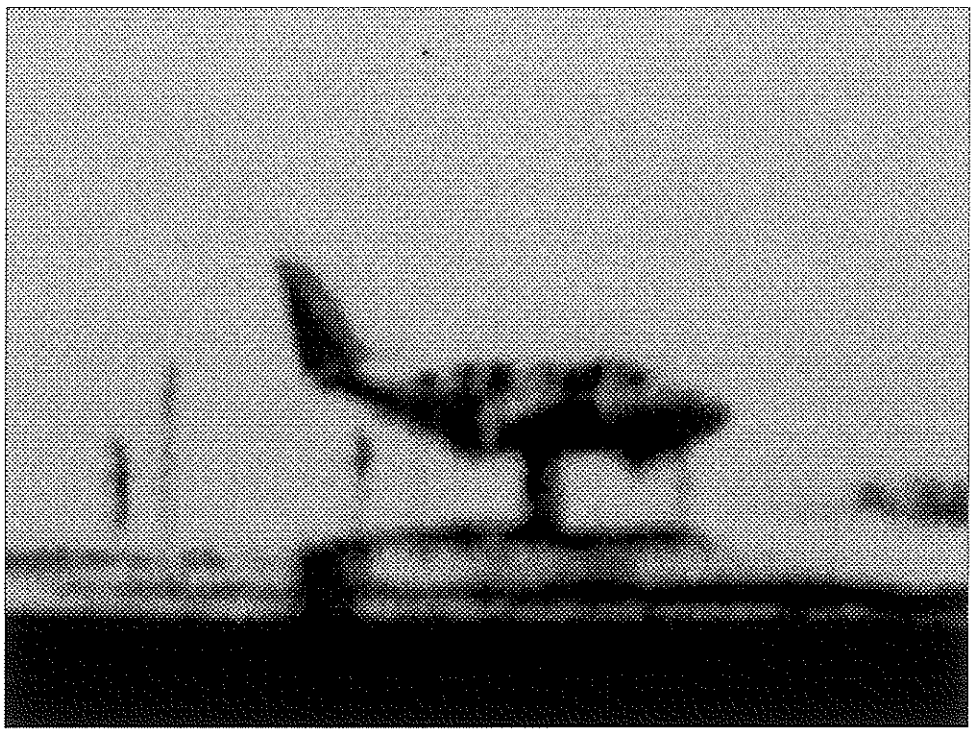


Figure 1.4: **Inferior Image of Small Plane**

medium with higher index of refraction the light bends towards the interface normal. This means that the ray will curve into the higher index region. Such a ray would curve upward in our atmosphere. This effect is demonstrated in Figure 1.3 where an observer receives two upwardly curved rays that originated from the object.

The observer perceives that these rays originated as straight lines and projects the ray paths back to see an inverted image below where the normal image is. This type of mirage is called an inferior mirage because the image is below the object and is associated with temperature decrease with respect to height. Figure 1.4[2]shows an actual image taken in 1979 at approximately 1200 meters from the plane. Notice the

[2]**Photo** 79-6-30

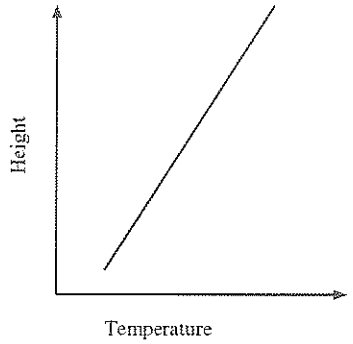


Figure 1.5: **Increasing Temperature Profile**

thin inverted image of the plane below the actual plane.

In the next case, temperature increases with height as in Figure 1.5. In such an atmosphere the index of refraction will decrease with height. When light travels into a region of lower index of refraction it will bend away from the interface normal. This means that in our atmosphere the ray will curve downward. The stronger the temperature gradient the more pronounced will be the ray curvature.

An example of this is Figure 1.6 where the rays leaving the object curve down to hit the observer's eyes. Again the eye will assume that the rays are straight and see an upright object which is much higher than the normal view gives. This type of view with a raised upright images is called a superior mirage because the images is above the object.

See Figure 1.7 [3] which is a photo of Whitefish, a hill at Tuktoyaktuk, NWT. This is a classic example of a superior mirage. This actual mirage is more complicated than the simple model presented in Figure 1.6. The hanging images contains a small upright image on top a larger inverted image. The inverted image is the result of

[3] **Photo** 79-4-20

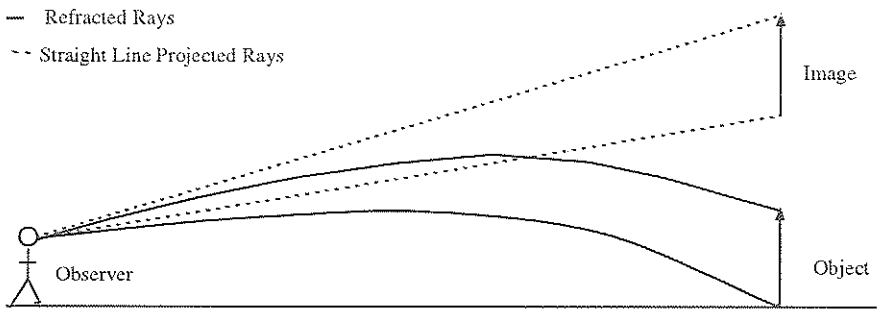


Figure 1.6: **Superior Mirage Light Refraction**

Figure 1.7: **Superior Mirage of Whitefish**

downward curved ray paths with lessening radius.

These are simple cases and seldom is the temperature gradient linear over the entire lower atmosphere. Using this basis we can go on and explain the three major types of superior mirages. The first is called the *hillingar* effect which is associated with mild temperature inversions, that is increasing temperature with increasing height. It features slightly downward bent ray paths. Figure 1.8 [22][4] shows some typical ray paths. An actual photo is shown in Figure 1.9[5]. Notice the trees on the far shore, now compare this to the normal image in Figure 1.14[6]. The trees are are introduced by downward bending rays which reached the far shore. These photos were taken at Arnes on Lake Winnipeg. The observer is about 3 km from the point seen to the right in the photo.

The *hafgerdingar* effect is associated with a strong, nonuniform temperature inversion. This will cause different ray paths to have different radii of curvature. Figure 1.10 [22][7] shows some typical ray paths and Figure 1.11 [8] shows an actual mirage of this type. Notice the vertical banding that suggests a discontinuity just below the trees on the point, this is characteristic of *hafgerdingar* mirages.

The final type is the Novaya Zemlya effect. It is associated with uniform temperature up to a certain height and then a strong inversion which sends the rays back toward the earth where they are returned upward again. These rays can travel hundreds of kilometers oscillating up and down before breaking free of this effect.

[4] see Figure 2 of [22]
[5] **Photo** 31-12
[6] **Photo** southpt14 2 Sept 1978
[7] see Figure 3 of [22]
[8] **Photo** 45-22

Figure 1.8: **Hillingar Effect**

Figure 1.9: **Hillingar on Lake Winnipeg**

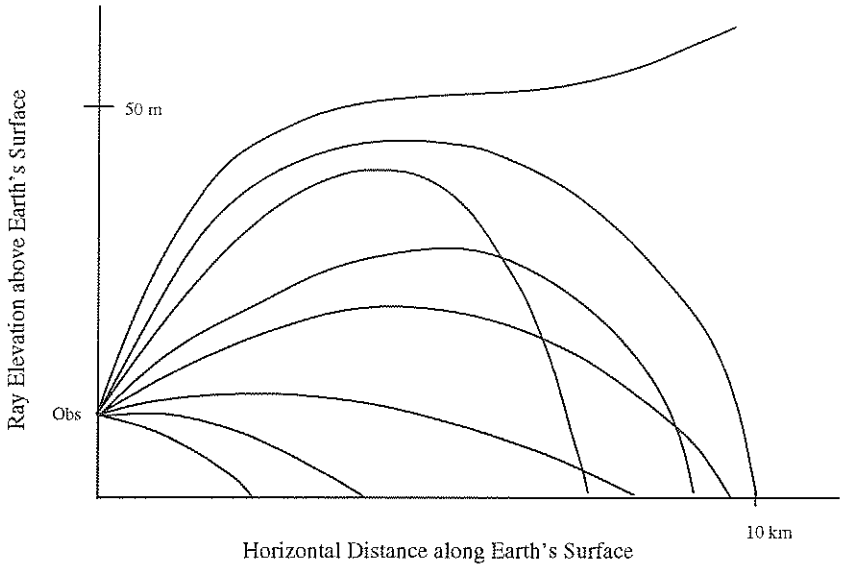


Figure 1.10: **Hafgerdingar Effect**

Figure 1.11: **Hafgerdingar on Lake Winnipeg**

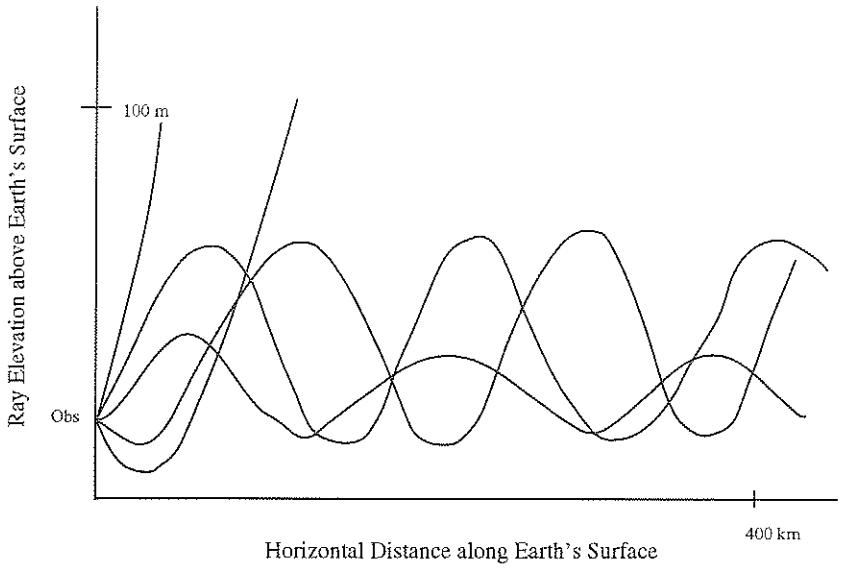


Figure 1.12: **Novaya Zemlya Effect**

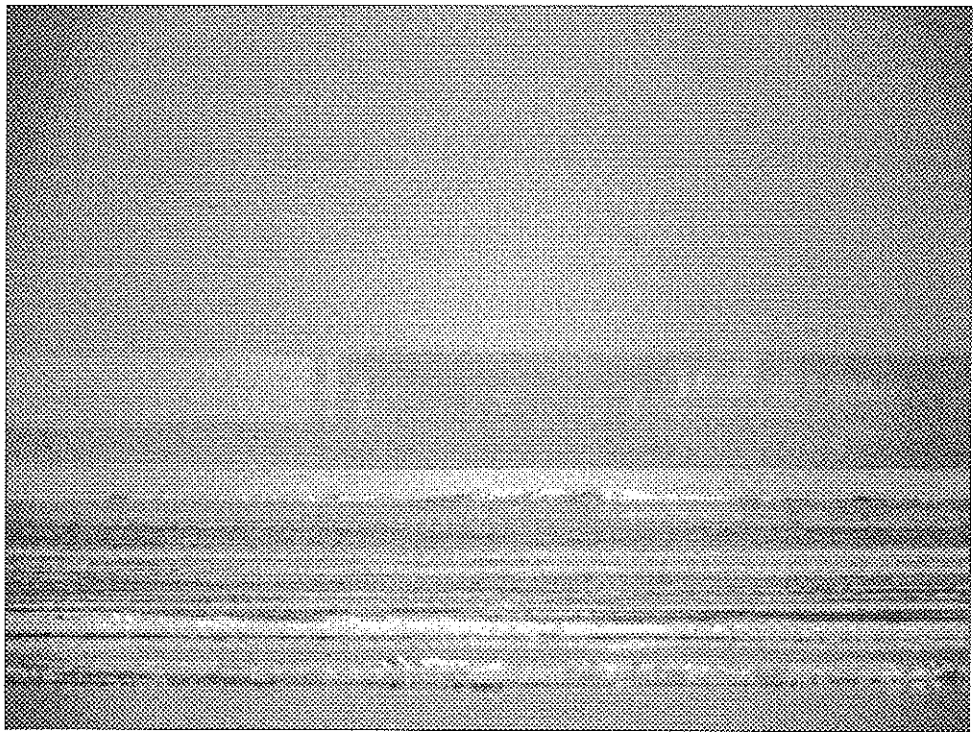


Figure 1.13: **Novaya Zemlya on Lake Winnipeg, to left of trees**

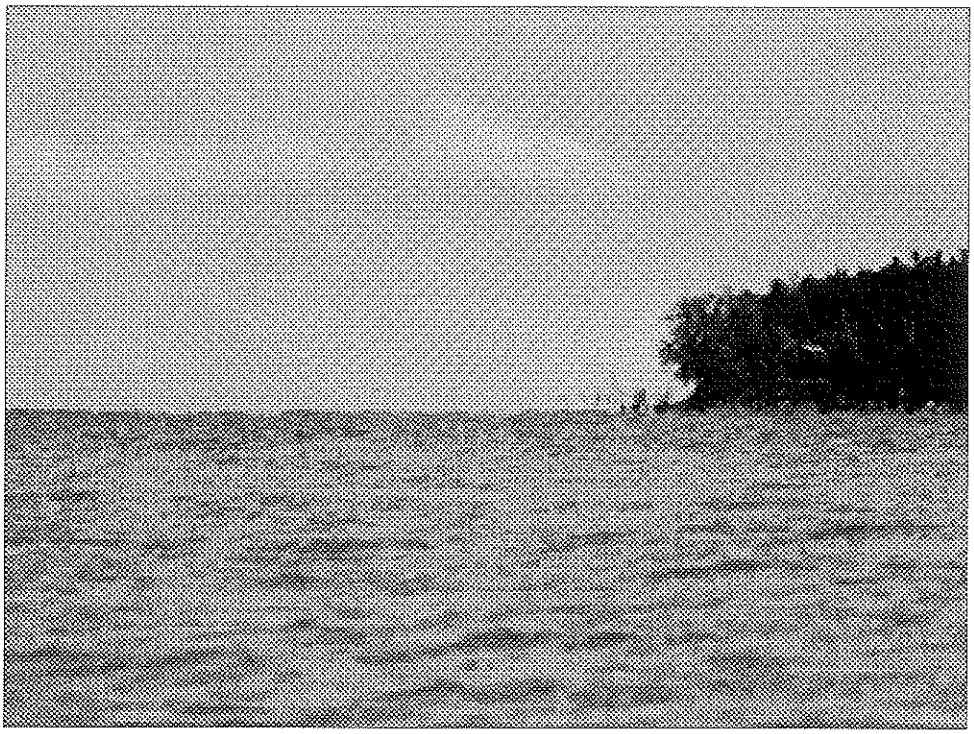


Figure 1.14: **Point on Lake Winnipeg as seen from Camp Arnes**

Figure 1.12[22][9] shows a typical ray set for this phenomena. A small group of rays will bounce back and forth between ground and the upper layer. Only a narrow band of greatly distorted image is produced by this type of atmosphere.

Examination of the photo in Figure 1.13[10] reveals a narrow band structure through which the far shore is faintly visible. This photo is taken to the left of the point in the previous photos on Lake Winnipeg. It was a warm spring day just before ice break-up.

1.3 Previous Work

Computation-based research began on the Fata Morgana type mirage in 1931. S. Fujiwhara [9] and K. Hidaka [12] both published papers that year which described the use of elliptical shells as a model for the atmosphere. These papers included a few supporting calculations to demonstrate the plausibility of their models. Detailed computation using this method was impossible at that time.

Only a few years later German Wolf-Egbert Schiele [34] wrote a paper on mirages. His 1935 paper included a review of relevant papers in the field and referenced the two Japanese papers mentioned above. He also was interested in modelling the atmosphere as confocal ellipses. By viewing an ellipse as a power series expansion he proposed that an ellipse could be modelled as a sphere plus a small perturbation. The paper is mainly theoretical in nature and involves no calculations based on the proposed model.

[9] see Figure 4 of [22]
[10] **Photo** 48-16

Sidney Bertram published a paper in 1971 [1] which described his studies of lower atmospheric refraction. He was trying to determine path errors of ray paths for correction work in areas like tracking radar, photogrammetric cameras, and laser range finders. Studies into analytic solutions of ray paths is evident in the work of A. B. Fraser. Fraser published a paper in 1977 [7] which described analytic solutions in a horizontally or spherically stratified medium. With analytic solutions it is possible to work from mirage image and normal image and determine a corresponding temperature profile. A further paper in 1979 with W. H. Mach introduced a set of nonlinear polynomial equations which give the temperature profile near the earth's surface. He compared the analysis results with field data collected at the mirage site.

Early work at the University of Manitoba began in 1973 under the guidance of Professor W. Lehn. This work resulted in a joint paper between W. Lehn and H. Sawatzky [18] in 1975. The paper discussed a procedure which was used to determine ray paths for nearly horizontal rays in the lower atmosphere. The technique used density profiles that were determined from temperature profiles for the lower atmosphere. The next step was to add visual information about how the refracted rays paths affected what was seen. A paper in 1978 with M. El-Arini [16] introduced a program which accepted an outline of an object and mapped it into an image based on the computed ray paths.

Arctic mirages were the focus of a paper written in 1979 by W. Lehn [22]. The paper discusses typical ray paths for three familiar arctic mirages. The *hillingar*, *hafgerdingar*, and Novaya Zemlya effect are discussed along with atmospheric conditions which give rise to the phenomena. After investigation of the Novaya Zemlya effect a further paper on this phenomena was written in 1981 [17] with B. German.

This paper described the atmospheric conditions and ray paths which produced the Novaya Zemlya effect in computer simulations. This information is used to refine our understanding of the Novaya Zemlya.

In 1983 W. Lehn published a paper [21] discussing how to analyze photographic data of mirages in order to postulate possible temperature profiles for the atmosphere which caused the mirage based on a multi-layered spherical shell model of the atmosphere. Also, in a 1985 paper W. Lehn described a simple model for atmospheric ray tracing. His model was composed of a series of small concentric layers with the surface of each layer being isothermal. With the additional simplification that the temperature varies linearly between the surfaces it was possible to implement ray tracing on computer. John Bock in his B.Sc. thesis in 1984 [2] extended Hidaka's work by using a computer to do the mathematics involved. Bock approximated his atmosphere as a series of confocal isothermal elliptical cylinders with constant index of refraction between shells. He used straight line projections of rays and made the assumption that all vertical planes of interest would produce the same ray paths which reduces the necessary calculations to one plane involving the observer. The approximations that Hidaka used in his work were found to create a downward bias on predicted ray paths. As a result Hidaka's approach was discarded on favour of successive application of Snell's law.

The next generation of the elliptical model involved adding a spherical earth surface and linear refractive index variation between shells. This work was done by Lorne Midford [26] in his B.Sc. thesis (1985). That same year John Morrish completed his M.Sc. work which dealt exclusively with inferior mirages. Using the spherical atmosphere models developed by W. Lehn he wrote computer programs to do ray tracing

for inferior mirages. Using these programs he analyzed data gathered at Tuktoyaktuk, N.W.T. (1983). Further, he attacked the difficult problem of determining temperature profiles given the normal image and mirage image.

It was P. Isaak who made the first study of a three dimensional atmosphere at the University of Manitoba [13]. His model used a spherical earth with a series of confocal ellipsoidal shells forming the atmosphere. Each shell surface was isothermal. The refractive index was assumed to vary linearly between atmospheric shells. His B.Sc. thesis (1986) was developing a computer program to implement this model. My own B.Sc. thesis (1988) work was implementing the recommendations of Isaak and Midford, as well as continuing work started earlier with M. El-Arini. This involved developing software which applied transfer characteristic information to a normal image to produce a mirage image. A transfer characteristic is the mapping of distorted ray path end point to observed ray path end point. The images used were actual site photos digitized and stored in a 512 by 512 pixel format with the standard 256 grey scale pixel levels.

1.4 Applications

Applications of ray tracing include expanding our understanding of lower atmospheric light transmission. This knowledge can be applied to a variety of areas including tracking radar, laser surveying, laser range finders, and surface signal transmission.

Novel applications include the study of graded index optic fibers using ray tracing. The Journal of Applied Optics over the last two decades has included many papers on geometric optics ray tracing and on ray tracing in the study of graded index optics.

1.5 Problem Description

I intend to briefly describe the refinement process of previous work to the atmospheric model and ray tracing technique.

The first atmospheric models were two dimensional and used spherical models for the atmospheric layers. Then the atmosphere model was changed to elliptical shells, still in two dimensions. Elliptical shells give more flexibility in modeling various situations and can be made to assume almost spherical shape. Next came the first three dimensional atmospheric model with ellipsoidal shells. Each model included successive refinements in the earth's surface and in shell temperature profiles. However, the ray paths in the latest model demonstrated a strong dependence on step size during ray path tracing that indicates a need for further refinement in the ray tracing process.

Simultaneously work on a program to apply the results of ray path tracing to normal images was going on for two dimensional models. This project proved its value in helping analyze the ray path data and by giving it a visual interpretation. Even more so the expanded data from a three dimensional model requires the ability to visually display the effect of the ray path deviations on the normal image.

Finally although some basic testing was done by Isaak on the three dimensional model no attempt to systematically explore the impact of ellipsoidal shells was made.

1.6 Scope of Project

This research project seeks to cover the following areas:

1. Investigate and develop an understanding of ray tracing in a three dimensional atmospheric model.

2. To improve the previous method of ray tracing through improving data consistency against step size variations by introducing predictor - corrector methods into ray path calculations.

3. Continue to make ray paths visually accessible by incorporating three dimensional ray path plotting.

4. To develop a method of applying ray path data to normal images in order to produce a distorted image representing the atmospheric affects.

5. To utilize the ray tracing and image programs to analyze several typical temperature, images, mirage data sets.

6. To explore the implications of the ellipsoidal atmosphere.

Chapter 2

Ray Tracing in an Ellipsoidal Atmosphere

2.1 Geometric Optics

Geometric Optics [1] it is one of the oldest models for light behaviour having its origins in classical Greece. Even as the modern view of light as waves became widely understood the primitive model of Geometric Optics remained firmly entrenched as a practical tool in the solution of optical problems. Geometric Optics has been applied to problems in lasers, interference, diffraction, and waveguides. It is however most widely applied in the field of optics design.

Geometric Optics is based on Fermat's principle of minimal transit time. This means that a given ray will travel the path between two points which takes the least time. For homogeneous media this means ray paths will be straight lines. In stratified homogeneous media light rays will bend at layer intersections according to Snell's law. Our concern is with light rays whose medium has a constantly varying media. We

[1] *McGraw-Hill Encyclopedia of Science & Technology*, 6th Ed., McGraw-Hill, New York, 1987

begin with Maxwell's Equations for waves and restrict our consideration to visible light wavelengths.

Born [3] begins his consideration of Geometric Optics by describing it as corresponding to the limiting case of $\lambda_0 \rightarrow 0$. This neglecting of wavelength leads to interesting approximations to the laws of optics which allow the individual light paths to be described simply.

By considering Maxwell's equations as $\lambda_0 \rightarrow 0$ the following relationship can be derived:

$$(\nabla \mathcal{L})^2 = \eta^2; \tag{2.1}$$

where η = index of refraction = $\sqrt{\mu\epsilon}$ and $\mathcal{L}$ is the optical path and is a real scalar function of position. It is possible to define a vector that is normal to the surface $\mathcal{L} =$ *constant* and normalized in length by recognizing that the gradient operation produces a vector normal to the given function. The surface $\mathcal{L} =$ *constant* is a geometrical wave surface or wavefront.Thus s a unit normal vector can be defined as,

$$\mathrm{s} = \frac{\nabla \mathcal{L}}{\mid \nabla \mathcal{L} \mid} = \frac{\nabla \mathcal{L}}{\eta}. \tag{2.2}$$

If r(s) denotes the position vector of a point P on the optical path $\mathcal{L}$ with respect to the parametric value s which is length along the path then,

$$\frac{d\mathbf{r}(s)}{ds} = \mathrm{s}. \tag{2.3}$$

Further then

$$\frac{d\mathbf{r}(s)}{ds} = \frac{\nabla \mathcal{L}}{\eta} \quad or \quad \eta \frac{d\mathbf{r}(s)}{ds} = \nabla \mathcal{L}. \tag{2.4}$$

Next both sides of Equation 2.4 are again differentiated with respect to s, the arc length, giving

$$\frac{d}{ds}\left(\eta\frac{d\mathbf{r}(s)}{ds}\right) = \frac{d}{ds}(\nabla\mathcal{L}). \tag{2.5}$$

If we apply the chain rule to the right side of Equation 2.5 then we get

$$\frac{\partial(\nabla\mathcal{L})}{\partial x}\frac{dx}{ds} + \frac{\partial(\nabla\mathcal{L})}{\partial y}\frac{dy}{ds} + \frac{\partial(\nabla\mathcal{L})}{\partial z}\frac{dz}{ds}. \tag{2.6}$$

It is possible to separate this term into a dot product as follows:

$$\left(\frac{dx}{ds}\hat{i} + \frac{dy}{ds}\hat{j} + \frac{dz}{ds}\hat{k}\right)\cdot\left(\frac{\partial(\nabla\mathcal{L})}{\partial x}\hat{i} + \frac{\partial(\nabla\mathcal{L})}{\partial y}\hat{j} + \frac{\partial(\nabla\mathcal{L})}{\partial z}\hat{k}\right). \tag{2.7}$$

By recognizing that the second term is the gradient of the vector function $\nabla\mathcal{L}$ it is possible to write the equation as

$$= \frac{d\mathbf{r}(s)}{ds}\cdot\nabla(\nabla\mathcal{L}).$$

Next substituting from Equation 2.4 into the first term gives

$$\frac{\nabla\mathcal{L}}{\eta}\cdot\nabla(\nabla\mathcal{L}) \tag{2.8}$$

$$= \frac{1}{\eta}\left(\nabla\mathcal{L}\cdot\nabla(\nabla\mathcal{L})\right)$$

$$= \frac{1}{\eta}\left[\nabla\mathcal{L}\cdot\left[\frac{\partial(\nabla\mathcal{L})}{\partial x}\hat{i} + \frac{\partial(\nabla\mathcal{L})}{\partial y}\hat{j} + \frac{\partial(\nabla\mathcal{L})}{\partial z}\hat{k}\right]\right].$$

We can examine this further if we recognize that the partial and gradient operators can be reversed without changing the results;

$$\frac{1}{\eta}\left[\nabla\mathcal{L}\cdot\left[\nabla\left(\frac{\partial\mathcal{L}}{\partial x}\right)\hat{i} + \nabla\left(\frac{\partial\mathcal{L}}{\partial y}\right)\hat{j} + \nabla\left(\frac{\partial\mathcal{L}}{\partial z}\right)\hat{k}\right]\right]$$

$$= \frac{1}{\eta}\left[\frac{\partial\mathcal{L}}{\partial x}\hat{i} + \frac{\partial\mathcal{L}}{\partial y}\hat{j} + \frac{\partial\mathcal{L}}{\partial z}\hat{k}\right]\cdot\left[\nabla\left(\frac{\partial\mathcal{L}}{\partial x}\right)\hat{i} + \nabla\left(\frac{\partial\mathcal{L}}{\partial y}\right)\hat{j} + \nabla\left(\frac{\partial\mathcal{L}}{\partial z}\right)\hat{k}\right]$$

$$= \frac{1}{\eta}\left[\frac{\partial \mathcal{L}}{\partial x}\left(\nabla\left(\frac{\partial \mathcal{L}}{\partial x}\right)\right) + \frac{\partial \mathcal{L}}{\partial y}\left(\nabla\left(\frac{\partial \mathcal{L}}{\partial y}\right)\right) + \frac{\partial \mathcal{L}}{\partial z}\left(\nabla\left(\frac{\partial \mathcal{L}}{\partial z}\right)\right)\right]. \tag{2.9}$$

Let us break down the first term of Equation 2.9 to see if it can be simplified.

$$\frac{\partial \mathcal{L}}{\partial x}\nabla\left(\frac{\partial \mathcal{L}}{\partial x}\right) = \frac{\partial \mathcal{L}}{\partial x}\left(\frac{\partial}{\partial x}\frac{\partial \mathcal{L}}{\partial x}\hat{i} + \frac{\partial}{\partial y}\frac{\partial \mathcal{L}}{\partial x}\hat{j} + \frac{\partial}{\partial z}\frac{\partial \mathcal{L}}{\partial x}\hat{k}\right)$$

$$= \left(\frac{\partial \mathcal{L}}{\partial x}\frac{\partial^2 \mathcal{L}}{\partial x^2}\hat{i} + \frac{\partial \mathcal{L}}{\partial x}\frac{\partial^2 \mathcal{L}}{\partial x \partial y}\hat{j} + \frac{\partial \mathcal{L}}{\partial x}\frac{\partial^2 \mathcal{L}}{\partial x \partial z}\hat{k}\right)$$

$$= \frac{1}{2}\frac{\partial}{\partial x}\left(\frac{\partial \mathcal{L}}{\partial x}\right)^2\hat{i} + \frac{1}{2}\frac{\partial}{\partial y}\left(\frac{\partial \mathcal{L}}{\partial x}\right)^2\hat{j} + \frac{1}{2}\frac{\partial}{\partial z}\left(\frac{\partial \mathcal{L}}{\partial x}\right)^2\hat{k}$$

$$= \frac{1}{2}\nabla\left(\frac{\partial \mathcal{L}}{\partial x}\right)^2. \tag{2.10}$$

Next substitute this simplified term into Equation 2.9 along with similar substitutions for the other terms. This gives

$$\frac{1}{2\eta}\left[\nabla\left(\frac{\partial \mathcal{L}}{\partial x}\right)^2 + \nabla\left(\frac{\partial \mathcal{L}}{\partial y}\right)^2 + \nabla\left(\frac{\partial \mathcal{L}}{\partial z}\right)^2\right]. \tag{2.11}$$

Due to the distributive nature of differentials this can be rewritten as

$$\frac{1}{2\eta}\nabla\left[\left(\frac{\partial \mathcal{L}}{\partial x}\right)^2 + \left(\frac{\partial \mathcal{L}}{\partial y}\right)^2 + \left(\frac{\partial \mathcal{L}}{\partial z}\right)^2\right]$$

$$= \frac{1}{2\eta}\nabla\left[(\nabla\mathcal{L})^2\right]. \tag{2.12}$$

Thus Equation 2.5 may be rewritten as

$$\frac{d}{ds}\left(\eta\frac{d\mathbf{r}(s)}{ds}\right) = \frac{1}{2\eta}\nabla\left[(\nabla\mathcal{L})^2\right]. \tag{2.13}$$

And by substitution of Equation 2.1 for $(\nabla\mathcal{L})^2$ gives

$$\frac{d}{ds}\left(\eta\frac{d\mathbf{r}(s)}{ds}\right) = \frac{1}{2\eta}\nabla\eta^2$$

$$= \frac{1}{2\eta}\left(\frac{\partial \eta^2}{\partial x}\hat{i} + \frac{\partial \eta^2}{\partial y}\hat{j} + \frac{\partial \eta^2}{\partial z}\hat{k}\right)$$

$$= \frac{1}{2\eta}\left(2\eta\frac{\partial \eta}{\partial x}\hat{i} + 2\eta\frac{\partial \eta}{\partial y}\hat{j} + 2\eta\frac{\partial \eta}{\partial z}\hat{k}\right)$$

$$= \left(\frac{\partial \eta}{\partial x}\hat{i} + \frac{\partial \eta}{\partial y}\hat{j} + \frac{\partial \eta}{\partial z}\hat{k}\right)$$

$$\frac{d}{ds}\left(\eta\frac{d\mathbf{r}(s)}{ds}\right) = \nabla\eta. \tag{2.14}$$

Born calls this the differential equation of the light ray. But this form of the equation is not very useful to us. It is by considering the curvature vector of a ray and its magnitude, which is the inverse of the radius of curvature, that a usable form of the differential equation is found.

The curvature vector $\boldsymbol{\kappa}$ is defined as,

$$\boldsymbol{\kappa}(s) = \frac{d\mathbf{T}}{ds} = \frac{d\mathbf{s}}{ds} = \frac{1}{\rho}\boldsymbol{\nu};$$

where ρ is the radius of curvature and $\boldsymbol{\nu}$ is the unit principal normal. $\mathbf{T}$ is the unit tangent vector to the curve s. If we substitute Equation 2.2 for s and expand the right side,

$$\frac{d\mathbf{s}}{ds} = \frac{d}{ds}\left(\frac{\nabla\mathcal{L}}{\eta}\right)$$

apply the chain rule for differentiation

$$= \frac{1}{\eta}\frac{d}{ds}(\nabla\mathcal{L}) + \nabla\mathcal{L}\frac{d}{ds}\left(\frac{1}{\eta}\right)$$

from Equation 2.5 and Equation 2.14

$$= \frac{\nabla\eta}{\eta} - \nabla\mathcal{L}\frac{1}{\eta^2}\frac{d\eta}{ds}$$

from Equation 2.2

$$\boldsymbol{\kappa} = \frac{\nabla\eta}{\eta} - \frac{\mathrm{s}}{\eta}\frac{d\eta}{ds}$$

multiply by η

$$\eta\boldsymbol{\kappa} = \nabla\eta - \mathrm{s}\frac{d\eta}{ds} \tag{2.15}$$

use definition of $\boldsymbol{\kappa}$

$$\frac{\eta}{\rho}\boldsymbol{\nu} = \nabla\eta - \mathrm{s}\frac{d\eta}{ds}$$

add last term on right to both sides

$$\frac{\eta}{\rho}\boldsymbol{\nu} + \mathrm{s}\frac{d\eta}{ds} = \nabla\eta$$

from chain rule $\frac{d\eta}{ds} = \nabla\eta \cdot \frac{d\mathbf{r}}{ds}$

$$\frac{\eta}{\rho}\boldsymbol{\nu} + \mathrm{s}\left(\nabla\eta \cdot \frac{d\mathbf{r}}{ds}\right) = \nabla\eta$$

by Equation 2.3

$$\frac{\eta}{\rho}\boldsymbol{\nu} + \mathrm{s}\left(\nabla\eta \cdot \mathrm{s}\right) = \nabla\eta. \tag{2.16}$$

This is the vector form of the differential equation of the light ray. This is the equation that Isaak used in his algorithm. Midford used a slightly different form of this equation. He was interested in the magnitude of $\boldsymbol{\kappa}$. If we go back to Equation 2.15 and dot both sides with $\eta\boldsymbol{\kappa}$ then substitute the definition of $\boldsymbol{\kappa}$ and simplify we get,

$$|\boldsymbol{\kappa}| = \frac{1}{\rho} = \nu \cdot \nabla\left(ln\,\eta\right).$$

This equation gives the value of the ray path radius in terms of the normal vector and the gradient of the natural logarithm of the index of refraction.

Having derived a useful form of the ray path equation we will leave the discussion of its application until after the atmosphere model has been described. However, first we need to examine the index of refraction.

2.2 Index of Refraction

Earlier it was mentioned that the index of refraction in the atmosphere is dependent on many variables. It is not necessary to be concerned with a complex equation for this function since we are really only interested in the case for light rays. Others have done the necessary studies which resulted in the discovery that for radio frequency waves the following empirical relationship holds, [1]

$$\eta = 1 + \frac{78.6 * 10^{-6}}{T}(P + \frac{4810e}{T}) \quad ^2 \tag{2.17}$$

where P = barometric pressure, millibars; e = water vapour pressure, millibars; T = temperature, Kelvin.

For light rays, the case in which we are interested, the water vapour pressure is not a factor thus reducing the above equation to,

$$\eta = 1 + \frac{78.6 * 10^{-6}}{T}P. \tag{2.18}$$

This is the general equation for the index of refraction that will be used in this study.

2.3 Atmosphere Model

Earlier the idea of the elliptical model was introduced. This model uses a series of confocal ellipses standing on the earth's surface to represent a near earth refractive

[2]Bertram's first equation is not consistant with his second. His first equation should read as given here in order to produce the second equation which is a well known relationship.

atmosphere. It was Hidaka who proposed that elliptical shapes could represent layers of constant temperature over a lake. Lakes are natural places to find temperature gradients since often the water is warmer or colder than the surrounding air. As the land air moves over the lake surface it will cool/warm toward the lake temperature. This air temperature will be nearest the lake temperature at the middle of the lake and land temperature at the edge. Naturally the land surrounding the lake will be at the same temperature. If you connect all the air points that have the same temperature it would form a shallow concave earthward surface over the lake. Together the various temperature shells would form a near earth refractive atmosphere. These shells can be modeled by elliptical surfaces which have considerable versatility in shape and size.

So far we have a series of individual shells standing on the earth's surface. Each shell must be individually sized. In order to simplify our elliptic model let's choose elliptic shells which are confocal. This means that the foci of all the ellipse are at the same point. Then we only need to specify the size of the outer shell and the height of any interior shell to completely specify any interior shell. This considerably simplifies the shell selection process.

It is natural to put the ellipses at the center of our model and its coordinate system. The elliptic shells will be centered on the x and y axes. Similarly the base of the ellipse will rest on the earth's surface.

The cartesian coordinate system was chosen for its simplicity over elliptical coordinates which have been suggested as a possibility.[12] There is one significant modification that needs to be mentioned. The elliptic shells are with respect to the earth's surface which is curved. Thus the shells will not be strictly elliptic with respect to

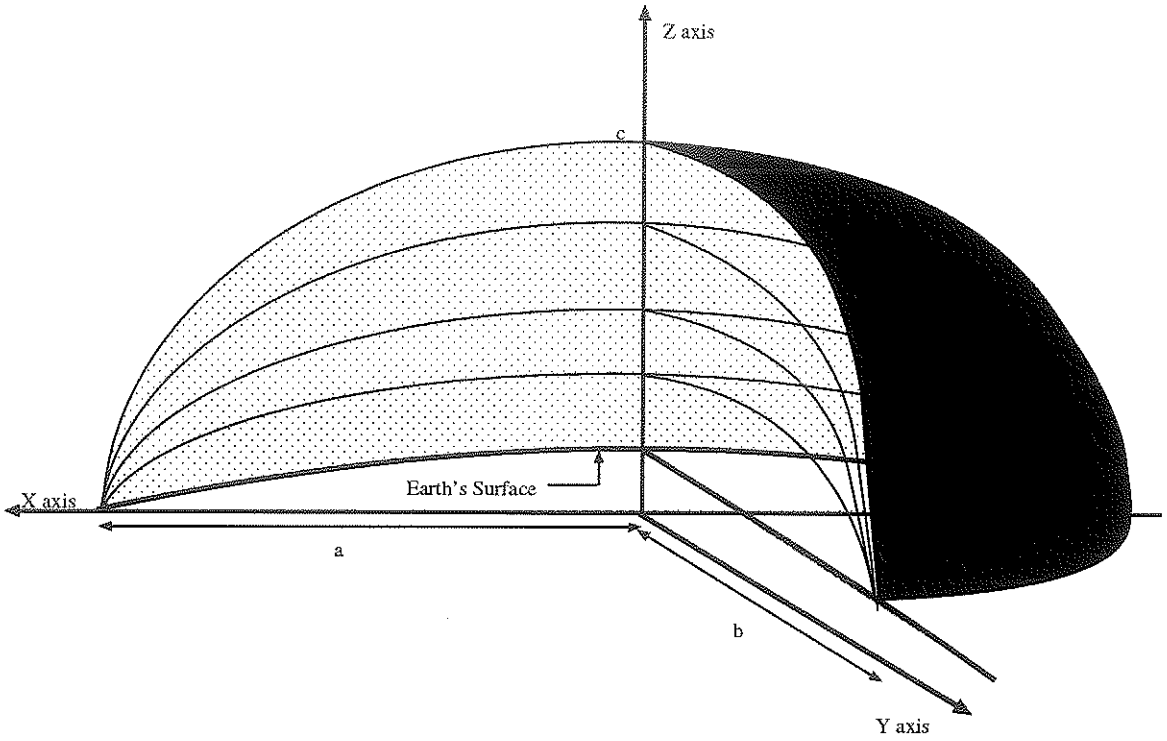


Figure 2.1: **Ellipsoidal Shells And Coordinate System**

the cartesian coordinates. The coordinates system, ellipsoid shells and earth surface are shown in Figure 2.1.

The variables a, b, and c are shell parameters used in the elliptic equation. By choice a is along the x direction, b along the y axis, and c is from the earth's surface to the top the shell. This figure shows the elliptic shell touching the z=0 plane at a along the x axis but not at b along the y axis. This is because a is the major axis. This means that a is greater than b. The major axis will always touch the earth on

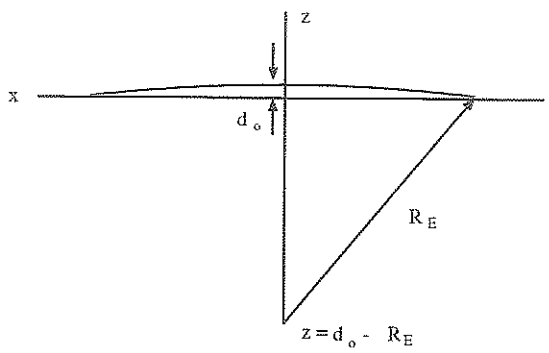


Figure 2.2: **Earth's Surface**

the z=0 plane. The equation for an ellipse is,

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1. \tag{2.19}$$

This equation represents the ellipse with respect to the earth's surface. For our coordinate system we must modify the z term to include the height of the surface of the earth at x,y. From Figure 2.2 the height of the earth is given by $x^2 + y^2 + (z - d_0 + R_E)^2 = R_E^2$. Here d_0 is the distance below the earth that the coordinate axes must be in order for the base of the ellipses to be a zero height. We can define d_0 as $d_0 = erad * (1 - \cos(\frac{t_1}{erad}))$. In this formula t_1 is the length of the arc along the earth's surface from the origin to the edge of the ellipse. The formula uses simple trigonometry to determine d_0. If we solve for z and use $R_E >> x, y$ then $z \approx d_0 - \frac{x^2+y^2}{2R_E}$. This happens to be a parabolic approximation to the earth's surface. At x=100 km the parabolic approximation will give a positive z error of 4.8 cm when compared to the true earth's surface. This error can be ignored when compared to the heights involved.

Now the elliptic equation can be rewritten with respect to the coordinate system

as,

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{(z - d_0 + \frac{x^2+y^2}{2R_E})^2}{c^2} = 1. \tag{2.20}$$

Typical outer shells will be up to twenty kilometers long, several kilometers wide and less than one hundred meters high.

Ideally the observer and object would have complete freedom to be at any location with respect to the origin. This freedom introduces many extra complications. Since the basic model had not yet been fully implemented it was decided to limit the object to be centered with respect to the x axis, located in a plane parallel to the y axis and perpendicular to the coordinate system (not the earth's surface) [3]. Further, while the observer has freedom of location, on the positive x side, his vision reference is parallel to the x axis. To see an object off to one side the view angles are adjusted to include the desired object.

2.4 Temperature Profiles

Temperatures in the elliptical model will be given by a piecewise linear function. Each corner on the piecewise function and the end points will correspond to a shell in the elliptical model. Temperature outside the model is considered constant and will give rise to a very large ray path radius. Large ray path radii result in straight lines. Temperature in the inner shell will be assumed to vary as in the layer above the inner region. Figure 2.3 shows an example of a piecewise linear temperature profile.

[3]Note, an object at a distance of 10 km from the origin would have an error of 5.4 arc minutes with respect to perpendicular to the earth's surface. This is an insignificant contribution with respect to the whole ray path and can be neglected.

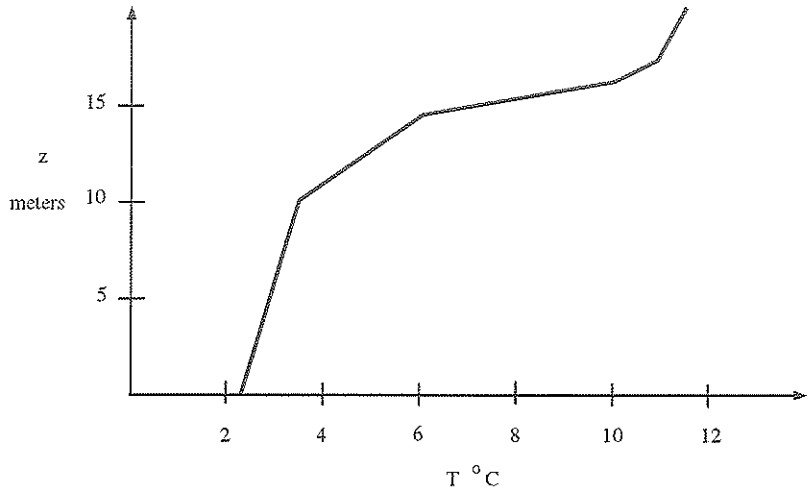


Figure 2.3: **Temperature Profile**

η = index of refraction
ρ = radius of ray path
$\boldsymbol{\nu} = \mathbf{n}$ = unit normal to light ray path
$\mathbf{s} = \mathbf{t}$ = unit tangent to light ray path

Table 2.1: **Variable Definitions**

2.5 The Ray Path Equation

Back in Section 1 we derived the vector differential equation of a light ray path.

$$\frac{\eta}{\rho}\boldsymbol{\nu} + \mathbf{s}\,(\nabla\eta \cdot \mathbf{s}) = \nabla\eta \tag{2.21}$$

In Table 2.1 the familiar symbols for the normal and tangent vectors are introduced. These traditional symbols will be used henceforth when talking about the tangent and normal vectors or Equation 2.21. This equation defines the gradient of the refractive index as a linear combination of the tangent and normal vectors of the

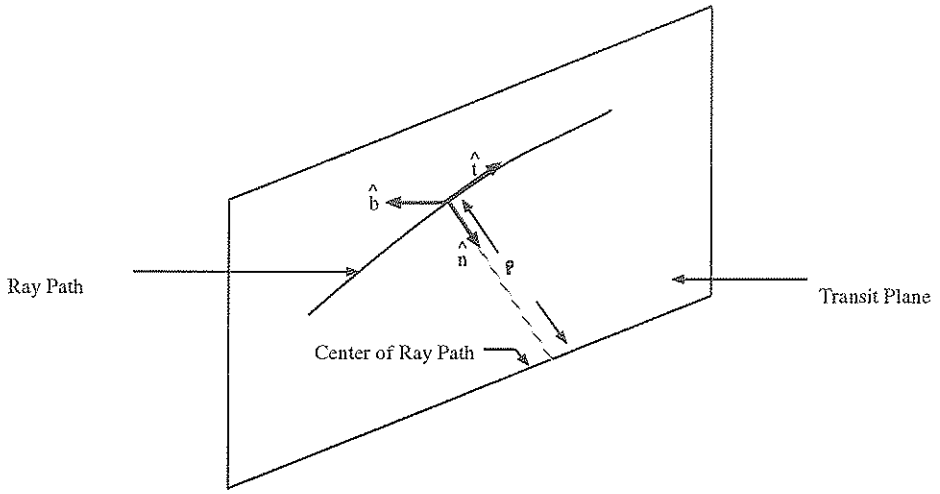


Figure 2.4: **Transit Plane**

ray path. This implies that $\nabla\eta$ is in the same plane as $\mathbf{n}$ and $\mathbf{t}$; this plane is called the transit plane. By taking the cross product of $\mathbf{t}$ and $\mathbf{n}$ a new vector $\mathbf{b}$ is introduced. $\mathbf{b}$ is the vector which represents the attitude numbers of the transit plane. The three orthonormal vectors $\mathbf{n}$, $\mathbf{t}$, and $\mathbf{b}$ form a trihedron and can be viewed as a moving coordinate system with origin on the ray path. Figure 2.4 shows the transit plane and the relationship between the three vectors. The ray path follows a sphere-plane intersection path.

How is the differential equation applied in step interative calculations in an algorithm? The following discussion explores this question.

Examine the direction of $\nabla\eta$: recall that our elliptical shells are surfaces of constant temperature. The index of refraction is defined in terms of temperature thus the shells are also surfaces of constant η. Therefore $\nabla\eta$ is going to be normal to the ellipsoidal surface.

This ellipsoidal normal vector will be in the transit plane. This vector along with

the ray path tangent, which is always known, can be used to determine the plane attitude vector for the current transit plane. i.e.

$$\mathbf{t} \times \mathbf{n}_e = \mathbf{b} \tag{2.22}$$

$\mathbf{n}_e$ - inward pointing ellipsoidic normal

The inward pointing normal rather than the outward pointing normal is used for convenience.

The assertion that the ray path tangent is always known can be traced back to the assumption that the temperature profile is at least a piecewise continuous function of elevation. If the temperature profile is piecewise continuous then so must be the index of refraction. It can be proved that if the index of refraction is continuous then the slope of the ray path is also continuous. For a proof of the above assertions see John Morrish's thesis [29]. The result of this assertion is that between any two segments of the ray path the tangent must be the same. When a new temperature shell is penetrated we automatically know that the current tangent will be the tangent for the new ray segment.

From $\mathbf{t}$ and $\mathbf{b}$ the normal to the ray path can be found from $\mathbf{b} \times \mathbf{t} = \mathbf{n}$. Thus we know all the vector directions in the ray path equation. The magnitude equation is,

$$\left(\frac{\eta}{\rho}\right)^2 + (\nabla\eta \cdot \mathbf{t})^2 = |\nabla\eta|^2 \tag{2.23}$$

We can substitute $(\nabla\eta \cdot \mathbf{t}) = \cos\gamma\,|\nabla\eta||\mathbf{t}| = \cos\gamma\,|\nabla\eta|$ giving,

$$\left(\frac{\eta}{\rho}\right)^2 + \cos^2\gamma\;|\nabla\eta|^2 = |\nabla\eta|^2 \tag{2.24}$$

From the cosine law γ is the angle between the gradient and tangent vectors. Continuing to simplify,

$$\begin{aligned}\left(\frac{\eta}{\rho}\right)^2 &= |\nabla\eta|^2 - |\nabla\eta|^2\cos^2\gamma \\ &= |\nabla\eta|^2(1-\cos^2\gamma) \\ &= |\nabla\eta|^2\sin^2\gamma. \end{aligned} \tag{2.25}$$

Thus

$$\left(\frac{\eta}{\rho}\right) = |\nabla\eta|\sin\gamma$$

or

$$\rho = \frac{\eta}{|\nabla\eta|\sin\gamma}.$$

Next we need to consider an approximation to $|\nabla\eta|$.

Consider the following equation,

$$\Delta\eta = \frac{\partial\eta}{\partial x}\Delta x + \frac{\partial\eta}{\partial y}\Delta y + \frac{\partial\eta}{\partial z}\Delta z.$$

This can be separated into a dot product of terms to give,

$$\Delta\eta = \nabla\eta\cdot\Delta\mathbf{r}$$

$$= |\nabla\eta||\Delta\mathbf{r}|\cos\lambda.$$

However in our case since $\nabla\eta$ is along $\mathbf{r}$ the angle λ is zero.

From this $\frac{1}{|\nabla\eta|}$ equals,

$$\frac{1}{|\nabla\eta|} = \frac{|\Delta\mathbf{r}|}{\Delta\eta}.$$

Therefore our discrete approximation to ρ is,

$$\rho \approx \frac{\eta\,(\text{distance})}{(\eta_2-\eta_1)\sin\gamma}. \tag{2.26}$$

This is as far as the ray path equation can be examined in an independent fashion. The next step is to consider how it is applied in calculating ray paths.

Chapter 3

Ray Path Calculations

In the previous chapter the equation for the index of refraction and the differential equation for a light ray path were introduced. These equations along with associated equations from the atmosphere model provide the fundamental equations of ray path calculations.

Ray path calculation is really the continual application of a step by step procedure starting at the current ray path location and producing a new ray path location.

3.1 Ray Path from Observer to Outer Ellipse

The ray path starts at the observer location. This location reflects the eye level position of the observer standing on the earth with respect to the coordinate system.

The observer may stand either outside or inside the elliptical shells. The starting direction of the ray is given by a vertical and horizontal angle at the eye location. The vertical angle is corrected to include the vertical angle introduced by the observer's

location on the earth's surface. This correction is called the base angle.

$$\text{base\_angle} = \frac{\text{distance of observer from origin}}{\text{earth radius}}$$

This equation is just the definition of arc length along the circumference of a circle with respect to the radius of the circle and the angle covered by the arc.

To test if the observer is inside the atmosphere consider the elliptic equation given in Equation 2.20. If a point off the ellipse surface is used in this equation the left and right sides won't be equal. If that point is outside the ellipse the function will be greater than one and if the point is inside the ellipse the function will be less than one.

A generic function *ellip* can be defined as,

$$ellip(a, b, c, x, y, z) = \frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{(z - d_0 + \frac{x^2+y^2}{2R_E})^2}{c^2}.$$

When the values associated with the outer ellipse and the location of the observer are substituted then the value of *ellip* will indicate if the observer is inside or outside the ellipse. Since we are interested in testing for the outside condition we create a function to test for this, i.e.

$$outside(a, b, c, x, y, z) = Boolean^1 \left[\left(\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{(z - d_0 + \frac{x^2+y^2}{2R_E})^2}{c^2} \right) > 1.0 \right].$$

For an outside observer we need to determine the intersection of the straight line projection of the ray start direction and the outer ellipse. If this intersection does not exist then it is necessary to find the intersection with the object plane. You will

[1] Boolean is a binary function which will assign to the variable a true or false state based on the given test function.

recall (p. 32) that outside the atmosphere the temperature is constant which gives rise to straight line ray paths.

The complicated nature of the *ellip* function makes it difficult to find an analytic solution for the intersection of the ray path and outer ellipse. The alternative is to iteratively find the intersection. In an iterative approach we can again use the *outside* function. By successively stepping in the direction of the ellipses until a point is found which is inside (not outside) the atmosphere it is possible to locate the region of intersection. If no point tests inside the atmosphere and the ray path length is approaching the radius of the earth then it is time to set a flag indicating that there is no intersection. The comparison of the ray length to the earth's radius ensures that the ray passes through the entire model before terminating the search. Once the intersection is surrounded by two known points a bisector search can be applied to quickly locate the intersection.

In the bisector search the step size is divided by two and the ray point is moved back one step if the current point is inside the outer ellipse or forward one step if the current point is outside the outer ellipse. Then the step size is reduced again and the point is moved back if it's inside and forward if it's outside. This is repeated until the ray path point is on the ellipse. Once the ray point is on the ellipse the search is terminated.

In our case *on* means that the point is close enough to the ellipse that it exceeds the general level of accuracy of the other functions in the computer program. *On* is defined as,

$$on(a,b,c,x,y,z) = Boolean\left[\left(\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{(z-d_0+\frac{x^2+y^2}{2R_E})^2}{c^2}-1\right)<1.0e-5\right].$$

STEP	*DESCRIPTION*
Step 1.	Determine current ellipsoidal normal
Step 2.	Find ray tangent
Step 3.	Calculate new plane numbers
Step 4.	Find the angle gamma
Step 5.	Find the inter ellipse distance along the ellipse normal between current and next ellipses
Step 6.	Calculate ray path radius using Equation 2.27
Step 7.	Using above find new ray path centre

Table 3.1: **Steps of Sphere Procedure**

Once the intersection point is found it is assigned as the current ray path point.

3.2 Projecting the Ray Path

The elliptical shells form a series of isothermal surfaces in space. When a ray path enters the shells its intersection is found with the outer shell. This intersection location becomes the current ray path point. Inside the surfaces straight line projection is no longer valid. Inside, ray path projection using the full ray equation is necessary. Similarly when a ray path originates from an observer inside the atmosphere the full ray equation is necessary.

From the ray path equation discussion in Chapter 2 a procedure can be outlined to determine the ray path radius. Table 3.1 lists the steps involved.

This procedure will be called *Sphere*. The first six steps determine the ray path radius. The last step calculates the new centre for the ray path. The radius and centre define a spherical surface along which the ray point will be moved.

Consideration of the transit plane, radius, and centre define a circle in space over which the ray path will be projected. Starting at the current ray path point the ray

path is projected along the circle in the direction of the object plane. Initially the ray is projected some standard x distance toward the object plane. The x distance value is called *iterate*. By varying this value it is possible to optimize the number of calculations made along the ray path to give the degree of convergence desired for the ray path. Testing of various iteration step sizes also helps to examine the effects of step size on ray path solutions.

The function which does the actual ray path projection is called *move*.

The basic building blocks of all projections are the functions *sphere* and *move*. This kind of approach is basically the Euler method of solution for differential equations.

Sphere is designed to make as few calculations as possible. It makes use of values that were usually calculated in the previous call to *sphere*. Because this approach is used we must do several initial calculations once the ray path reaches the outer elliptical surface to provide the information required by *sphere*.

To use *sphere* the following ray path information must be found; the current transit plane numbers, the ray path centre, ray path location, current ellipse, next ellipse, and ray path radius.

At the current ray path location we know the ray path tangent since this is just the projection direction of the intersection vector or the starting ray direction at the observer's eye.

If the current ray path location is on the outer ellipse intersection then the ellipse through the ray point is known. The temperature of this shell defines the index of refraction at the current point according to Equation 2.18. This information can also

be found within the elliptical shells. First find an ellipse that fits through the current point. Next, linearly interpolate the temperature based on the shell height of the two surrounding shells to determine the temperature associated with the fitted ellipse. Then the index of refraction is found directly from this temperature.

To fit an ellipse through the current point first find the surrounding ellipses by starting with the *outside* ellipse and stepping inward until the point tests outside the current ellipse. The previous ellipse and current ellipse will surround the current point.

Set the new ellipse parameters to the current ellipse parameters. Let $shift$ equal the difference in c parameters between the surrounding ellipses.

Divide $shift$ by two. If the point is outside the new ellipse then add $shift$ to the new ellipse c parameter, otherwise subtract it. Using the formula for the foci of the ellipses $t = a^2 - c^2$, $t_2 = b^2 - c^2$ and the values of t, t_2 from the outer ellipse determine the values of b, a for the c value. Test if the point is *on* the new ellipse. If the test fails then repeat the above steps. Finally, linearly interpolate the temperature of the new ellipse.

The above procedure is called $fit\_ellipse$.

Only the tangent vector is known in the transit plane so far. However, we can find the ellipsoidal normal and use Equation 2.22 to determine $\mathbf{b}$. The current ellipse is known because it was found in order to determine the current index of refraction.

The inward pointing ellipsoidal normal is the negative gradient of *ellip* and is,

$$ellip\_norm(x, y, z) =$$

$$-\frac{2x}{a^2}-\frac{2\left(z-d_0+\frac{x^2+y^2}{2R_E}\right)}{c^2}\frac{x}{R_E},-\frac{2y}{b^2}-\frac{2\left(z-d_0+\frac{x^2+y^2}{2R_E}\right)}{c^2}\frac{y}{R_E},-\frac{2\left(z-d_0+\frac{x^2+y^2}{2R_E}\right)}{c^2}. \tag{3.1}$$

Substituting the current ray point will give $\mathbf{n_e}$. The function *plane_of_sight* finds the cross product given by Equation 2.22 and then substitutes the plane direction numbers and the current point into the plane equation to determine the remaining plane equation number. The plane equation is $Ax+By+Cz+D=0$. *Plane_of_sight* returns the plane numbers A, B, C, D.

The ray path radius associated with a straight line is $1.0 * 10^{12}$ meters. This number will give a ray arc which approximates a straight line.

In order to find the ray path center we need the ray path normal. The ray path normal is defined by $\mathbf{b} \times \mathbf{t} = \mathbf{n}$ for an inward pointing normal. The ray path center is determined by multiplying the normal vector by the scalar radius and adding this vector to the current location. The function *sphere_centre* performs these steps and returns the ray path center location.

The final piece of information needed by *sphere* is data on the next ellipse the ray path will hit. If the current point is from the outer ellipse intersection then the next ellipse is the first interior shell. For a ray path originating within the ellipses the process is slightly more complicated. How do we know which is the next ellipse? It could be the ellipse below or above the current ellipse.

We can tell if we know if the ray path is heading upward or downward. In order to keep track of which way the ray path is heading we will introduce a variable called *upward*. This variable is a flag. If the ray is heading upward then *upward* is true otherwise its value is false. For the outer ellipse intersection case *upward* would be

set to false.

For an interior originating ray, in order to set the *upward* flag we need to determine which way the ray path is heading. One way to do this is by projecting a fixed length in the ray start direction from the originating point. If this new point tests *outside* the current ellipse then the ray path is heading upward (*upward*=true) and the next ellipse is the first shell larger than the current shell. If the new point is not *outside* then the ray is heading downward (*upward*=false) and the next ellipse is the first ellipse smaller than the current ellipse.

Once all the initial information is calculated or after the last projection is completed the sphere function is called.

As mentioned earlier the first step is to determine the current elliptical normal. The function *ellip_norm* described earlier is used to determine this vector.

The second step is to determine the ray path tangent. This is done by a function called *tangent*. Using the ray path centre and the current path location a vector normal to the ray path is formed and normalized by dividing each component by the vectors magnitude. Forming the cross product $\mathbf{b} \times \mathbf{n}$ gives the tangent vector.

In step 3 the new transit plane or plane of sight are calculated. A function called *new_plane* passes the tangent and inward normal vectors to *plane_of_sight* which then forms the cross product $\mathbf{t} \times \mathbf{n}_e$ and substitutes the current path point to determine the transit plane numbers.

Step 4 consists of determining the angle gamma between the elliptical norm and tangent. The function *angle* does this by using the cosine law. Since both vectors are unit normal the values of a and c are equal to one. This leaves $b^2 = 2 - 2\cos\gamma$.

Assume the two vectors originate from the same location; then b is the distance from the tip of one vector to the tip of the other. Since the normal vector is the inward pointing normal and *gamma* is the angle between the positive normal and tangent, then b^2 is the sum of the squares of the sum of the corresponding vector's components.

From the cosine law $\cos\gamma = \frac{2-b^2}{2}$. Taking the arccos of this gives the required formula for gamma.

Next, in step 4, we determine the gradient distance between the current ellipse and the next ellipse. The function *line_intercept* is used to determine the intersection of a line starting at the current ray location in the inward normal direction with the next ellipse. Once the intercept is located it is then tested to determine if the new point is below the surface of the earth. If the point is below the earth's surface then a new intercept with the earth's surface needs to be found. The function *sphere_intercept* performs this procedure. *Sphere_intercept* solves a line - sphere intercept equation.

Using the parametric form of a line,

$$x = x_1 + ta, y = y_1 + tb, z = z_1 + tc$$

where x_1, y_1, z_1 is the current ray location and a, b, c is the line direction vector.

And,

$$x^2 + y^2 + (z - d_0 + R_E)^2 = \text{erad}^2$$

the equation for the earth's surface.

Substituting the line equation into the spherical equation and separating gives a quadratic equation in terms of the parameter t. Solving for t and substituting into the line equation yields the desired intercept location. Also, if the intercept is with

the surface, the next ellipse is temporarily set to the innermost ellipse so that the radius calculation uses the index of refraction most closely associated with the earth's surface.

Once a valid intercept is found then the magnitude of the vector from the ray path location to the intercept location is calculated and returned to *sphere*.

Step 6 uses the inter ellipse distance, index of refraction of the current and intercepted ellipse, and gamma to determine the *radius* based on Equation 2.27. The final step determines the ray path centre by using the function *sphere_centre* mentioned earlier.

Earlier, the function *move* was mentioned. It does the actual projection along the ray path arc. *Move* uses the ray path centre, transit plane, ray path radius, and the current ray location with modified x location. The modified x was produced by adding the iteration step to the old x location. This gives a simple way of moving along the arc path.

We have two equations, the first is the plane equation,

$$Ax + By + Cz + d = 0$$

and the second is the ray path sphere equation,

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = \text{radius}^2$$

These two equations allow us to eliminate one variable. If a third equation for the next ellipse where added a complete system of equations would be formed. However, so far an analytic solution for this set of equations has not been discovered. An alternative to finding the intersection point with the next ellipse is use of an iteration

step. By specifying one of the variables and eliminating another it becomes possible to determine a second variable. The eliminated variable is then determined from the other two. This is the approached used for *move*.

The plane equation was solved in terms of y and substituted into the sphere equation. The sphere equation then is solved as a quadratic in terms of z. The appropriate z solution and x are substituted in to the plane equation to give y. This gives the new ray path location along the ray path circle.

The basic ray path projection procedure involves updating the various ray parameters to reflect the ray path current location and then moving some distance along the ray path circle to a new location. The next section will refine the basic procedure and describe the various specific procedures required to take care of moving through the different ellipses.

3.3 Projection Cases

The first version of the ray projection software used the Euler method described in the previous section.

As various data sets where processed in order to check the correct functioning of the algorithm a strong dependence on step length showed up in the ray path. Table 3.2 lists the termination height of the ray path at the object plane against the step size used in the ray path projection.

In comparing the results of the various step sizes for the two listed versions of Euler projection method, the solutions in both cases showed more deviation than was considered acceptable. The older version of the Euler method had more errors in the

Programs: Step Size	PC(July)	Euler(July)	PC(April)	Euler(April)
3000m	3.936890	4.216431	3.900269	
1000m	3.916870	4.185425	3.921875	4.552856
500m	4.029053	4.013794	4.055664	3.535767
200m	4.019287	4.015137	4.054443	3.999634
100m	3.859375	3.906494	3.821655	3.972656
50m	4.004150	3.974121	3.902954	3.781128
Mean	3.96094	4.05190	3.94281	3.96841
STD	0.06745	0.12239	0.09350	0.37577

Table 3.2: **Projection Method Comparison**

algorithm which certainly is a contributor to the higher deviation level. However, even after further refinement the newer Euler algorithm still had high levels of deviation. To improve the solution accuracy a more sophisticated method of numerically solving differential equations was needed. The Predictor-Corrector(PC) method is another simple method for solving differential equations. The PC approach builds on the Euler method by using the initial projection to generate information that indicates the deviation in some known property between the locations. In a smooth function for example the derivative would be a continuous function. This information about the property is used to improve the initial projection.

In our case the path parameters at the new location looking back toward the initial location are calculated. Given a sufficiently small step we would expect a property like the path radius to be essentially constant. By averaging the path properties from each point looking toward the other point a better estimate of the overall path properties should be obtained than only using the path properties of the initial location.

Compare the results of the PC method against the Euler method for the two par-

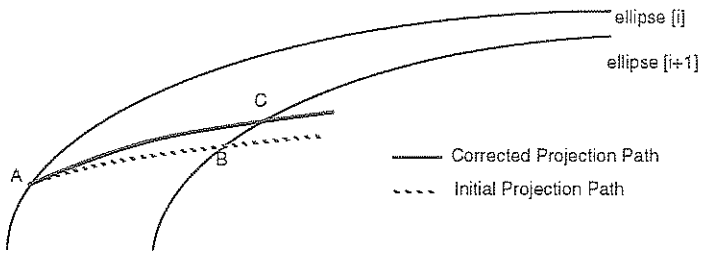


Figure 3.1: **Case 1 Ray Penetration**

allel versions of the program. In both cases the PC method gives improved stability in solution over the Euler method. Typically the PC method will give the same accuracy as the Euler method while using less computer time allowing quicker processing of the data.

The improved solution stability provided by the PC approach lead to its choice as the projection method in the final versions of the projection algorithm.

The use of PC led to the need to test for various special problems in the projection cases. These problems and the general procedure involved in the projection case will be discussed in the following sections.

While projecting along the ray path it is necessary to check for various cases. In each case some special actions are applied. This is necessary to accommodate travel through the elliptical layers, exiting the atmosphere shells, and entering the inner shell area.

3.3.1 Case 1: Penetrate Next Ellipse

When the ray path enters a new layer the current ellipse and next ellipse variables must be updated. Either the ray has penetrated the next ellipse or repenetrated the

current ellipse. There are separate procedures for each of these situations.

This procedure deals with entering the next layer. Each new layer means that the index of refraction is varying in a new way. Because of this we will move the ray path location to the point where it first intersects the new layer. In Figure 3.1 this is point B. This helps maintain ray path accuracy through the various layers.

The function *intercept* performs a series of checks to see of the current ray path arc intersects the given ellipse. If an intersection is found the function returns a true state.

Intercept begins by testing the current location. The *exor* of not *outside* and *upward* will give a true results if the point is outside and traveling upward or if the point is not outside and the direction is not upward. If the test is true intercept returns a true.

In the case of a false test the function does additional checking. This is to reduce the number of intersections missed due to the ray point passing into and out of the next layer in the same step. At most three additional *exor* tests will be made. The points tested in order are $\frac{1}{4}$ iteration back, $\frac{1}{2}$ iteration back, and $\frac{3}{4}$ iteration back. If at any of these locations the test is true the point location is updated to the current test location and *intercept* returns a true value. If none of the additional points test true then *intercept* returns false.

Once the intercept test determines that an intersection with the next ellipse exists the next step is to locate it.

Locate_isotherm is a function which uses a binary search to locate the isotherm intercept. The search continues until *on* tests true for the current path location and

the next ellipse. The search starts by dividing the current step size in half. If the point is not outside the ellipse and moving upward the point is moved back by $shift$. Otherwise the point is moved forward by $shift$.

Once the isotherm is located the *sphere* procedure is applied at the new path location and looking back toward the starting point. By averaging the new ray path parameters with the previous path parameters a new set is created which better reflects the path between the two points. This correction to the initial projection parameters improves the ray path accuracy. From the start location the path is reprojected.

One of two problems can arise at this point. The new path point may not intersect with the next ellipse anymore or the path might hit the ground before finding the ellipse. Both of these cases must be checked and dealt with separately.

The first step is to test for grounding. Any point below ground causes problems when passed for use in the various functions as well as not being physical realizable. A simple function *grounded* substitutes the current point into a test which checks the left side of the earth surface equation against the earth's radius. If the value tests less than the earth's radius a true value is returned.

For grounded points the correction is to find the ground intercept and then continue to the next check. *Find_ground* uses a binary search to move successively closer to the ground location. Once the $shift$ step size is less than .001 the search is terminated. The point is moved forward or back by a decreasing $shift$ step based on whether it is above or below ground on successive tests.

The next step is to again test for an *intercept*. If the test fails only the corrector

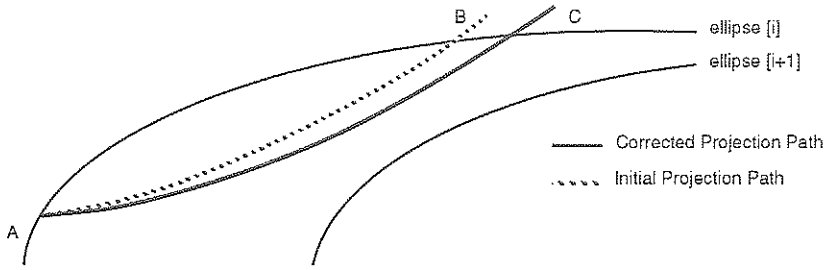


Figure 3.2: **Case 2 Ray Repenetration**

variables (path parameters) are updated to reflect the new path parameters. The procedure drops out of the current case and continues testing with the next main case test.

If *intercept* tests true then the new intersection is found using *locate_isotherm*. Figure 3.1 shows the new intersection location as point C. The current ellipse variable is updated to reflect the current shell location. Next an ellipse search is done to find the next ellipse above the current ellipse if the ray direction is upward or the next ellipse below if the direction is downward. The next ellipse variables are updated with this ellipse's data.

If the current ellipse is the innermost ellipse then the next ellipse is left the same as the current ellipse. Similarly if the current ellipse is the outer ellipse and the ray is heading upward then the next ellipse is left as the outer ellipse. Both of these conditions are handled by special cases during the next iteration.

3.3.2 Case 2: Repenetrate Current Ellipse

The second case deals with ray path curves that instead of hitting the next ellipse bend back to hit the current ellipse. Figure 3.2 shows an example repenetration

procedure. For this case the test is the *exor* of *outside* and *upward*. The consequences of missing an intersection in this case have less impact than for the previous case and are neglected.

If a repenetration is detected then the direction flag is reversed to indicate that the ray is now heading in the opposite direction through the layers. Next, the repenetration location, B, is found using *locate_isotherm*. The ray path parameters are updated by calling *sphere* while looking toward the start location. Then a function called *correction* averages the new and old path variables. The path is then reprojected.

Again several problems can arise after reprojection. The new point must be tested for grounding and moved to the surface if it is grounded. If the new path fails to penetrate the current ellipse then as an alternative *fit_ellip* is called. This creates an ellipse that is fitted through the current point. This new current ellipse along with original upward state are passed on to the next projection iteration.

If the new path does still penetrate the current ellipse then the new intersection, location, C, is located. Finally the current ellipse variable is updated and the next ellipse data is determined as in the previous case.

3.3.3 Case 3: Ray Does Not Penetrate

Examine Figure 3.3 which shows a case in which the ray path does not penetrate any known ellipse. If the current location, B, failed the last two tests then the location is intermediate between the current ellipse and the next ellipse. In this case an ellipse is fitted through B, the current location, using *fit_ellip*. Then the path parameters

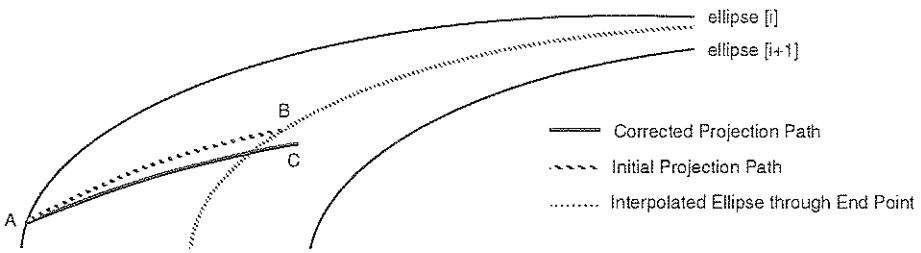


Figure 3.3: **Case 3 Ray Path**

looking back to the start location are found using *sphere*. *Correction* is applied to produce the new path parameters and the point is reprojected ending at C.

As in the other cases the point is tested for grounding and moved to the surface if necessary.

The special situations to check for include testing for penetration of the next ellipse and ensuring that the point does not exit the atmosphere. In the normal case a new ellipse is fitted through the current point.

For the penetration case the isotherm is located by *locate_isotherm*. The current and next ellipse are updated similarly as discussed in case 1.

If the point fails the normal and penetration tests this means that the point is outside the outer ellipse. In this case the intersection with the outer ellipse is located and both the current and next ellipse are set to the outer ellipse.

3.3.4 Case 4: Outside Last Ellipse

The final two cases check the extremes of the atmosphere shells. This case tests for the current ellipse equal to the outer ellipse and *upward* true. This means that the ray has reached its exit point from the atmospheric shells.

During the last call to sphere the path radius will have been set to the value for a straight line reflecting the constant index of refraction in the outer atmosphere.

The ray point is repeatedly moved forward by *iterate* until the point either hits ground or the object plane. This case stops after it detects one of these conditions and lets the end point test finish the ray path.

The *Hit_object_plane* function simply tests if the x variable at the current location is greater than the object plane location. The function returns a true or false indicating the state produced by the test.

3.3.5 Case 5: Inside All Ellipses

The inside case is indicated by the current ellipse being set to the innermost ellipse and not *upward*.

Similarly to Case 3.3.4 the path location is moved forward using the previous path radius and centre until one of three conditions tests true.

The simple conditions that are tested are for grounding and for hitting the object plane. In this case it is also necessary to test if the point repenetrates the current ellipse. To check for repenetration the *outside* function is checked for a true state and the value of the path location is tested for a positive value. This test will only be true for a ray path penetrating upward out of the inner ellipse.

In case of repenetration *upward* is set to true and the intersection with the inner ellipse is located. Finally, next ellipse is set to the next higher ellipse.

3.4 End Point Test

At the end of every projection cycle the resulting end location is tested for two termination conditions.

The first termination condition is a test to check if the ray point has penetrated past the object plane. The other termination test checks if the ray is at or below the earth's surface.

If a point makes it past both these tests then the ray advance procedure is executed in preparation for the projection cycle to repeat.

3.4.1 Object Plane Test

Once a point tests true in *hit_object_plane* we know that the current ray path has penetrated the object plane. The ray path has reached its final destination. To terminate the projection cycle a status variable called *finished* is set to true. To move the point to its exact point in the object plane is quite simple. The x variable is set to the value of the x parameter of the object plane. Then calling *move* will move the ray path point to the desired position along the ray path at the object plane.

3.4.2 Grounded Test

Similar to the previous case after testing a grounded condition using *grounded* the finished flag is set to true. Next the *find_ground* function is called to move the point to the surface intersection.

3.5 Ray Advance

In preparation for the next cycle of the projection process an initial projection must be made. The *sphere* function updates the ray path parameters based on the current end location. These parameters are stored for use in the *correction* procedure and the predictor-corrector process. Next the point is projected by a call to *move*. The new point is tested for violation of the grounding condition and moved to the surface if it is violated.

The exception to this procedure is when the ray is inside the innermost shell. In this case there is no next inner shell to update the ray path parameters with. The inital project procedure is skipped and Case 5 takes over using the previous ray path data as an approximation to the inner shell conditions.

At the end of this case the program flow returns to the projection case tests.

Chapter 4

The 3d_ray Algorithm

The language chosen to implement this thesis was C. C was selected for its simplicity and versatility as a programming language. C's other virtue is the ease with which it can be ported to other platforms. As a result initial development was done using a 80286 Personal Computer and latter completed in parallel on the 80286 machine as well as on a SUN workstation.

Although there are some architectural considerations in porting to the SUN system the changes were fairly simple.

C code tends to be naturally readable. Most of the code contains extra comments to clarify what the code does. As a result very little time will be spent explaining the details of the algorithm. Instead Appendices A, B, and C include the source code of the program as it is implemented under Microsoft Quick C Version 2.0 for an 80286 computer.

Name	Value	Description
erad	6.37e6m	earth's radius in meters
kelvin	273.15	conversion factor from celsuis to kelvin
eyel	1.8m	assumed height of eyes of observer above ground level
minutes	3437.7467...	number of minutes per radian
pi	3.1415...	value of constant pi
SIZE	60	maximum number of elliptical shells allowed
TRUE	1	define a boolean True
FALSE	0	define a boolean False
LARGE	1.0e12m	radius to give a straight line

Table 4.1: **Constants**

4.1 Program Design

The program is divided into three sections according to function. The first section, in Appendix A, contains all the mainline instructions. The second section, in Appendix B, contains all the ray projection procedures called by the mainline program. The final section, in Appendix C, contains procedures used in the Personal Computer version to plot actual ray paths with the aid of Halo '88 graphics package by Media Cybernetics.

4.1.1 Mainline

The components of the mainline file are definition of constants, definition of structures, definition of new types, function definitions, and main program.

Table 4.1 is a list of the constants used in 3d_ray and what they represent.

In addition to the above constant variables three macro functions are defined which implement widely used functions. The first is the index of refraction equation with P=101.325 kPa or 1013.25 millibars. Substituting this value in the index of

refraction equation in Chapter 2 gives the equation $\eta\{T\} = 1 + 0.0799/T$. The second is a squaring function and the final function is a square root function.

There are three structures defined which are the basis for new variable types. The first structure, vector, is a double precision triplet $\{x, y, z\}$ used to store cartesian coordinate locations. The second structure, ellipsiodata, stores the double precision triplet of the ellipsoid parameters and associated double precision shell temperature value. The third structure, planenum, contains the double precision values of the plane equation constants.

The following four new variable types were defined. The structure vector is defined as type vec. The structure ellipsoidata is assigned to type elp. The structure planenum is defined to be type plane. An additional type called bool is defined as a subset of the integer type. Bool will only be used for variables with logic or boolean values.

The function definition section prototypes the procedure type, name, and passed variable types. These definitions help C to track if the right type variables are passed to and returned from the given procedure during compilation. The actual functions will be described later on.

Table 4.2 lists the name, type, and purpose of all variables used in the mainline.

To facilitate program development and verification two special sets of processes were included in the mainline. The first process kept track of each corrected projection point. By displaying or printing this data it was possible to examine the ray paths in detail. Another way of examining the ray paths is to display the rays in three dimensional space. A special procedure was developed which projected vertical groups

Name	Type	Purpose
out	FILE	Pointer to current output file
ellip[]	elp	Array of ellipsoid shell data and temperatures
nexelp	elp	Vector containing next shell expected along ray path
curelp	elp	Vector containing the current shell
interelp	elp	Vector for an intermediate fit ellipse
obs	vec	Observer location
map[]	vec	Actual and observer ray end locations
destn	vec	Point the eye believes it is seeing
point	vec	Current location on ray path
oldpt	vec	Storage of projection start point
line	vec	Direction numbers for ray path line
centre	vec	Centre of sphere on which ray travels
norm	vec	Normal direction numbers
tan	vec	Tangent to ellipse at point
oldtan	vec	Tangent to ellipse at oldpt
rays	vec	A far pointer to the array storing path points
sight[]	double	Array storing the double precision sight angles
obj	double	Object plane location on x axis
base_angle	double	Sighting angle due to earth curvature
radius	double	Radius of ray path arc
oldradius	double	Radius associated with initial projection
shift	double	Stepping variable for local intercepts
hor	double	Horizontal angle loop variable
ver	double	Vertical group loop variable
iterate	double	Iteration step size
origin_depth	double	Distance to earth surface at origin
finished	bool	Ray path termination flag
upward	bool	Ray direction indication flag
ellipnum	int	Number of shells in atmosphere
c	int	Counter
index	int	Counter
lastindex	int	Count for last point of previous pair
NumberOfRay	int	Ray path counter
NumPoint[]	int	Count of points in each ray
plotf	int	Flag to invoke 3d plotting call
print	int	Flag to invoke ray point printing
plofs	plane	Plane of sight data
oldplos	plane	Plane of sight data associated with initial projection
title[]	char	Space to store title for 3d plots

Table 4.2: **Variables**

of rays on to a two dimensional plot on the computer display. The display procedure is discussed in detail in the 3d_plot section.

Because each point uses 24 bytes a maximum of about 3500 points can be stored on the average computer. The combination of step size and number of rays per group must be adjusted so that the total number of points is less than the maximum. There is no formula to determine how many points a certain step size will generate so it is not possible to check this condition. Extreme caution has to be exercised not to exceed the memory capacity since unpredictable events will occur otherwise. Tracking ray paths is valuable in the study of the atmospheric model but the memory limitation is a serious problem when considering the whole atmosphere at once. As a result the program was modified to make the print and 3-D display optional.

There are four choices. They are print data, plot data, plot and print data, or do neither. Only in the last case is the ray point array not defined so that it is possible to evaluate as many ray paths as desired. No storage of any ray data occurs so there is no limit to the number of ray path evaluations. The ray path start and end location are printed to the standard output file to record the results. These options allow two levels of study for the model.

There is a flow chart for the mainline in Figure 4.1. Examination of this flow chart shows how the ray path projection procedure discussed in Chapter 3 is used repeatedly for the various angles of interest.

After each ray projection is completed the original destination projection along with the actual end point determined from the projection process is sent to an output file. The output file also has a copy of the initialization data and optionally the

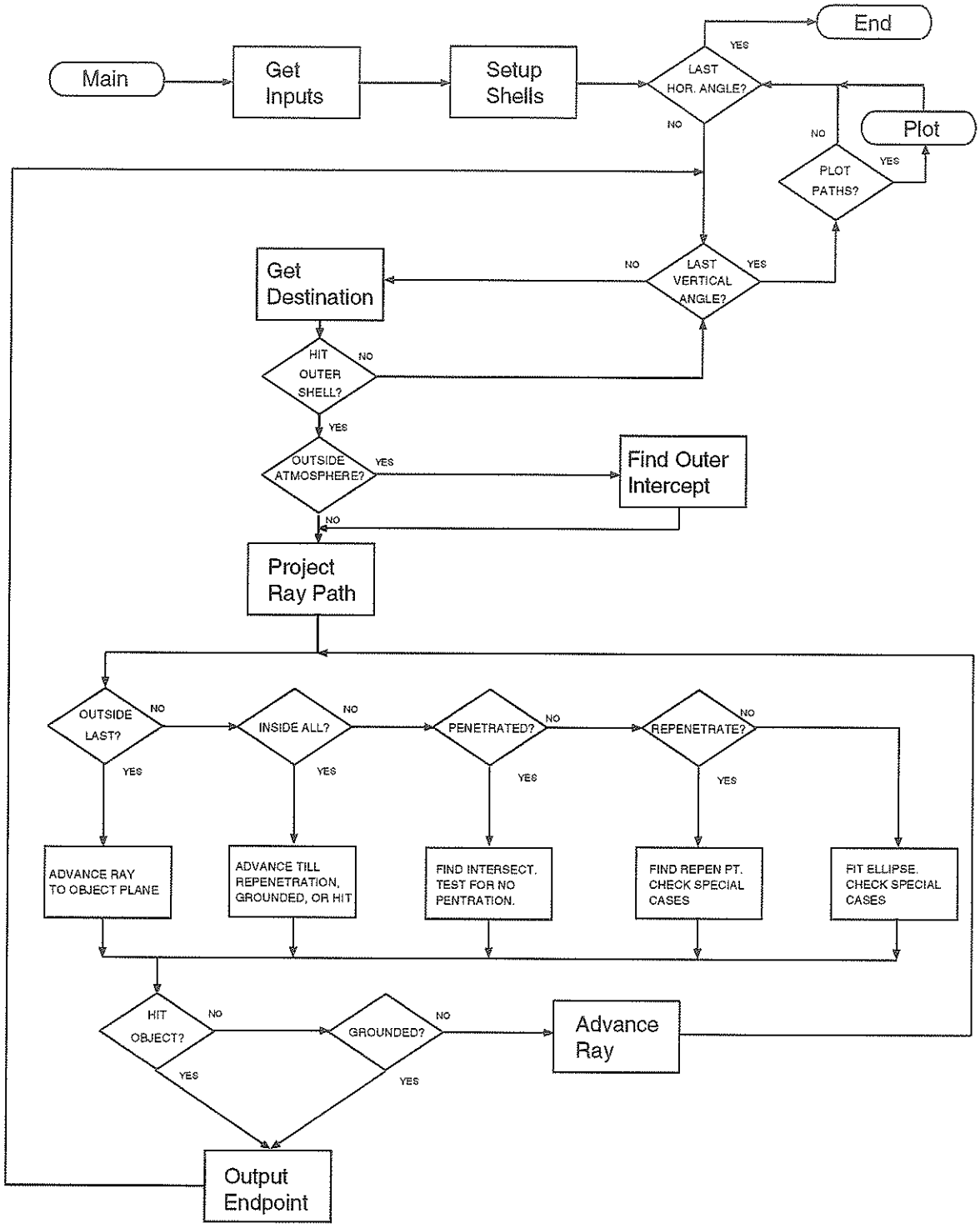


Figure 4.1: **Mainline Flow Chart**

ray path data. The output file is a record of the last 3d_ray run. The stored ray path endpoint data is with respect to the earth's surface. This means the point was converted from the model coordinate system back to an earth surface location.

The data input procedure is outlined in Appendix D along with an example data file. Once all the data is collected by *inputdata* it must be interpreted. The function *setupshells* uses the height and temperature profile data and the exterior shell data to set up the elliptical shells in the ellip array. The procedure *conv_coord* is used to convert the observer and object location on the surface of the earth to locations in the model's coordinate system. It also converts the sighting angles from minutes to radians. Further, the depth of the origin of the coordinate system below the earth at the origin is calculated and the base angle of the observers position is determined.

At the beginning of each ray projection the procedure *get_destn* is called. *get_destn* uses the starting direction of the ray path at the observer and finds the straight line intercept with the object plane. If the ray path grounds during processing then the destination is updated to a point which intersects with the x plane at the grounding point.

The destination point represents where the eye sees the point at the end of the projected ray path.

4.1.2 Subroutines

The 3d_subro.c file contains all mainline procedures except those related to the 3d ray plotting program.

Appendix B contains a copy of the C file. The procedures are well documented and

list input and output data. They should be self explanatory given the descriptions made earlier.

4.2 3d Plotting

As mentioned previously the Halo '88 graphics library by Media Cybernetics was used to display the ray path plots. Halo '88 is an implementation of the Graphic Kernel System (GKS) standard. GKS is a system of communication between an application program and graphics devices. This GKS is a purely 2-D system.

Halo '88 supports five types of devices: graphics devices, locators, printers, plotters, and scanners. There are three coordinate systems available: device coordinate system, world coordinate system, and normalized coordinate system. Over 190 different functions are implemented which control everything from text size and orientation to shape generation.

It is the flexibility of this GKS which led to its acceptance for this project. The other available 3-D software were stand alone programs with limited configurability.

The use of Halo '88 gives rise to the need for a procedure to project the 3-D data on to a 2-D plane for device display.

To project 3-D data on to a 2-D plane is a two step process. The first step is to transform standard (x,y,z) coordinates to eye coordinates. The second step involves determining the screen coordinates from the eye coordinates.

The 2-D view is always dependent on the location of the observer. In our system the eye location is defined by the spherical coordinates (ρ, ϕ, θ).

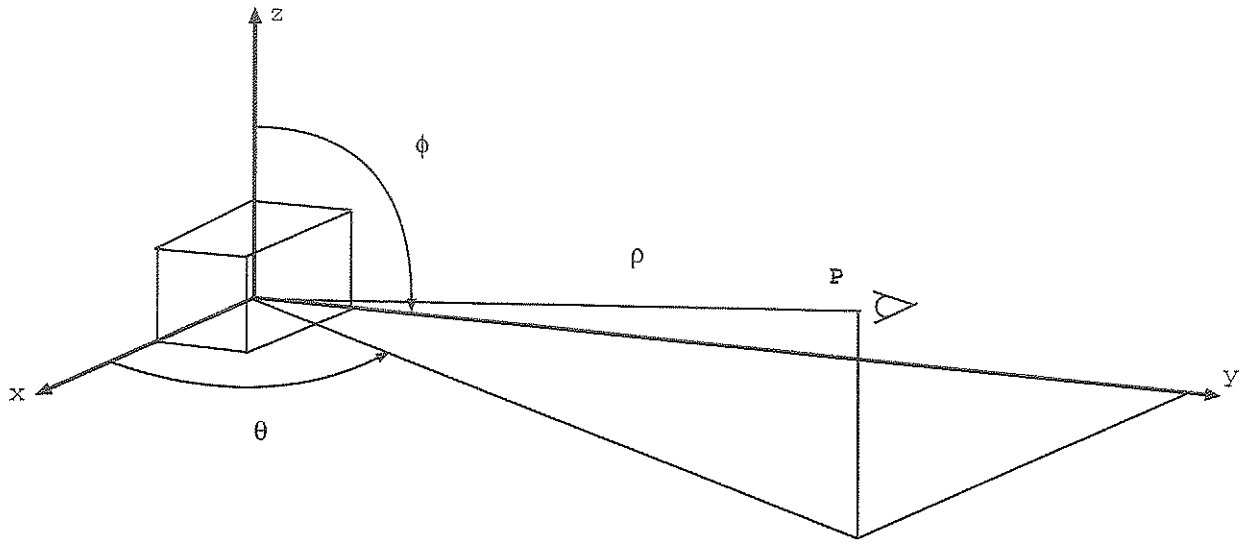


Figure 4.2: **Observer's View Point**

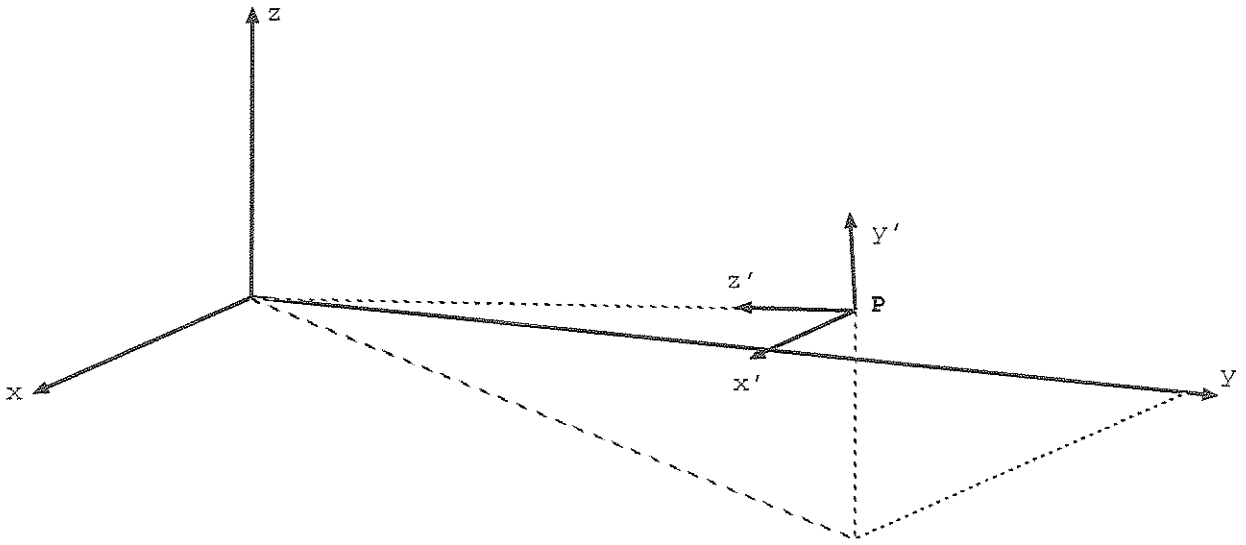


Figure 4.3: **New Coordinate System**

As Figure 4.2 shows the observer's view point is at a distance ρ from the origin, and at angle ϕ from the z axis and θ from the x axis. By adjusting these three quantities the object can be seen from any side and at any distance.

Since the location of observer affects the view it makes sense to make this the new center of the coordinate system. More importantly the use of $\mathbb{P}$ as the new center of the coordinate system simplifies the procedure for determining the 2-D plane coordinates.

The orientation of the new coordinate system is determined by the orientation of the coordinates of the 2-D view. In our case the x axis is to the left and the y axis points upward. By choice the z axis for the observer points away from the observer toward the old coordinate system. This selection leads to a right hand coordinate system. Figure 4.3 shows the new coordinate system.

There are three steps involved in the transformation from object to eye coordinates.

To move the origin to $\mathbb{P}$, the eye's location, the individual components of the vector must be subtracted from the corresponding standard coordinates. This operation can be obtained from the following transformation matrix.

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\rho\sin\phi\cos\theta & -\rho\sin\phi\sin\theta & -\rho\cos\phi & 1 \end{pmatrix}$$

$(\rho\sin\phi\cos\theta, \rho\sin\phi\sin\theta, \rho\cos\phi)$ are the (x, y, z) coordinates of $\mathbb{P}$.

The next two steps accomplish the reorientation of the coordinate axis. The object is to have the new z' axis intersect the old origin.

The first transformation matrix,

$$B = \begin{pmatrix} \sin\theta & \cos\theta & 0 & 0 \\ -\cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

will rotate clockwise about the z' axis such that the negative y' axis will intersect the z axis. This angle is $(90^0 - \theta)$.

The second matrix,

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -\cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & -\cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

will rotate about the x' axis in the counter clockwise direction by $(180^0 - \phi)$. This will align the z' axis so that it points to the original origin.

By multiplying ABC a composite matrix is created which performs the transformation in one step. $T = ABC$.

$$T = \left(A \right) \left(B \right) \left(C \right)$$

$$T = \begin{pmatrix} \sin\theta & \cos\theta & 0 & 0 \\ -\cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\rho\sin\phi & -\rho\cos\phi & 1 \end{pmatrix} \left(C \right)$$

$$T = \begin{pmatrix} \sin\theta & -\cos\theta\cos\phi & -\cos\theta\sin\phi & 0 \\ -\cos\theta & -\sin\theta\cos\phi & -\sin\theta\sin\phi & 0 \\ 0 & \sin\phi & -\cos\phi & 0 \\ 0 & 0 & \rho & 1 \end{pmatrix}$$

Thus (Xe, Ye, Ze, 1) = (X, Y, Z)(T)

Now that the data are transformed to eye coordinates they must be projected to screen coordinates.

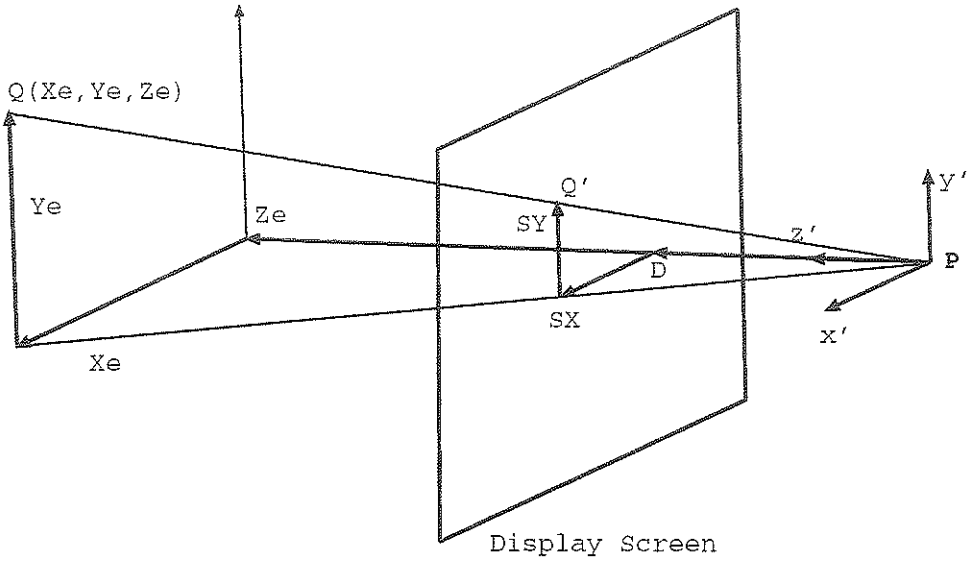


Figure 4.4: **Display Coordinates**

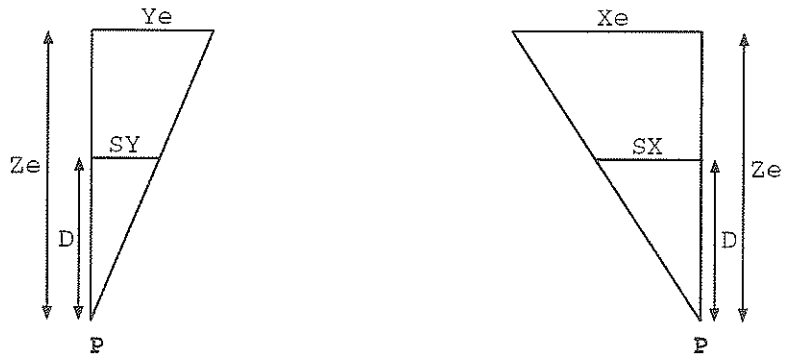


Figure 4.5: **Similar Triangles**

In Figure 4.4 the screen is placed at a distance D from the origin. Point Q has coordinates (Xe, Ye, Ze). Similarly, where Q' would appear in screen coordinates is given coordinates (SX, SY, D). By similar triangles $\frac{SY}{D} = \frac{Ye}{Ze}$.

or $SY = D(\frac{Ye}{Ze})$

Similarly $\frac{SX}{D} = \frac{Xe}{Ze}$ or $SX = D(\frac{Xe}{Ze})$. Figure 4.5 shows the similar triangles for the two planes of interest. For a more detailed description refer to Meyers.[25]

The combination of the eye coordinate transformation and screen coordinate projection complete the 3-D to 2-D projection necessary to prepare the data for display.

The overall procedure for implementing the 3D plot is contained in *plot*. The first step is to convert the data to 2D data. This is accomplished by the *Convert* procedure which implements the transformation described earlier. Once the data is transformed we are ready to create the display view. The display view is started by writing the title over the graph area.

Next, the positive extreme for the horizontal variable is found. Most of the atmospheres which will be considered will be concerned with heights less than a hundred meters versus atmosphere lengths of multiple kilometers. It is for this reason that the vertical scale is calculated from the horizontal scale. This calculation takes into account the aspect ratio of the display and the need to display the image in its real proportions.

Once the vertical and horizontal ranges are known then the plot border and coordinate axes are added to the display. Each axis is foreshortened according to the viewing angle to give some size reference to the display. All the axes have the same numerical length.

Finally it is time to plot the curves that represent the plotted curves. Each ray path is plotted individually. This is done using the Halo' 88 function POLYCABS which fits a smooth curve through the points of the ray path. With the addition of the current view statistics and menu options the plot is complete.

Plot provides for the menu functions by testing for key strokes once the plot is displayed. Based on the menu choices various plot parameters can be changed to alter the view displayed. Each time a parameter is changed the display process restarts at the beginning of the plot function.

The basic plot parameters that can be changed include: d, the distance to the viewing plane; ρ, the distance to the original origin; ϕ; θ; x-axis limit; y-axis limit. By changing either axis limit the aspect ratio is changed from the real ratio and details hidden by long flat curves can be resolved. A function is included which rescales the y axis so that the x and y axis again form the actual aspect ratio for the plot.

Also a function is included in the plot menu which prints a copy of the plot via the printer support functions provided by Halo' 88.

The menu choices are:

Q=QUIT P=PRN SCRN A=ASPECT SCALE R=ROTATE OBJT S=SCALE V=VIEW SIZE

- Q will terminate the plot display and return control to the main program.
- P provides for the plot to be printed.
- A will recalculate the y axis scale based on the current x axis scale to give the true image shape.

- R allows the two viewing angles ϕ and θ to be changed.
- S allows the x and y axis scales to be set.
- V allows d and ρ to be changed.

Appendix C includes the source code for the *plot* procedure. For more information on the Halo' 88 functions see the Halo' 88 manual.

4.3 Verifying the Software

In the two previous sections the main program and ray plotting programs were discussed. The functionality and validity of these programs was tested using several methods. In total four tests where devised.

Of these the simplest test performs checks to see if a constant temperature profile produces straight ray paths through the model. Another simple test uses the left & right symmetry of the model when an observer stands on the x axis. For an observer on the x axis the left and right ray path groups should be mirror images of each other.

A more sophisticated test can be devised to check certain ray paths. Imagine a ray that starts and stops at equal distance and height from the center of the atmosphere model. If its points are in the y=0 plane and opposite each other with respect to the center of the model then the points of the path should be symmetric. Further, the maximum of the path should occur over the model center. By assessing the deviation in maximum and in symmetric points it is possible to draw conclusions about the validity of the model.

a=5000m b=1000m obs at (5000,0,1.8)m object at 2500m	
Height	Temp
30m	8.0 °C
22m	8.0 °C
21m	8.0 °C
20m	8.0 °C
15m	8.0 °C
11m	8.0 °C
0m	8.0 °C

Table 4.3: **Constant Temperature Profile**

Other models which have been validated can also form a standard to which the new model can be compared. By selecting a situation which can be emulated by both models and comparing the simulation results it is possible to assess the new model.

Some of these tests will indirectly depend on factors which are not being tested. In this case the objective is to perform the test in such away as to minimize these factors or recognize that the results are valid within limiting factors. As with all software it is impossible to check all possible situations. It is therefore important to devise careful programs of study which use the model against itself and other sources before placing faith in its results.

The first test was run using an atmosphere based on the data given in Table 4.3. The a and b parameters are the major and minor axis lengths of the ellipsoid. The observer stands at 5000 meters from the origin and the object plane is 2500 meters on the other side of the origin. In total 7500 meters are between the observer and the object plane.

The range of angles examined was -4 arc minutes to 40 arc minutes in 4 arc

Step	Z Dev mean	Std
50	0.0039320	0.002690500
100	0.0018503	0.001022750
500	0.0006480	0.000329806
1000	0.0005030	0.000434436

Table 4.4: **Ray Path Deviation Results**

minute increments in the vertical angle and at lateral angles 120 , 0, -120 arc minutes. Iterations were run with step sizes 1000m, 500m, 100m, and 50m.

The results are shown in Table 4.4. The second column represents the average difference between the z height of a straight line and the calculated height over all the rays consider in each iteration. The third column represents the spread of the deviations among the different ray paths in an iteration. From this table we can tell that the program produced ray path endpoints accurate to two decimal points. Similar analysis could be applied to each point generated in each ray path however simple examination of these points showed that they formed a straight line.

For the constant temperature profile data set the left/right symmetry is dead on to the six decimal points printed. Table 4.5 contains the output data for Data Set 2 which has a more complicated temperature profile. Data Set 2 is given in Appendix D. The step size used was 100m and the output displays vertical angles from 0 arc minutes to 40 arc minutes in 4 arc minute increments. Compare the path endpoints (x,y,z) of a ray in the positive angle group, 60 arc mins, with the mirror ray in the negative angle group. The values are mirror images with respect to the x-z plane. This point for point symmetry is matched along the ray path at each calculated point.

The symmetric ray test was done using Data Set 2 with the observer placed at (5300,0,1.8)m and the object plane at -5300m. The objective was to find a horizontal

Transfer Characteristic coordinates with respect to earth's surface.

apparent versus actual location					
x	y	z	x	y	z
Hor Sight Angle=120.000000 (min)					
+1715.02193	+114.71398	+2.649628	+1715.02193	+114.71142	-0.000001
-2499.99994	+261.90625	+14.956183	-2499.99994	+261.89815	+6.130396
-2499.99994	+261.90702	+23.688204	-2499.99994	+261.89505	+14.263588
-2499.99994	+261.90814	+32.420289	-2499.99994	+261.87133	+17.031823
-2499.99994	+261.90963	+41.152461	-2499.99994	+261.83371	+13.624424
-2499.99994	+261.91146	+49.884744	-2499.99994	+261.81321	+10.841589
-2499.99994	+261.91365	+58.617161	-2499.99994	+261.80151	+8.898144
-2499.99994	+261.91619	+67.349735	-2499.99994	+261.85136	+32.190797
-2499.99994	+261.91909	+76.082491	-2499.99994	+261.86770	+44.456383
-2499.99994	+261.92235	+84.815452	-2499.99994	+261.87859	+55.294828
-2499.99994	+261.92595	+93.548642	-2499.99994	+261.88656	+65.153337
Hor Sight Angle=60.000000 (min)					
+1708.08421	+57.46061	+2.651258	+1708.08421	+57.45933	-0.000001
-2499.99994	+130.91323	+14.945462	-2499.99994	+130.90921	+6.154891
-2499.99994	+130.91361	+23.673494	-2499.99994	+130.90786	+14.368440
-2499.99994	+130.91417	+32.401589	-2499.99994	+130.89771	+17.743454
-2499.99994	+130.91491	+41.129771	-2499.99994	+130.87865	+14.402802
-2499.99994	+130.91583	+49.858063	-2499.99994	+130.86791	+11.497449
-2499.99994	+130.91692	+58.586489	-2499.99994	+130.86172	+9.425399
-2499.99994	+130.91820	+67.315074	-2499.99994	+130.88532	+31.780306
-2499.99994	+130.91965	+76.043839	-2499.99994	+130.89381	+44.282503
-2499.99994	+130.92127	+84.772810	-2499.99994	+130.89930	+55.153767
-2499.99994	+130.92308	+93.502009	-2499.99994	+130.90336	+65.081407
Hor Sight Angle=-60.000000 (min)					
+1708.08421	-57.46061	+2.651258	+1708.08421	-57.45933	-0.000001
-2499.99994	-130.91323	+14.945462	-2499.99994	-130.90921	+6.154891
-2499.99994	-130.91361	+23.673494	-2499.99994	-130.90786	+14.368440
-2499.99994	-130.91417	+32.401589	-2499.99994	-130.89771	+17.743454
-2499.99994	-130.91491	+41.129771	-2499.99994	-130.87865	+14.402802
-2499.99994	-130.91583	+49.858063	-2499.99994	-130.86791	+11.497449
-2499.99994	-130.91692	+58.586489	-2499.99994	-130.86172	+9.425399
-2499.99994	-130.91820	+67.315074	-2499.99994	-130.88532	+31.780306
-2499.99994	-130.91965	+76.043839	-2499.99994	-130.89381	+44.282503
-2499.99994	-130.92127	+84.772810	-2499.99994	-130.89930	+55.153767
-2499.99994	-130.92308	+93.502009	-2499.99994	-130.90336	+65.081407
Hor Sight Angle=-120.000000 (min)					
+1715.02193	-114.71398	+2.649628	+1715.02193	-114.71142	-0.000001
-2499.99994	-261.90625	+14.956183	-2499.99994	-261.89815	+6.130396
-2499.99994	-261.90702	+23.688204	-2499.99994	-261.89505	+14.263588
-2499.99994	-261.90814	+32.420289	-2499.99994	-261.87133	+17.031823
-2499.99994	-261.90963	+41.152461	-2499.99994	-261.83371	+13.624424
-2499.99994	-261.91146	+49.884744	-2499.99994	-261.81321	+10.841589
-2499.99994	-261.91365	+58.617161	-2499.99994	-261.80151	+8.898144
-2499.99994	-261.91619	+67.349735	-2499.99994	-261.85136	+32.190797
-2499.99994	-261.91909	+76.082491	-2499.99994	-261.86770	+44.456383
-2499.99994	-261.92235	+84.815452	-2499.99994	-261.87859	+55.294828
-2499.99994	-261.92595	+93.548642	-2499.99994	-261.88656	+65.153337

Table 4.5: **Path Endpoints for Second Symmetry Run**

X	Z left	Z right	Z difference
5350	1.79999	1.80875	0.00876
5000	2.00993	2.01763	0.00770
4000	2.15752	2.15956	0.00204
3000	2.16823	2.17823	0.00081
2000	2.16823	2.16833	0.00010
1000	2.15553	2.15537	0.00016
Zmax is at 13.36m			
Horizontal Angle=1.96 arc minutes Step Size=500m			

X	Z left	Z right	Z difference
5350	1.79999	1.80363	0.00364
5000	2.00932	2.01224	0.00292
4000	2.14930	2.14990	0.00060
3000	2.16742	2.16718	0.00024
2000	2.15737	2.15678	0.00059
1000	2.14416	2.14367	0.00049
Zmax is at -28.15m			
Horizontal Angle=1.96 arc minutes Step Size=250			

X	Z left	Z right	Z difference
5350	1.79999	1.80375	0.00376
5000	2.00932	2.01221	0.00289
4000	2.14706	2.14827	0.00121
3000	2.16483	2.16505	0.00022
2000	2.15469	2.15440	0.00029
1000	2.14148	2.14113	0.00035
Zmax is at -29.46m			
Horizontal Angle=1.96 arc minutes Step Size=100m			

X	Z left	Z right	Z difference
5350	1.79999	1.80009	0.00091
5000	2.00932	2.00862	0.00071
4000	2.14672	2.14504	0.00168
3000	2.16443	2.16246	0.00197
2000	2.15426	2.15250	0.00176
1000	2.14102	2.13993	0.00109
Zmax is at -82.64m			
Horizontal Angle=1.966 arc minutes Step Size=50m			

Table 4.6: **Symmetry Test Results**

angle at which the ray path ended at (-5300,0,1.8)m. The test was run with 500m, 250m, 100m, and 50m step sizes.

Examine the results of the symmetry test as given in Table 4.6. X measures the distance from the origin. Z left is the Z height for positive X values and Z right is the Z height for negative X values. Over the 10,600 meters that the ray was tracked the begin and end points are less than a centimeter different. Overall all the results indicate an excellent degree of symmetry. In addition the path peak (zmax) occurred within one percent of the exact center of the path when compared to overall path length.

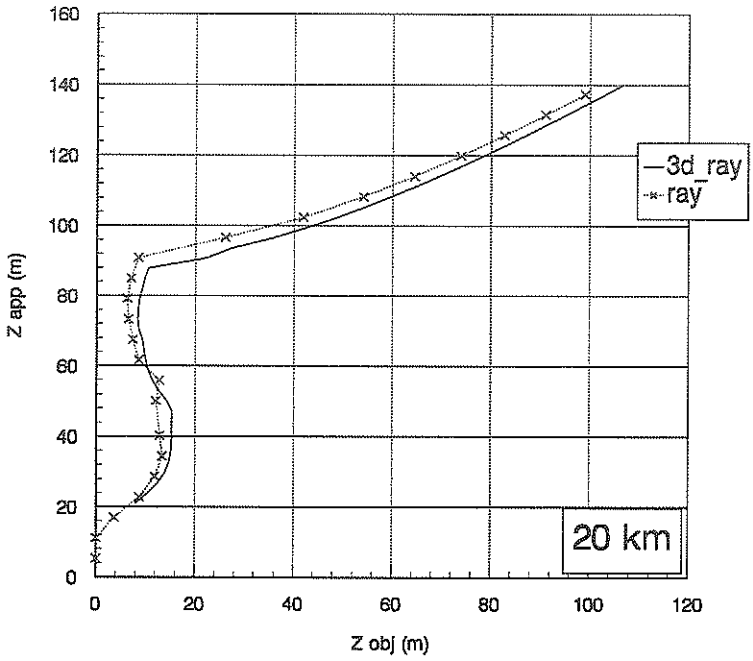


Figure 4.6: Transfer Characteristic

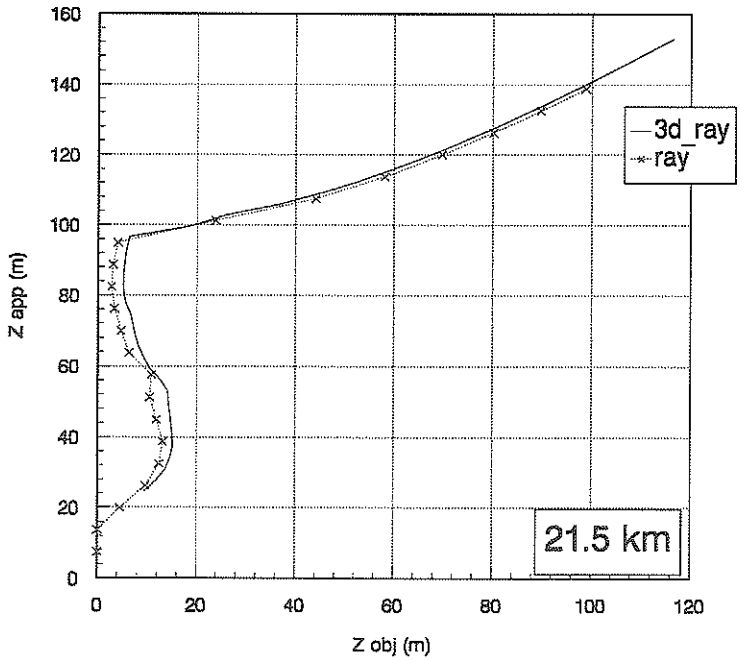


Figure 4.7: Transfer Characteristic

Figure 4.6 shows a comparison between the output of the *3d_ray* tracing program and current version of the *ray* program (2d). The data from the *ray* program was adjusted to account for a difference in reference systems and to compensate for pressure effects included in its algorithm. In *ray* all the Z object heights are with respect to the observer's location while in *3d_ray* the heights are with respect to the object plane location. This results in a difference in Z height equal to the difference between the height of the earth at the origin and the height of the earth at the object plane. The *ray* program also included pressure effects in its equations.

Pressure introduces additional curvature in the ray paths. In order to remove the pressure effects from the ray results an additional data run was done with constant temperature to determine the displacement introduced by pressure variations. From the pressure data it was determined that at 20 km pressure variations introduced an additional 6.77 meter upward shift in the Z object heights. Similarly at 21.5 km pressure variations introduced an additional 7.83 meter upward shift in the Z object height.

In both Figure 4.6 and Figure 4.7 the Data Set used was SETJM3 which is included in Appendix D. Using both *3d_ray* and *ray* the data was run with the object plane at 20 km and 21.5 km respectively. After compensating the *ray* data both data sets were plotted. Plotting the Z apparent height (where the eye sees something) versus the Z object height (what the eye sees) gave the transfer characteristic for each data set. In both Figures it is easy to tell that the two curves are very similar and match quite well. It is quite reasonable to suggest that the observable differences between the curves are likely the results of difference in mechanics between the two programs.

Overall the results of the four tests have proven the functionality of the programs and validated the data produced by the program.

4.4 Using The Output

The output from 3d_ray includes the output record file, and hardcopies of the 3d plots. The output record file contains the ray end point mapping between where the ray path actual terminated, the object view, and the apparent end point seen by the mind. The output file may also contain a ray path point record for each ray traced.

These outputs provide the basis for systematic study of the model on an individual ray or ray bundle basis.

Our interest in this study is the ray bundle information. In the following chapters this information on the mapping between the object view and the apparent view will be applied to images of the object to produce a simulated image of the apparent view. The end goal is to examine these apparent images to assess how the model has affected (distorted) the view that the observer sees.

Chapter 5

Mirage Simulation

The previous two chapters discussed the ray tracing procedure and implemented the process through computer simulation. The process traces light ray paths from their eye entry location backwards to their origination point. By comparing this point with the point perceived as the origin a relationship is developed between the apparent object and actual object on a point to point basis. In this chapter the focus is on utilizing this point to point relationship to construct a simulation of the apparent view. Since the effect we are studying is visual in nature this allows direct comparison with actual observations. When the atmosphere has been distorted by a temperature gradient this simulation of the apparent view will be a mirage of the actual object.

5.1 Vision

An interesting question, that will hopefully clarify what we are trying to do, is, how do we see things? In the atmospheric environment the major source of light is the sun. Light rays originate from the sun and travel in all directions following basic physical principles.

Unlike the sun, most objects we see do not create their own light but instead reflect and absorb portions of the light that hits them. It is the reflected portion which contains the intensity and color information that is received by our eyes.

If a ray of light leaves the sun and travels in a straight line to our eye, then we see that speck at its true location. The individual photons of this ray travel to the eye, enter through the lens and are detected by receptors on the retina. The affected receptors on the retina record the color and intensity of the photons on a continuous basis. The information from the entire retina is processed to give a coherent image of the scene. When straight light rays enter the eye the relative spatial relationship of the different rays is maintained. Thus the image on the retina has the object's true shape.

When the light rays travel through a nonhomogeneous medium the ray path will bend. Different ray paths will be bent differing amounts. The relative spatial relationship will be distorted from its true shape. An image with the distorted relationship will be detected and processed by the eye. However, when the image is processed, the mind assumes that the spatial relationship represents the true shape since it has no way of knowing that the ray paths were bent. By mimicking the vision process we are able to synthesize the eye's view.

Starting at the eye location a range of horizontal and vertical ray entry angles are examined. For each entry angle a straight line back projection to the object plane is computed. The assumption is that the eye's image will be its correct size at the object plane. The end point of the straight line represents the eye's perception of that ray's origin. Next, the ray is traced backwards over its real path to determine

its true origin.

This process produces the point to point relationship described earlier.

5.2 Mapping Concept

Consider the relationship between they ray path endpoints and the straight path perceived end points. The resulting data set represents a mapping between two image spaces. The data points determined from the traced rays are points on the real or actual object. The data points of the perceived path are points on the apparent object. Thus the data forms a mapping from the actual object space to the apparent image space.

The complete data set is the medium of translation involved in this mapping. It is for this reason that the data set will be referred to as the transfer characteristic. Table 4.5 contains a typical transfer characteristic except that the 0 angle group was removed. The three columns on the left are apparent point while the three columns on the right specify actual points. The data in Table 4.5 is broken into groups. Each group represents a vertical slice of the mapping relationship. Together the slices form the mapping space. Typically due to the relative invariance of the atmosphere horizontally only a few slices are needed to adequately describe the changes. However, in the vertical direction atmospheric changes are greater and therefore more vertical data will be required in each slice.

While it is not possible to plot the entire transfer characteristic it is possible to plot the object plane to object plane points in the transfer characteristic. Figure 5.1 contains a plot of the transfer characteristic for data set SET 2. Notice that there is

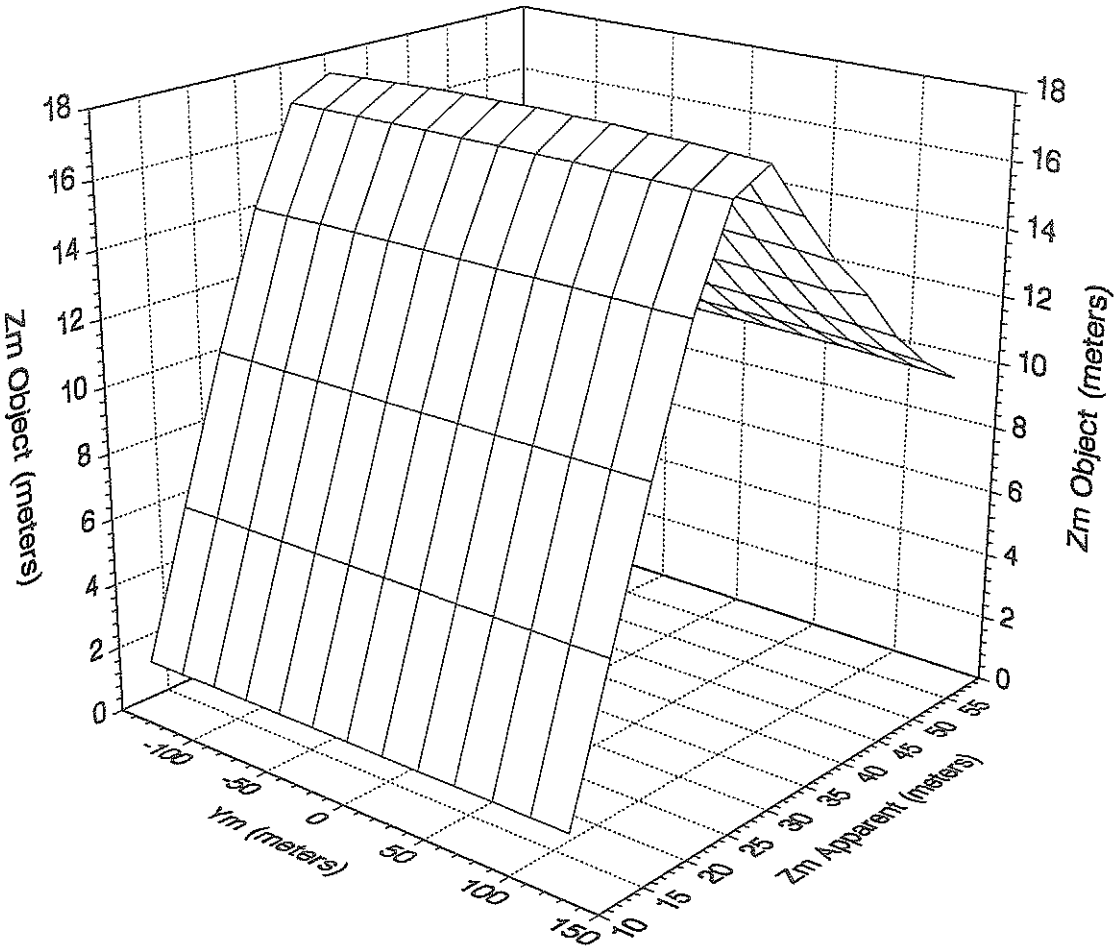


Figure 5.1: **Set2 Transfer Characteristic**

very little vertical change in the transfer characteristic. In this transfer characteristic there are two planes joined by a transition region. The larger front plane represents a upright image scaled by (30-13)/(17-1) (the slope of the plane). The smaller back plane produces an inverted image scaled by its slope. Thus the transfer characteristic plot can be used to predict or verify the apparent image. For example consider a spot on the object at $ym = 50$ and $zm = 12$. Project this spot perpendicular to the zm object plane (back one). This projection will intersect the surface twice. At each intersection project a line perpendicular to the zm apparent plane downward. The termination point of this line on the zm apparent plane is the mapped location of the object spot. Our example spot will be mapped to $(ym = 50, zm = 54)$ and $(ym = 50, zm = 29)$.

In general the following term convention will be used henceforth. The term "point" will refer to the actual (x,y,z) distances (in meters) in the real world while the term "location" will refer to the location (in pixels) in the image of the real world.

To better understand what the mapping process involves let us discuss the image space. The typical image space includes the surface of the earth between the observer and the object, the actual object, and the background sky. Figure 5.2 shows an observer looking at a house. The dashed lines represent the extent of his view. This view seen from the observer's perspective might look like Figure 5.3.

In our case the image space will be captured on film. The view through a camera will have much narrower vertical and horizontal extent then with the naked eye. The dashed lines in Figure 5.3 represent the view through a camera. This view is then transferred to film which is a two dimensional medium. A film image of Figure 5.2

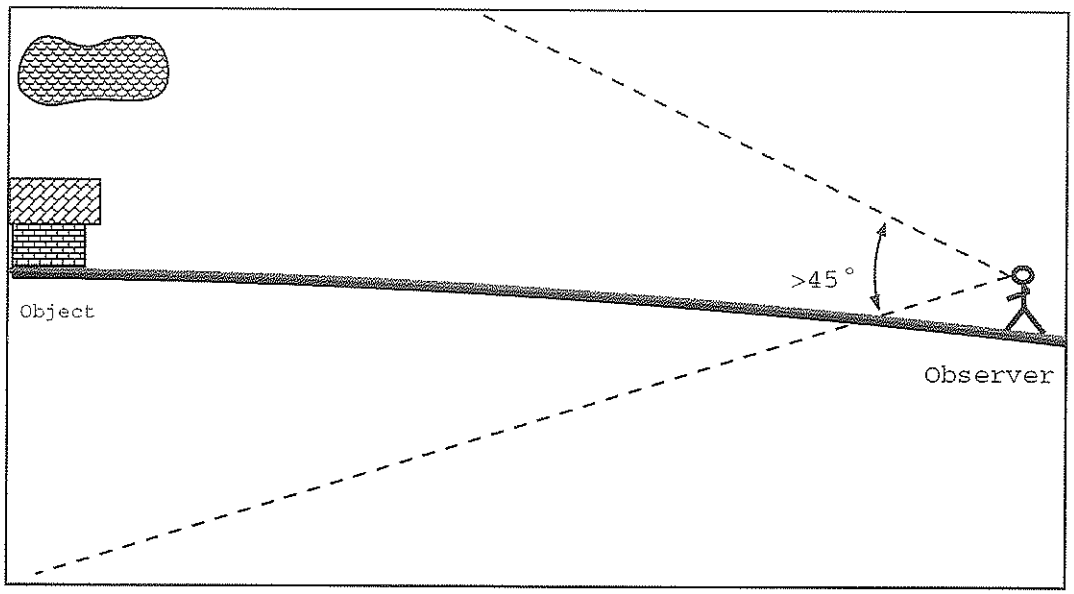


Figure 5.2: **Observer and Object**

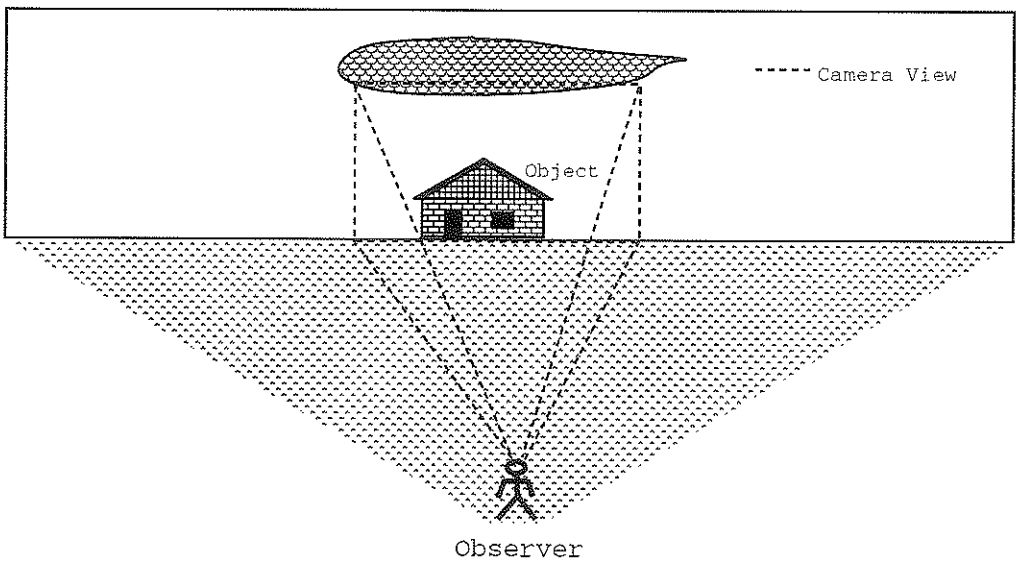


Figure 5.3: **Observer's View**

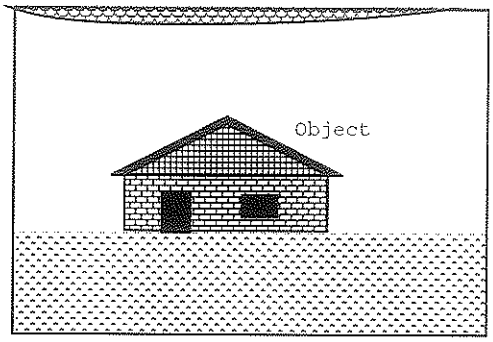


Figure 5.4: **Film View**

might look like Figure 5.4. Notice that the pie shaped section of foreground in Figure 5.3 ends up a rectangular section in Figure 5.4. If the object is beyond the horizon then the lower part of the object will be cut off from the observer and camera views. Without extensive information it would be difficult to determine the portion of object cut off.

The next step is to relate the apparent and actual points to their respective image spaces. In the ray tracing procedures we treated the object and background as a single plane. This made the ray tracing process much simpler.

Since the depth of the object is much smaller then the length of the ray path, the change in the ray path end point due to traveling to the actual depth of the object is negligible. We are not actually interested in what happens with the background. But due to the complexity of differentiating object points from background points it is convenient to include the background in the mapping. The background does add to the appearance of the image. This gives some justification for treating background and object as a single plane. Since the object size and distances involved are known

for the actual image it becomes quite easy to relate the actual end points to the object and sky portion of the image.

The relationship is more complicated for the foreground portion. Basically the way distance is translated in an image has to do with size. The same two objects at different distances will appear at different sizes. That is, the closer image will be larger. This principles applies to the rectangular foreground section of the image space.

To accurately relate foreground points to foreground locations in the object image is complicated. It requires knowledge of where the photographer stood and at what height the image was taken to determine how much foreground is included in the photo. Since the camera translates the spherical surface into a plane the equation $x = \sqrt{R^2 - (z + R)^2}$, which relates x distance to horizontal height, must be used in translating foreground points to foreground locations. Also required are assumptions about the elevation variations of the foreground. Such a process would be very complicated and still not provide a realistic model for the foreground. It is also useful to remember that our chief area of interest is the object.

The following approximate model will be used in relating to the foreground. Our first assumption is that the foreground region begins at the observer's feet and extends to the base of the object. This in conjunction with the assumption that horizontal image distance translates linearly to distance along the foreground gives a simple way of determining x locations in the image space.

Using the pie shaped approximation of the actual foreground and known x distance it becomes possible to determine the y scale for that x distance. We will work out

the actual equations later. Another assumption that needs to be mentioned is that the image is centered with respect to the x axis.

Implicit in our assumption about the foreground being up to the base of the object is that the image space contains the entire image and that the image is on the horizon. This obviously will not be always true of the real images available for analysis. The analysis can still be valid despite this as long as the image is reasonably close to our assumptions. The phenomena we are interested in are mostly visual in nature and the imprecision of our model will affect some detail but not the general shape and trends of the apparent image.

5.3 Interpolation

As we have already described the transfer characteristic is a table of data. In order to map a specific point we need to develop a method of data interpolation. Interpolation will allow us to translate the relative position within the mapping of an apparent point to an object point.

A given apparent location will be surrounded by four apparent points from the transfer characteristic. Associated with these surrounding points will be four object points. The four closest apparent points will best represent the mapping at the given location.

The general idea is to find the relative position of the given location to the surrounding apparent points. Once the relative information is applied to the surrounding object points an object location corresponding to the given location will be found.

Each of these points will either be from the object plane or foreground region.

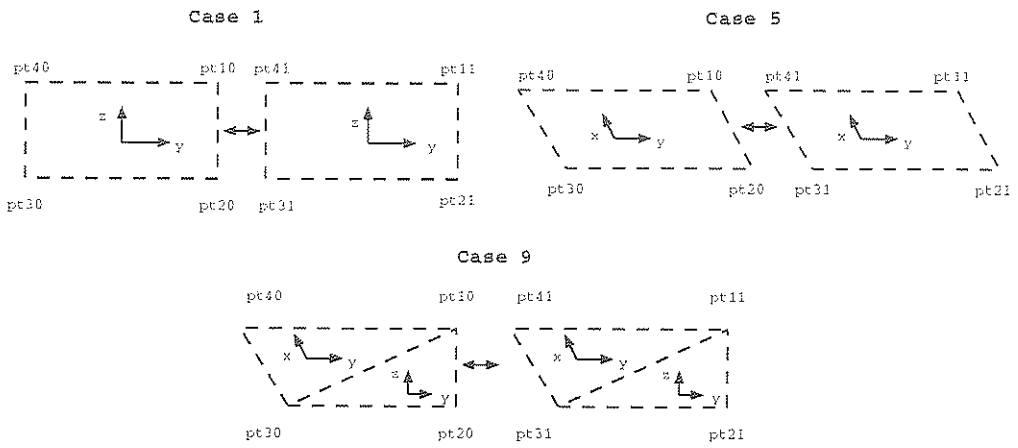


Figure 5.5: **Interpolation Cases**

From this there are three possible combinations of points. The points could all be in the object plane. The points could be all in the foreground. Or, they could be some combination of both.

Between the corresponding apparent and actual points there are nine potential interpolation cases. They are:

Case	Apparent		Object
1	Object	$\leftrightarrow$	Object
2	Object	$\leftrightarrow$	Foreground
3	Object	$\leftrightarrow$	Mixture
4	Foreground	$\leftrightarrow$	Object
5	Foreground	$\leftrightarrow$	Foreground
6	Foreground	$\leftrightarrow$	Mixture
7	Mixture	$\leftrightarrow$	Object
8	Mixture	$\leftrightarrow$	Foreground
9	Mixture	$\leftrightarrow$	Mixture

We can eliminate cases 2,3, and 8 because actual foreground points are always mapped to apparent foreground points. Similarly object points are always mapped to object points which eliminates cases 4,6, and 7. Figure 5.5 contains examples of

the remaining three cases.

Cases 1 and 5 are the basic interpolation set. The other case is necessary if there are horizontal transitions into a inferior projection, or foreground zone in the sky. Case 9 was not implemented. It is not needed for the data available for simulation.

5.3.1 Case 1

In Case 1 we are concerned with determining the relative (z,y) position of the actual point corresponding to a given apparent point. Consider the (z,y) position of the apparent point as the intersection of the two perpendicular lines z=constant and y=constant. In the z direction let e be the normalized distance of the z value between pt10 and pt20 with respect to pt20. Similarly, let f be the position between pt40 and pt30. In the y direction let g be the normalized position of the y value between pt10 and pt40 with respect to pt40 and h between pt20 and pt30. The four values e, f, g, h represents the relative position of the apparent point.

Next use e to find a point between pt11 and pt21. Similarly f, g, h can used to find three other points on the periphery of the four actual points. Join opposing points into lines and determine the intersection of the two lines. This gives the desired relative actual point.

Figure 5.6 shows the various points. The equations are:

e = (zm-pt20.z)/(pt10.z-pt20.z)
f = (zm-pt30.z)/(pt40.z-pt30.z)
g = (ym-pt40.y)/(pt10.y-pt40.y)
h = (ym-pt30.y)/(pt20.y-pt30.y)

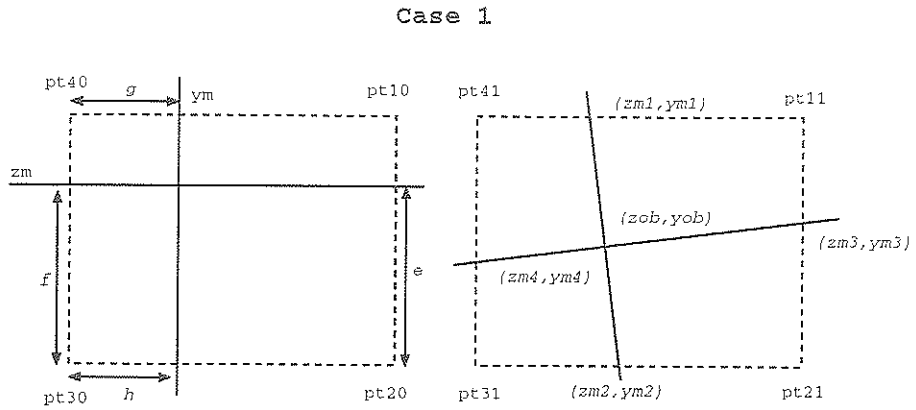


Figure 5.6: **Interpolation Points**

ym1 = g*pt11.y+(1-g)*pt41.y
zm1 = g*pt11.z+(1-g)*pt41.z
ym2 = h*pt21.y+(1-h)*pt31.y
zm2 = h*pt21.z+(1-h)*pt31.z
ym3 = e*pt11.y+(1-e)*pt21.y
zm3 = e*pt11.z+(1-e)*pt21.z
ym4 = f*pt41.y+(1-f)*pt31.y
zm4 = f*pt41.z+(1-f)*pt31.z
zob = (zm3*(ym4-ym3)*(zm2-zm1)-zm1*(ym2-ym1)*(zm4-zm3)
-(ym3-ym1)*(zm4-zm3)*(zm2-zm1))
/((ym4-ym3)*(zm2-zm1)-(ym2-ym1)*(zm4-zm3))
yob = ym1+(ym2-ym1)*(*zob-zm1)/(zm2-zm1)

5.3.2 Case 5

In this case the position of the apparent point will be in the (x,y) plane. The same process as in Case 1 can be used with the z values replaced by their respective x values.

5.4 Images

In order to use the mapping concept we need a method of storing images. The choice of this method will be constrained by the equipment used to display the image.

A computer stored image is often called a digital image. Essentially a digital image is a two dimensional array of pixels. This array is arranged into rows and columns. Each row of pixels defines a horizontal slice of the image. Each column of the array is a vertical slice of the image. By using these two coordinates any pixel in the array can be referenced. The value of the pixel represents the intensity of light associated with that position. A pixel can be used to code color images by storing the intensity of three primary reference colors associated with the pixel. The dimensions of the array and the range of values for the pixels determines how close the digital image approximates the actual image. The dimensions of the array are limited by the method used to determine the digital image, and by the hardware used to display the image. Further, the hardware will specify the range of values to be assigned to each pixel. For our system the image is limited to 512 rows by 512 columns, and to pixel values between 0 and 255. The image is stored in file in byte form in a row by row format.

The source images used were digitized from slides using a commercially available image board and video camera. Most of the images were translated from 480 X 640 format to 512 X 512 format.

Images are sized in units of pixels. The actual scale of a pixel is dependent on what the image contains. Points in the transfer characteristic are with respect to the horizon and the coordinate system. This system must also be applied to images

in order for us to find the pixel location associated with each point in the mapping process.

5.5 Map Procedure

By building on the interpolation process a procedure for mapping an entire image will be developed. Prior to the mapping procedure the actual image will be loaded, the reference horizon determined, and the scale factor calculated. This together with the assumption that the center of the image is the x=0 axis will allow us to find the appropriate pixel for a given image location.

The mapping from object to apparent image is not one to one. An object point may be mapped to multiple apparent locations. However, the reverse mapping, apparent to object, is one to one. By reverse I mean that we will select an apparent point and find its corresponding object point. This process is much simpler than trying to find all the apparent points corresponding to a object point. One problem is that the resulting image may not fit the image space defined by our file format. We must either limit the range of the mapping process or pad the apparent image to fit the file size.

Let's define $zmax$ as the highest height in the object plane covered by the mapping. Then let $zmin$ be the lowest height in the object plane. And $xmax$ is the point closest to the observer in the foreground region. These three parameters define the vertical extent of the apparent image. The difference between $zmax$ and $zmin$ is the extent of the object plane and $xmax$ plus the distance to the object is the extent of the foreground plane. From these parameters we can determine $zmaxi$ which is initially

is $zmax$ measured in pixels, $xmaxi$ which is the negative length of the foreground in pixels, and $zmini$ which is $zmin$ in pixels.

The extent of the apparent image is $zmaxi - zmini - xmaxi$. If this number is larger than the size of the image then $xmaxi$ is halved and $zmaxi$ is recalculated so that the extent is exactly the image size. When the extent is less than the image size then half the difference is placed as white space before (top of) the image and the rest as white space after (below) the image.

The horizontal extent is automatically set to the width of the image size. By using a generous range of horizontal angles in the ray tracing process there should be no problems with the horizontal coverage of the mapping data. If the corresponding actual point is beyond the actual image then a white pixel is stored in the apparent image. $Ymini$ and $ymaxi$ are the left and right image limits. They both equal one half the image width in pixels. $Ymini$ will be negative.

The apparent image is bounded by $zmaxi, zmini, xmaxi, ymini, ymaxi$. The mapping process will begin from $zmaxi, ymini$ and proceed across to $zmaxi, ymaxi$. Then the next row down will be processed left to right and so on until $zmini$ is reached. The process is then continued by beginning at the top of the foreground and proceeding until $xmaxi$ is reached.

There are three counters which track the apparent image location. They are xm, ym, and zm. Two other counters b, and a track the surrounding group in the transfer characteristic. To initialize the mapping process we step down through the horizontal region until we reach the region that contains the starting zm value. The b counter is set to the current horizontal region. The b counter forms the outer do-while

loop of $Map\_$. Within this loop is another do-while loop which test for end of region conditions. This loop keeps b the same until xm or zm reach the end of the current region. It also increments a row counter which forces the size of the image to equal the file size. Both foreground in the sky and inferior type mirages tend to introduce extra rows. Within this loop is the inner counter loop a. a starts as the last group in the current region and is decremented until the first group. Each time the a loop cycles xm and zm are incremented/decremented to the next row and the surrounding do-while loop test for end of region.

Within the a loop is the case test switch. If the surrounding mapping points defined by $\{a, b\}, \{a-1, b\}, \{a, b-1\}, \{a-1, b-1\}$ are not in the object plane then a foreground flag is set. Case 1 interpolation is chosen if the foreground flag is not set otherwise Case 5 is used.

Each Case is contained in a while loop which increments the y counter across the current group. As long as the ym value is less than one of the left surrounding points or greater than one of the right points that point is considered in the current group.

The next step is to check if the current apparent location is below the lower border of the current group. If it is then the group below is used for the interpolation process.

The appropriate surrounding points are passed to the interpolation routine along with the current apparent point. The resulting object point is translated into a object location using image scaling factors. The object location is used to transfer the object pixel to the apparent image. If the object location is outside the object image then a white pixel is transferred instead.

The end of region test contained in *end_of_region* starts by assuming that an end

of region condition exists. If we are testing zm then it compares zm to all the lower points in the current region. End_of_region becomes false if zm is higher than any point. There are two conditions to check for in the xm case. The first condition ensures that xm is not less than any lower region point if the mapping is moving down the foreground. The second condition checks that xm is not greater than any lower region point if the mapping is moving up the foreground.

The main scale factor is determined by the procedure *Getscale* described latter. $scale = elev/(hor - refel)$. *Elev* is the height of the object, *hor* is the row number of the horizon, and $refel$ is the row of the top of the object. This *scale* factor is used to translate (z,y) locations to (z,y) points and vice versa in the object and background portion of the image.

For the foreground portion two different scale factors are used. They are $xscale$ and $yscale$. $xscale = (obs- > x - obj)/(HOR - hor)$. $xscale$ is equal to the distance between the observer and object divided by the number of rows below the horizon. This gives the average $xscale$ over the entire foreground region. $yscale$ is calculated based on xm. $yscale = (scale) * (obs- > x - xm)/(obs- > x - obj)$. $yscale$ equals $scale$ at the object and zero at the observer and varies linearly with distance in between. Since xm is incremented by $xscale$; $yscale$ is indirectly dependent on $xscale$.

If a foreground mapping occurs above $zmini$ then a special $xscale$ is used. For each layer the two extreme foreground points are found along with the lowest object point. Together with the current zm value these define an $xscale$ which increments xm so that at the lower edge of the layer it will equal the lowest foreground mapping point.

$xscale = (high\_x - low\_x)/(zm - low\_z) * (scale)$. If the surrounding points are all in the object plane $xscale$ equals zero and if these points are all in the foreground then the regular equation is used. This $xscale$ is used when part of the actual foreground appears in apparent object plane.

Table 5.1 lists all the variables used in $Map\_$ and indicates their function. The flowchart of the map procedure is given in Figure 5.7.

5.6 Mirage Simulation

The program which performs the mirage simulation is called *mir.c*. *Mir.c* incorporates procedures to load the actual image, transfer characteristic, apply the mapping process, and save the apparent image. It is included as Appendix E.

Table 5.2 lists all the procedures in *mir.c*. The last five procedures are used by $Map\_$ and described in the previous section.

Procedure $Init\_6845$ sends the image board initialization commands. $Write\_port$ is used to send these commands to a custom University of Manitoba image board. The procedure *Get_picture* accepts the filename of the object picture and opens that file. If it can't find the file it will tell you and ask for a new file name. The file is loaded in to a memory buffer.

The procedure $DISPLAY$ is designed to display the memory buffer on either a PC Vision image board or a custom University of Manitoba image board. The board type is selected when the program is started.

Gettc loads a file produced by $3d\_ray$ and extracts the transfer characteristic, the

hor	int	Horizon level in object image in pixels
nog	int	Number of vertical groups in transfer characteristic
nor[]	int	Number of horizontal regions in current group
scale	float	Scale of actual image object plane in pixels
obs	vec	Location of observer
obj	double	Distance to object plane from origin
zmax	float	Highest z value in tc
zmin	float	Lowest z value on object plane in tc
xmax	float	X value closest to observer in tc
ym	double	Current apparent y location in meters
zm	double	Current apparent z location in meters
xm	double	Current apparent x location in meters
yob	double	Corresponding actual location to ym
zob	double	Corresponding actual location to zm
xob	double	Corresponding actual location to xm
xscale	double	X scale in foreground region
yscale	double	Y scale in foreground region
low_z	double	Lowest z in current region
low_x	double	Lowest x in current region
high_x	double	Highest z in current region
ymaxi	int	(VER+1)/2 pixels to right of center
ymini	int	-(VER+1)/2 pixels to left of center
zmaxi	int	Pixel height to start mapping
zmini	int	Pixel height to start foreground mapping
xmaxi	int	Pixel distance to stop foreground mapping
a	int	Current group in transfer characteristic
b	int	Current region in transfer characteristic
diff	int	Difference between image size and mapping extent
z	int	Current z pixel
y	int	Current y pixel
vert1	int	= a
vert2	int	= a-1
hort1	int	= b
hort2	int	= b-1
xline	int	Corresponding actual location to xm
zline	int	Corresponding actual location to zm
yline	int	Corresponding actual location to ym
i	long	Pointer to actual image pixel
foreground	bool	Foreground indicator
str[HOR+1]	str	String to hold white pixel row
fp	FILE	File pointer

Table 5.1: **Map_ Variables**

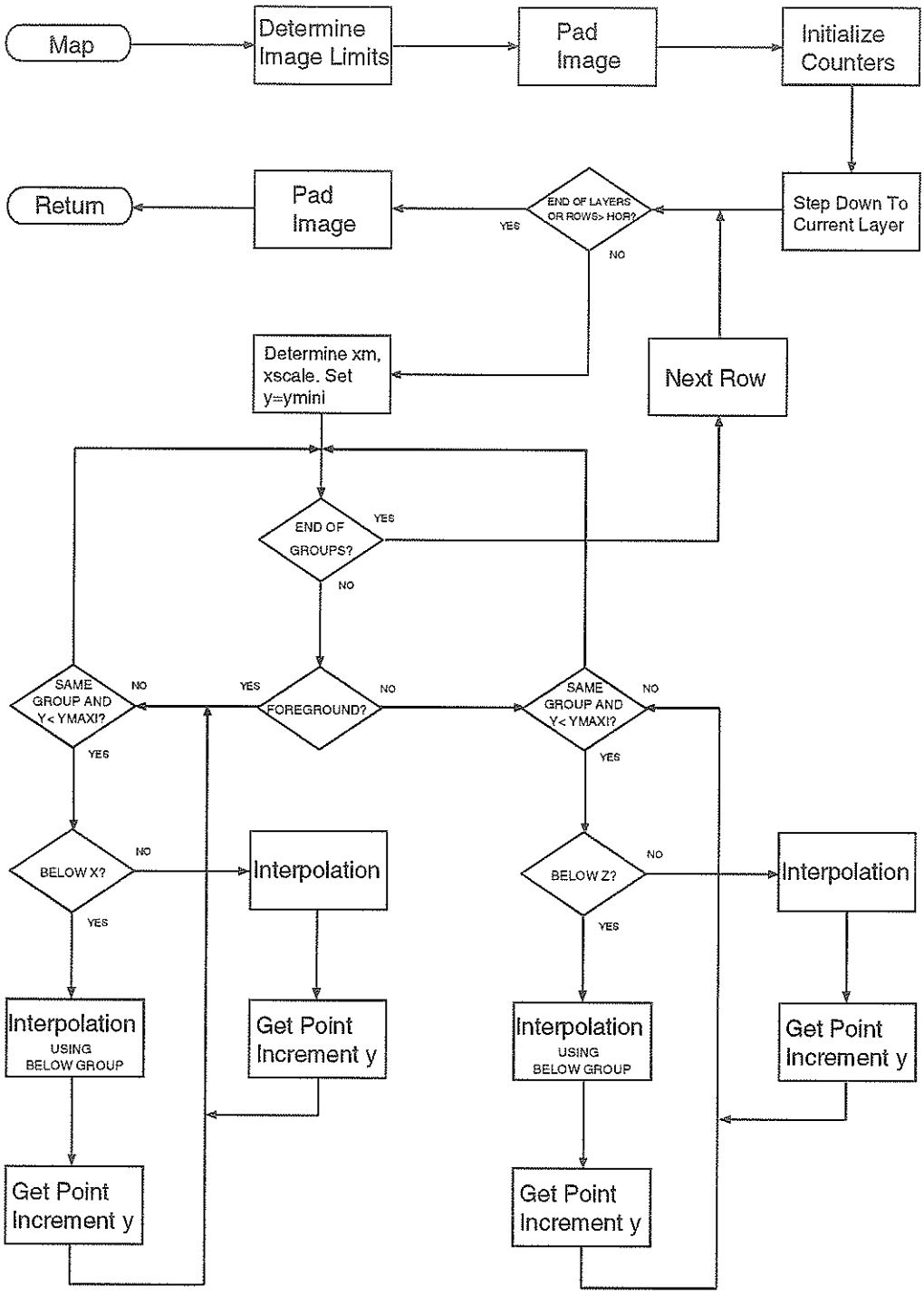


Figure 5.7: **Map Flow Chart**

Name	Purpose
Init_6845	Initialize Image Board
Write_port	Write Pixel to Custom Board
Get_picture	Load Actual Image
DISPLAY	Display Actual Image
Gettc	Load Transfer Characteristic
Getscale	Determine Image Scale
Marker	Add Maker to Image
unMarker	remove Marker from Image
Map_	Mapping Procedure
interpolate	interpolation cases
roundoff	round an integer
end_of_region	test for end of region
belowz	test if point is below current group
belowx	test if point is below current group

Table 5.2: **Mirage Procedures**

object plane distance, and the observer location. It determines the number of regions and groups in the transfer characteristic.

Getscale allows the user to position a movable marker over two image reference locations. First the marker is positioned over the point in the actual image for which the elevation is known. Then the marker is positioned over the horizon. In each case the marker is moved by entering row and column numbers for the pixel desired. The marker is moved until the desired location is reached. The marker movement is terminated by entering -1 -1 and the current marker position is stored. After selecting both reference levels the program asks for the reference height. From this the object plane scale, *scale*, is calculated. *Gettc* uses *Marker* and *unMarker* in conjunction with *DISPLAY* to accomplish the marker movements.

After the mapping process is complete the resulting image is loaded into memory

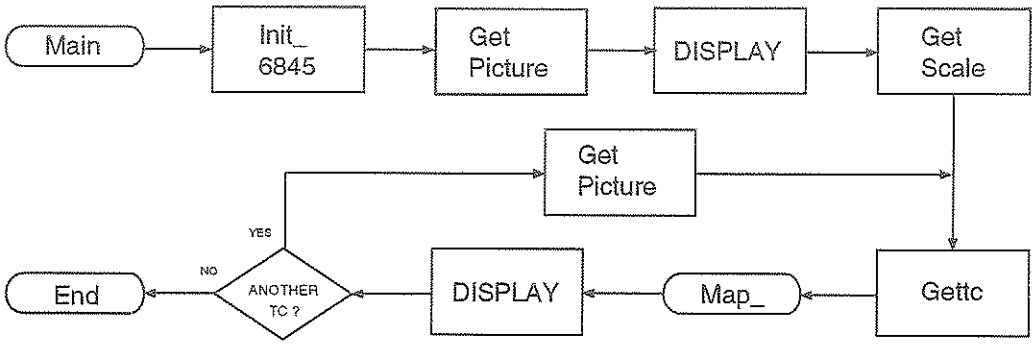


Figure 5.8: **Mir.c Flowchart**

and displayed. Finally if the user chooses to process another transfer characteristic the actual image is reloaded into memory.

The apparent image file will be stored under the transfer characteristic file name plus the identifier ".pic". Figure 5.8 contains the flowchart for *mir.c.*

Chapter 6

Comparisons to Known Situations

In the previous chapters two software tools were developed. The first, *3d_ray*, traces light rays to determine their perceived and actual origin points. These data are used by the second program, *mir*, to simulate the apparent view seen by the eye from an image of the actual scene. In this chapter we will be using these two tools to replicate some actual mirage images and to compare with some earlier two dimensional work.

In each case a data set containing an atmospheric temperature profile, outer shell limits, observer location, and object plane distance is available. From this data *3d_ray* will produce a transfer characteristic file with sufficient coverage to remap the actual image of the object site. The transfer characteristic and actual image will be used by *mir* to produce a simulation image of the apparent view. The main adjustable parameter will be the shell size. A series of initial shell sizes will be processed to determine the general shell size that will produce the desired mirage type. Then additional shell sizes around the general shell shape will be processed to find the best match to the actual mirage image.

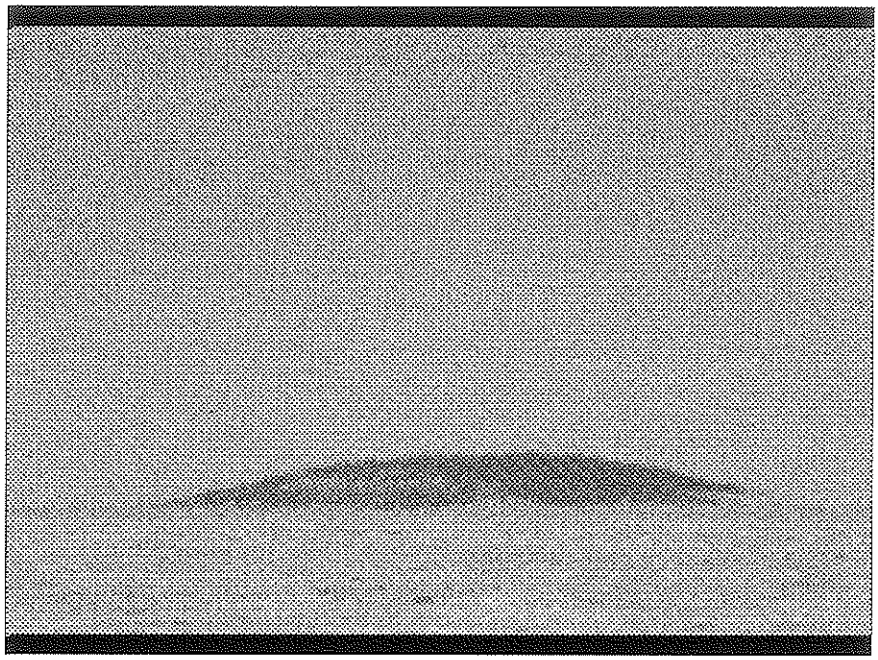


Figure 6.1: **Original Image 79-7-12**

Figure 6.2: **Original Image 83-16-23**

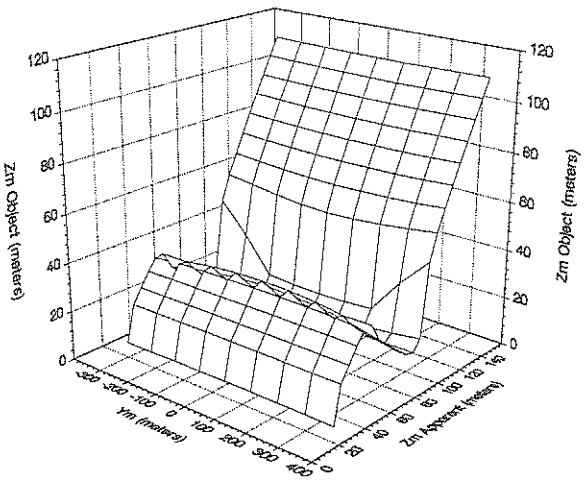


Figure 6.3: **Transfer Characteristic TB8-79-4**

6.1 Whitefish TB8 data set

The first mirage we will simulate is photo 79-4-20, shown in Figure 6.5. This photo is one of many taken in May 1979 during an expedition to Tuktoyaktuk, Northwest Territories, Canada. The photos taken during this expedition provide a collection of normal and mirage scenes. In a similar expedition in May 1983, Professor W.H. Lehn and graduate student John Morrish collected additional photos and measured some associated temperature profiles.

This particular photo, 79-4-20, is of a hill called Whitefish Summit. The height of the hill is 18.7m. The object image will be photo 79-7-12, Figure 6.1, which contains the same hill undistorted. The observer is standing at 20.0km from the hill and eye level is 2.5m above ground. The temperature profile is contained in data set TB8-79-4. This temperature profile was developed by inversion of the refraction data contained in photo 79-4-20 using the technique described in [21].

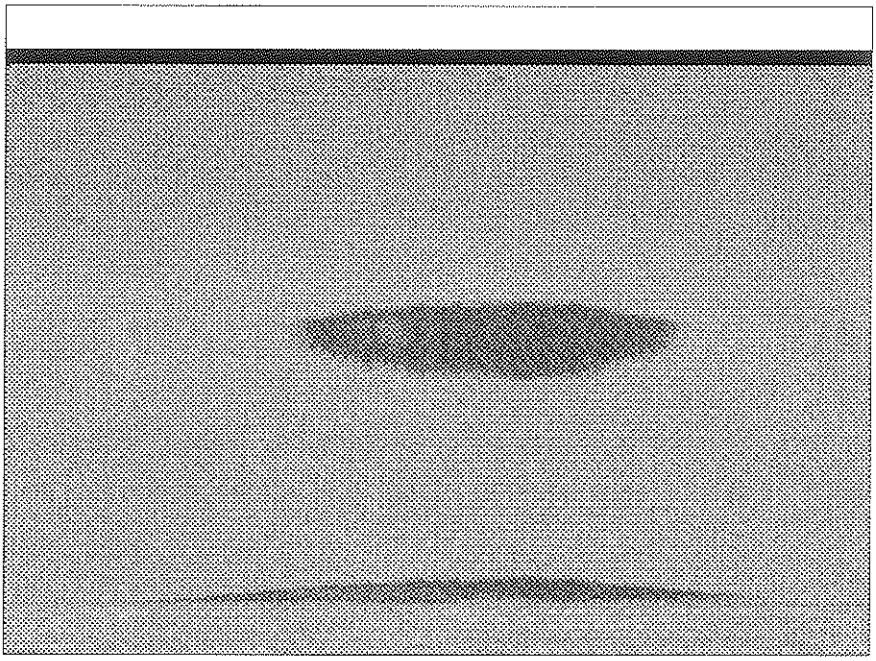


Figure 6.4: **Simulation using 79-7-12**

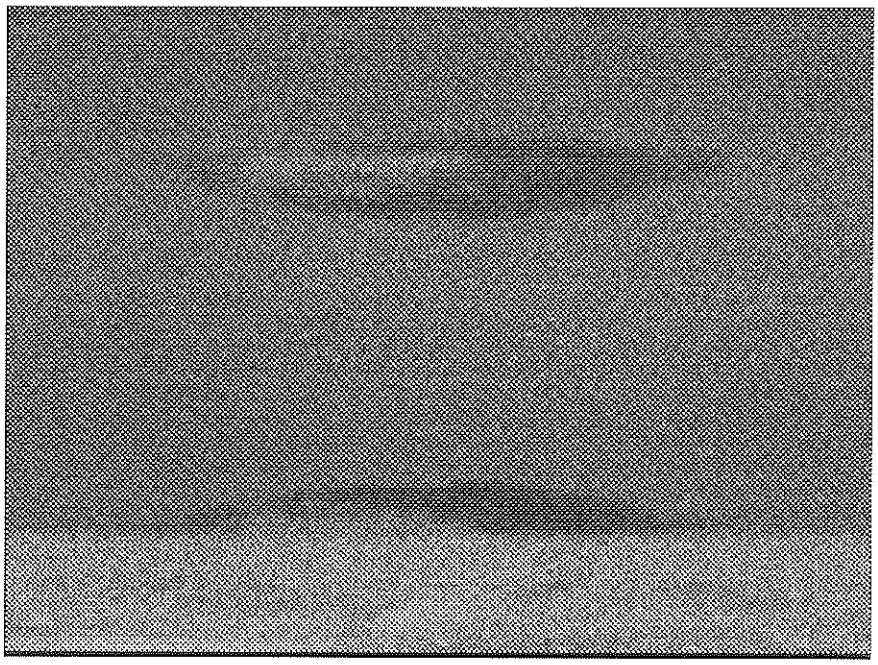


Figure 6.5: **Whitefish Slide 79-4-20**

The atmosphere will be centered half way between the observer and the object similar to earlier two dimensional work. The two remaining parameters are the length and width of the outer atmospheric shell. By adjusting these parameters different atmosphere configurations will be created.

Each atmosphere is processed by *3d_ray* to generate a transfer characteristic. The step size used is 100m. The elevation angles processed are -2 arcmin to 35 arcmin in 1 arcmin steps. The lateral angles processed are 60, 30 , 0, -30, -60 arcmin.

The transfer characteristic is then applied to the object image. The reference pixel level of the object is 350 and the horizon is 410. The object height is 18.7m. The resulting image was compared to the actual mirage.

The first three images processed had atmospheric dimensions of (**a**=8000, **b**=3000), (**a**=10000, **b**=3000), and (**a**=15000, **b**=3000). The last image was closest to the mirage photo. These initial choices were based on examining an atmosphere smaller than the separation between observer and object, an atmosphere the same as the separation, and an atmosphere larger than the separation.

The atmospheric dimension **a** represents the x axis length of the atmosphere and **b** represents the width. Additional experimenting with only the **b** parameter proved that in this case the width could be any value between 100000m and 1000m with out altering the resulting image. Small **b** values however did introduce additional distortion that made the image less like the actual mirage.

Using a width of 3000m the simulation was fine tuned by a series of simulation runs with **a** varying between 14000m and 30000m. The simulation which best matched the original in overall shape was with **a**=17000m. Figure 6.3 contains the transfer

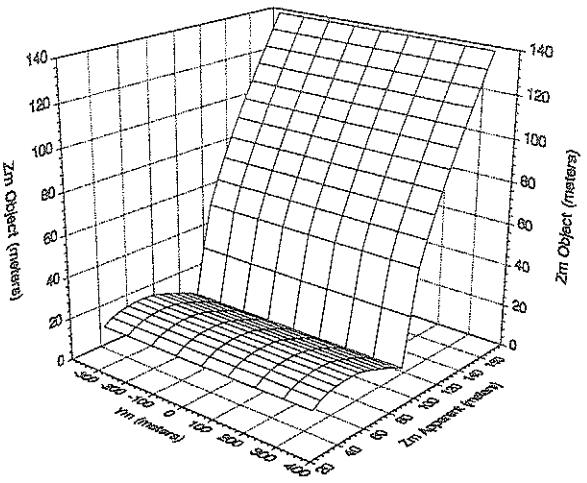


Figure 6.6: **Transfer Characteristic 79-4-27**

characteristic plot of data set TB8-79-4, Figure 6.4 contains the simulation mirage, and Figure 6.5 the real mirage.

The transfer characteristic plot indicates that there will be two upright segments with nearly the same scale and an inverted segment with a slightly smaller scale. Each of the segments will map from the same object area.

Both simulation and real mirages contain a white area to the left side of the floating object. The cigar like shape is similar and the proportions are a like. The lower object however is different. The simulation object has more lateral extent than the real image. Both images have a splotchy texture.

6.2 Whitefish 79-4-27 Data

The photo 79-4-27 is another of the photos taken in May 1979 at Tuktoyaktuk by Professor W.H. Lehn. Again the mirage is of Whitefish Summit and the observer is

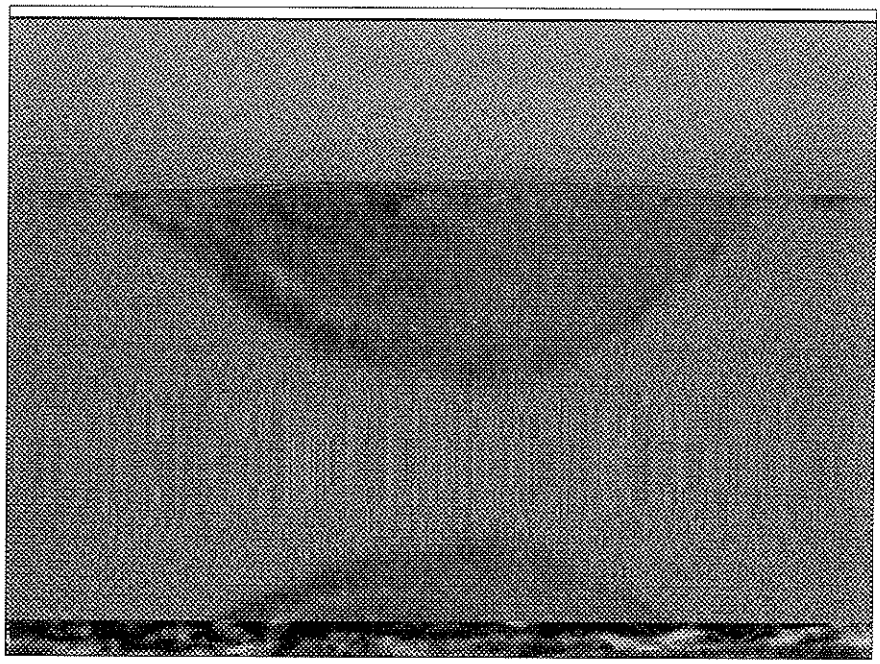


Figure 6.7: **Simulation using 83-16-23**

Figure 6.8: **Whitefish Slide 79-4-27**

at 20.0km.

The associated temperature profile was also developed by inverting refraction data. The profile is contained in data set 79-4-27.

The atmosphere was centered between observer and object. The observer's eye level was 2.5m. The range of elevation angles was -2 arcmin to 35 arcmin in 1 arcmin steps. The lateral angles were 60, 30, 0, -30, -60 arcmin. The object image will be photo 83-16-23, Figure 6.2, which was taken in 1983. The reference row is 350 and the horizon is 440 with a height of 18.7m. In this case the horizon row was moved down into the foreground to compensate for part of the hill that is cutoff by the horizon. This adjustment produced a better matching simulation than the real horizon level of 410 did.

The initial three atmospheric parameter sets were (**a**=8000, **b**=3000), (**a**=10000, **b**=3000), and (**a**=15000, **b**=3000). Again the last set produced the most similar mirage and altering the width had no effect expect at widths less than 1000m. Additional iterations with different a parameters yield a best match with **a**=17000m.

Figure 6.6 contains the transfer characteristic plot, Figure 6.7 contains the simulation mirage, and Figure 6.8 contains the real mirage. The transfer characteristic plot shows that the lower segment will consist of a stretched upright into a stretched inverted region. This will be topped by an upright image. The upright image will have a large scale and be very small.

Both images show a stretched image of the hill at the horizon, a stretched inverted image hanging above the horizon, and a squeezed upright image above the inverted image. Both photos have a band of dark connecting upper and lower images (some-

what hard to see). The overall shapes compare well. The differences in shape detail can partially be attributed to differences between source image and the real shape of the hill and different snow distribution over the hill.

6.3 Sailboat Mirage

In Morrish's [29] work he suggests a temperature profile which might exist over a warm lake on a cool summer day. He later processes a sailboat outline to see how it was distorted. He used a two dimension ray tracing program. The same temperature profile is given in data set Set6. The temperature profile equation is $T = 2.0e^z - 0.02z + 30(^\circ C)$. The observer and object are 8km apart and the boat is 7m high. The observer's eve level is 4m.

For our simulation we used a picture of a sailboat as the actual image. The two dimensional process had spherical shells with radius larger than the earth's radius. To match this process we will select (a=100km, b=100km). Also, the atmosphere will be centered over the observer.

The step size will be 100m, the elevation range -15 arcmin to 40 arcmin in 2 acrmin steps. The lateral range is 60, 30, 0, -30, -60 arcmin. The resulting transfer characteristic is shown in Figure 6.9. The reference row is 115 and the horizon row is 511. The resulting simulation is shown in Figure 6.11. Morrish's result is shown in Figure 6.12. The source image of the sailboat is included as Figure 6.10 for reference.

The transfer characteristic plot shows a lower segment which will invert and compact followed by a segment which is upright and stretched. The upper segment is near normal scale and upright.

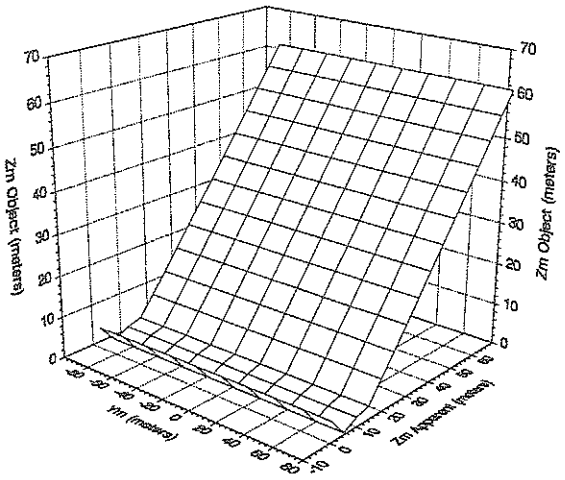


Figure 6.9: **Set6 Transfer Characteristic Plot**

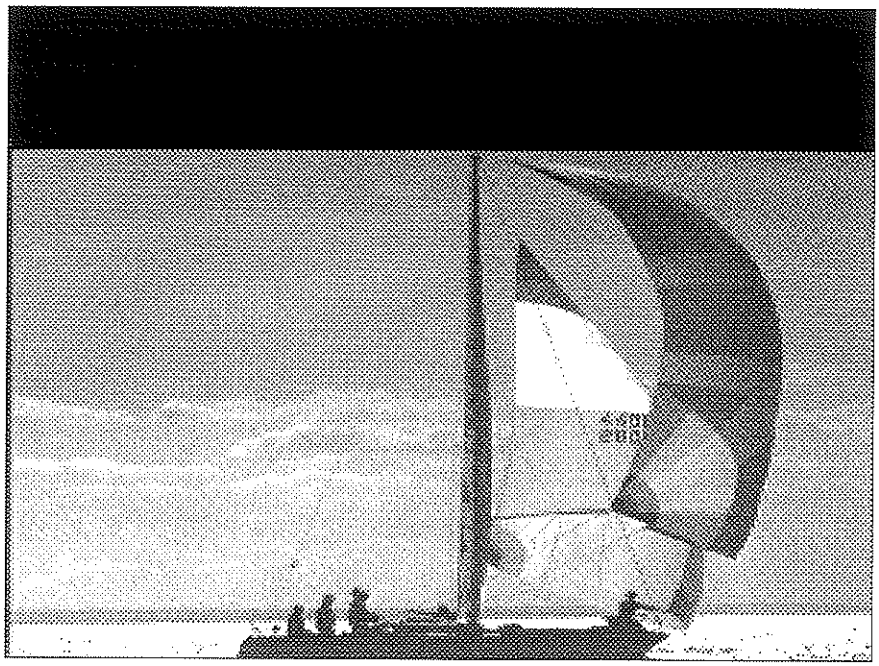


Figure 6.10: **Sailboat Image**

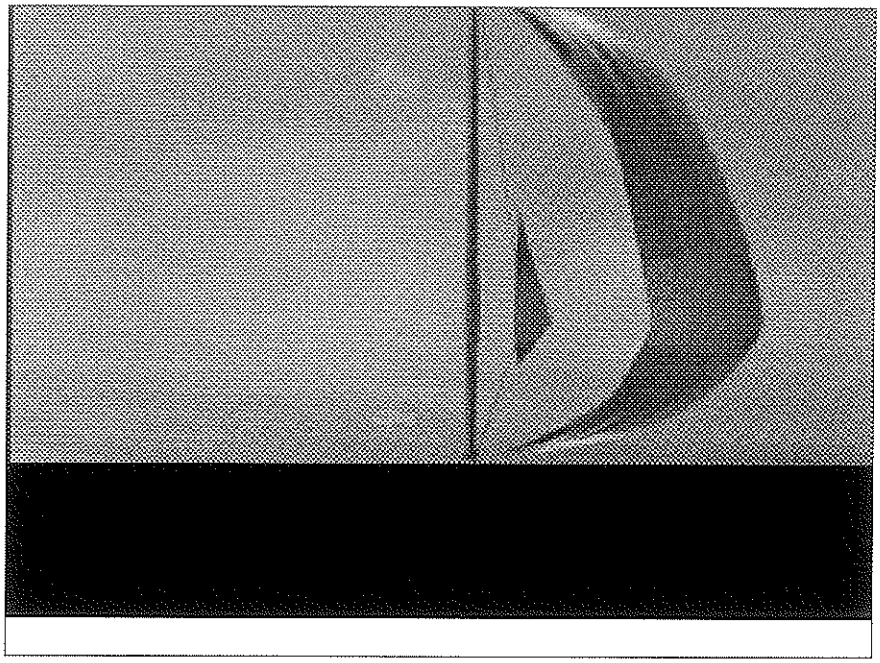


Figure 6.11: **Simulation of Sailboat Mirage**

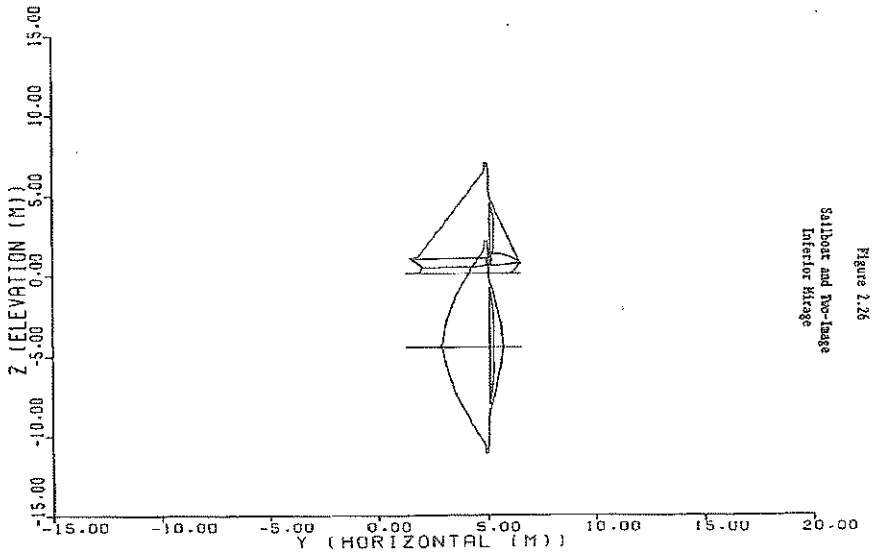


Figure 6.12: **Morrish's Mirage**

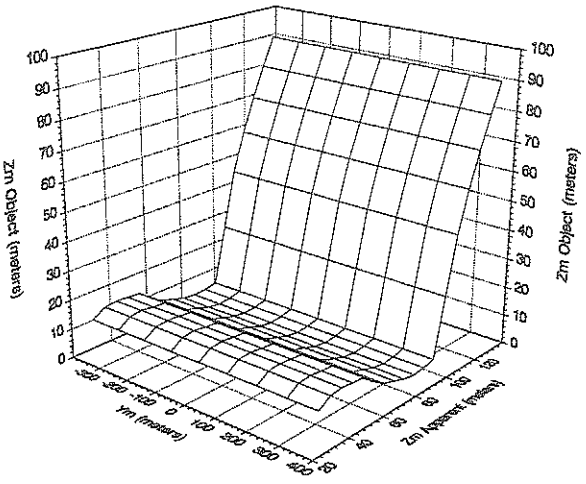


Figure 6.13: **Transfer Characteristic 21.5km**

As expected the data set produced an inferior mirage of an object on the horizon. Both images feature upright and inverted images. These images do not feature the third segment seen in the transfer characteristic plot due to the large size of the source image. The upright image is stretched very similiarly in both images. However, the inverted image is not stretched as much in our simulation as in Morrish's figure. The smaller inverted image is typical of inferior mirages and is predicted by the transfer characteristic plot.

6.4 SETJM3 Data

This data set was also used by Morrish and is included in Appendix D as data set SETJM3. In his case the object was a semi circle with a height of 25m. The resulting mirages are shown in Figure 6.15 and Figure 6.18. For the first mirage the observer-object separation was 21.5km and for the second it was 21.5km.

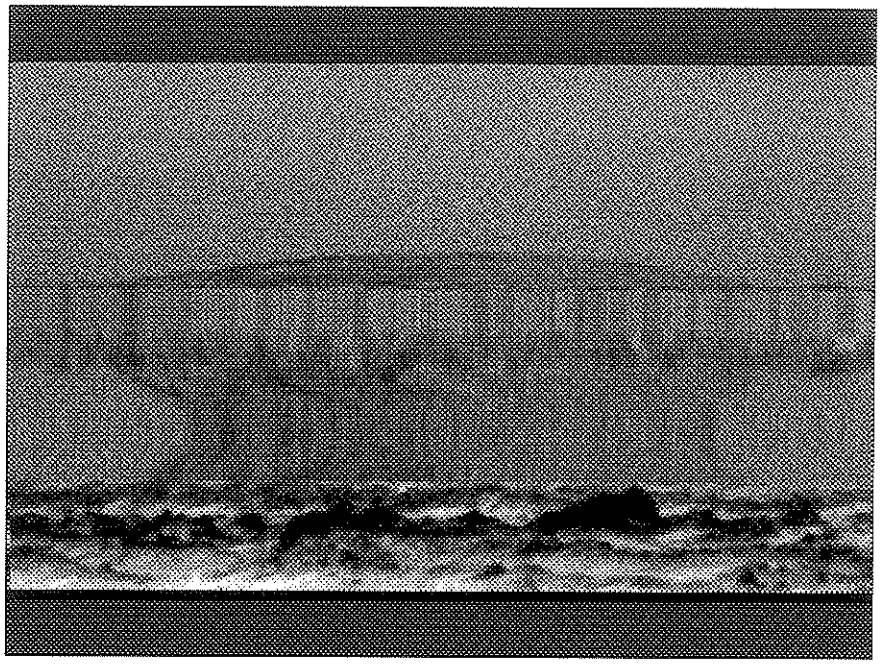


Figure 6.14: **Setjm3 21.5 km**

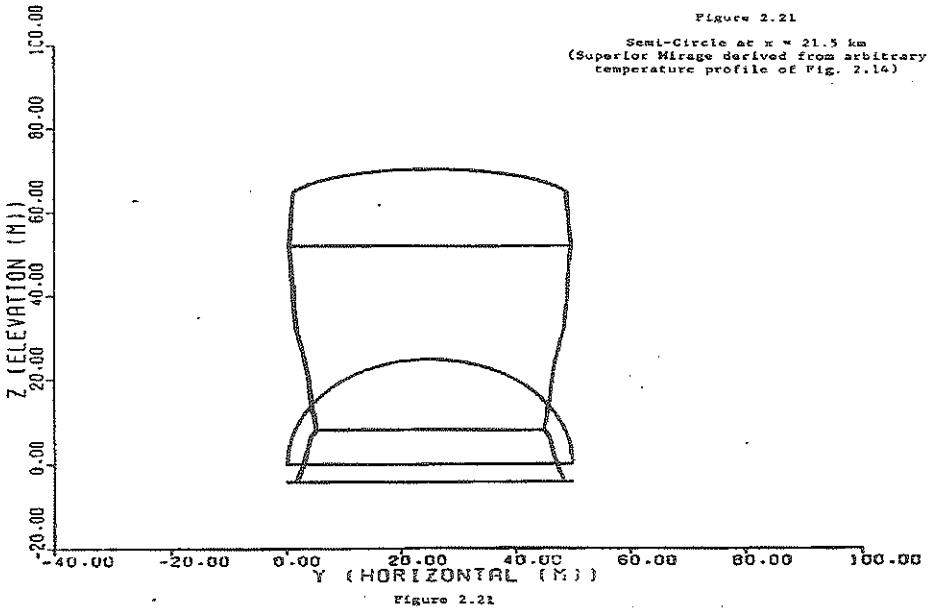


Figure 6.15: **Morrish's 21.5km Mirage**

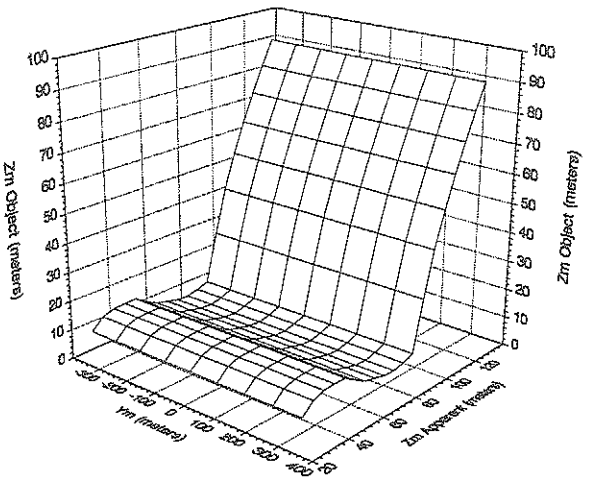


Figure 6.16: **Transfer Characteristic 20km**

For our simulation we will use (a=100km, b=100km), observer eye level 4m, and atmosphere centered over observer. The elevations used are -4 arcmin to 20 arcmin in 2 arcmin steps. The lateral angles are 60, 30, 0, -30, -60 arcmin. The step size is 200m. The resulting transfer characteristic is shown in Figures 6.13 & 6.16 respectively.

The object image used was photo 83-16-23. It contains the Whitefish Summit which is somewhat like a semi circle. The reference row is 350, the horizon row is 410, and the height is 25m. The resulting simulations are shown in Figures 6.14 & 6.17.

The transfer characteristic plots indicate two transition segments followed by a compressed upright segment. The first transition segment is from a stretched upright image into a stretched inverted image and the second zone is from the stretched inverted image into a upright image.

The first simulation, Figure 6.14, starts with a short upright hill followed by a

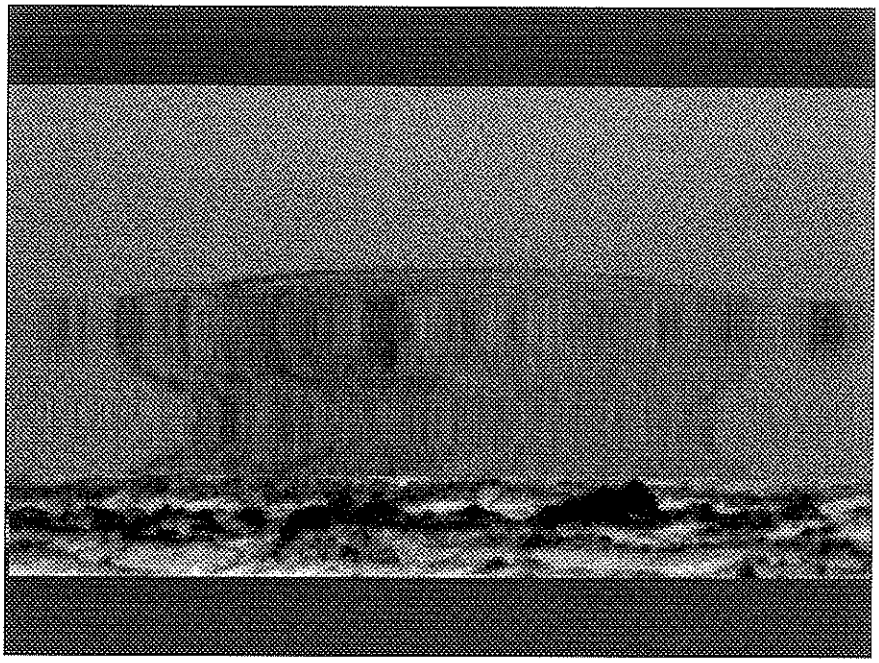


Figure 6.17: **Setjm3 20 km**

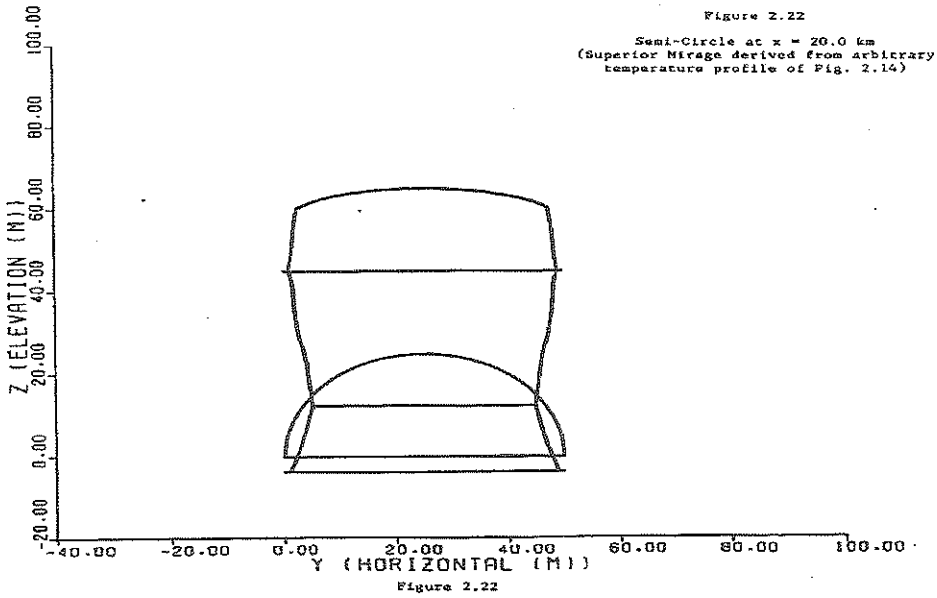


Figure 6.18: **Morrish's 20km Mirage**

stretched transition zone and an inverted region. Another transition zone and an upright hill complete the image. Note in Figure 6.15 that the outline is two thirds the width of the base of the semicircle at its narrowist. The simulation poses the same relationship. Since the hill used in the simulation is flatter than a semicircle the simulation outline does get narrower than the comparing outline.

As expected by examining Figure 6.18 the image in Figure 6.17 is smaller than the first simulation. The same comments as above apply to these images.

Chapter 7

Visual Experiments

The three dimensional nature of our model gives many opportunities to explore.

7.1 Narrow Atmosphere Experiments

As mentioned earlier the width of the atmosphere was not a significant contributing factor to the mirage. Using the TB8-79-4 mirage the following series of images will show the same mirage with different atmospheric widths. We will start with an atmosphere with near earth radius size and step down rapidly.

Figure 7.1 contains the first four images. The width of the corresponding atmosphere is shown below each image. The first image has an atmosphere width of 6000km which approaches the earth's radius of 6730km. Each following image has a width an order of magnitude smaller. All four images are essentially identical.

The second Figure 7.2 contains six images. The 1000m image is the starting point at which atmospheric width becomes a contributing factor. Each image shows the effects of successively narrower atmospheres. The final image with width 125m was

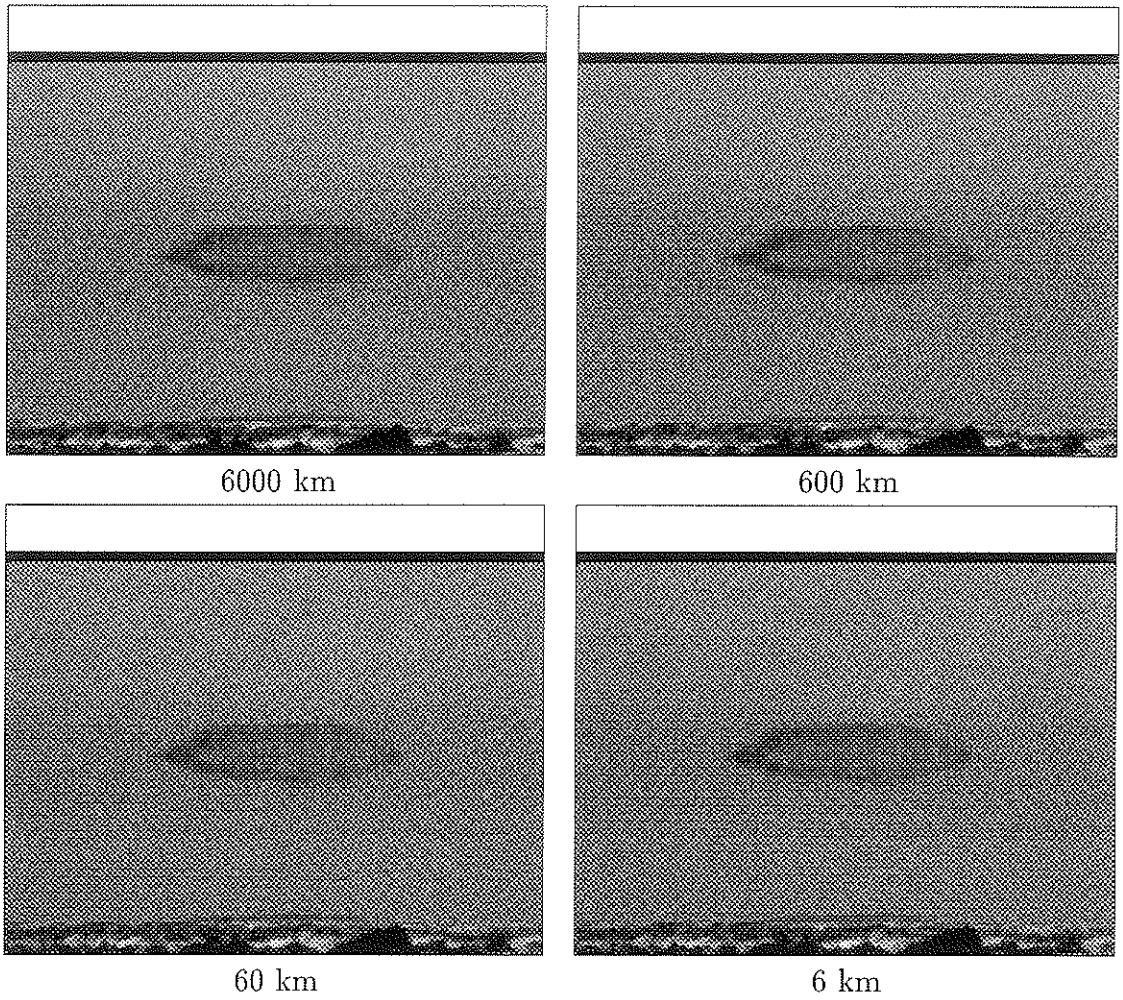


Figure 7.1: **Different Atmospheric Widths Without Distortion**

the narrowest atmosphere that $3d\_ray$ could still trace. In all cases the atmosphere length was 17000m. The elevation angles traced were -2 arcmin to 35 arcmin in 2 arcmin steps. Lateral angles were 60, 30, 0, -30, -60. The step size was 200m except for the last three images which used 100m or less.

We can conclude that in this case the wide atmospheres provided essentially the same conditions in the range of rays that were traced. Such large widths would create

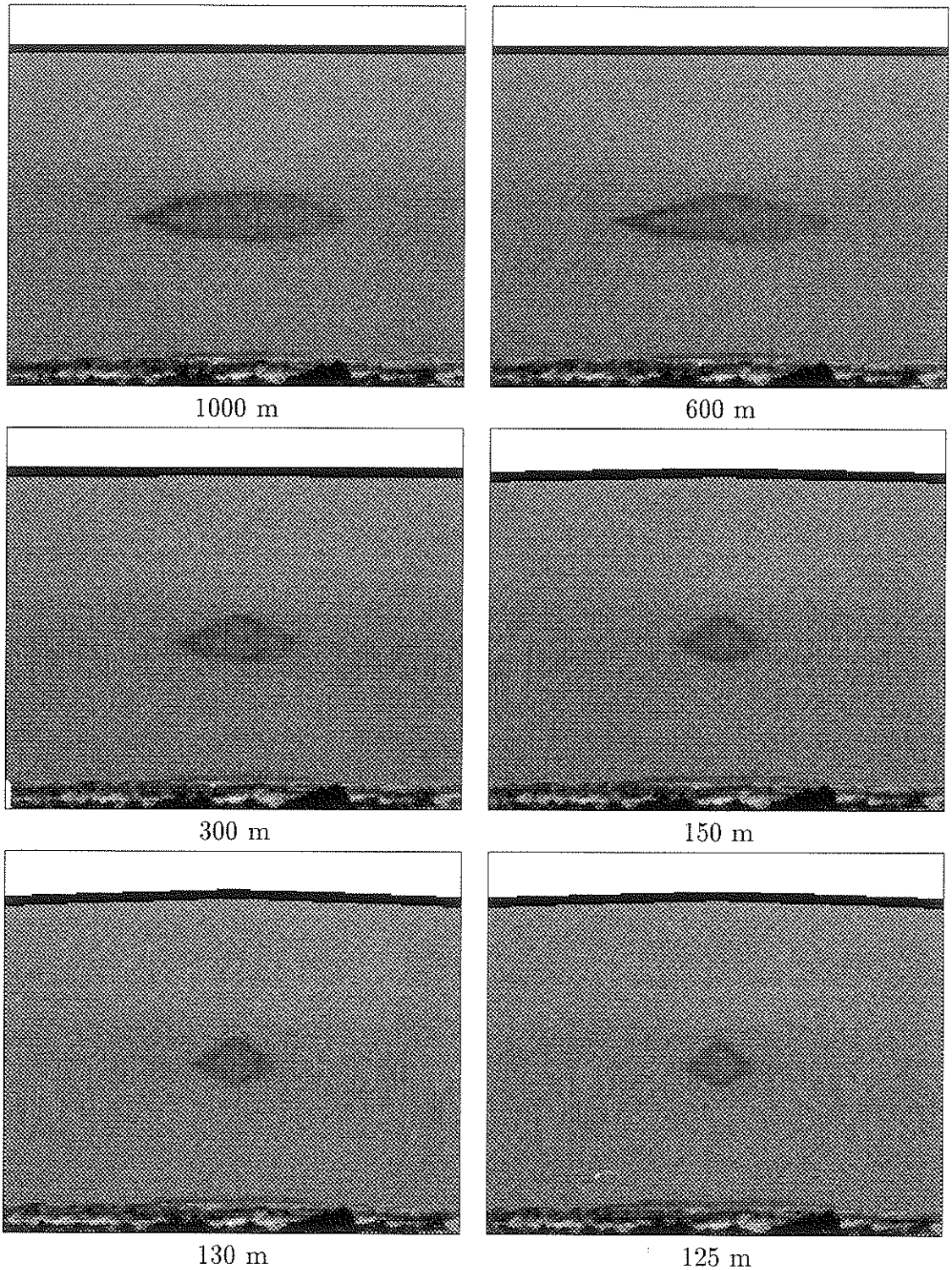


Figure 7.2: **Atmospheric Widths With Distortion**

fairly similar shapes around the origin. Shells which would be fairly flat and uniformly spaced. Once the width approached within an order of magnitude of the width of the object the atmosphere width became a contributing factor. As the atmospheres got narrower the point was reached when the spacing between layers changed near the area of the object. And as the atmosphere continued to get narrower the layer spacing narrowed as well. Due to the elliptical shape of the shells the sides would narrow faster than the central portion. This narrowing is demonstrated by the narrowing sky image. Also notice that the ground hill image remains relatively unaffected in all the images. The lower shells would be less affected by narrowing than the upper shells.

While more analysis with other images and situations needs to done it is reasonable to say that for a large selection of mirages the atmospheric width is not a critical factor in forming the mirage. A wide range of conditions could produce essentially the same mirage.

7.2 Observer Displaced to One Side

The ray tracing model was designed to allow the observer mobility within the model. Until now however there has been no opportunity to utilize this feature. The main reason was that earlier analysis was two dimensional and we where comparing our results with the earlier results. Again using the TB8-79-4 mirage. The following images will show the view of the observer if the observer where moved at a constant distance from the object's center at various angles to the x axis.

Figure 7.3 shows most of the observer locations used, the object plane, and the

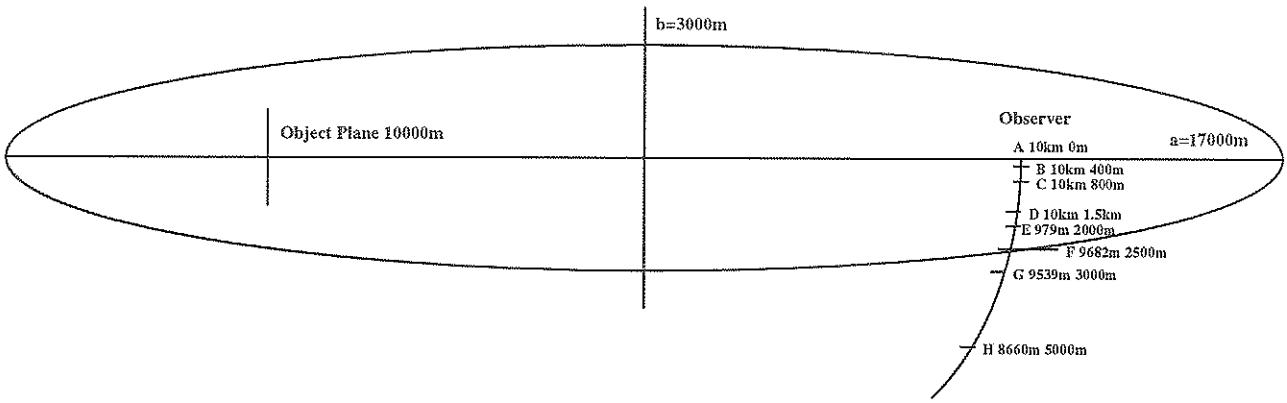
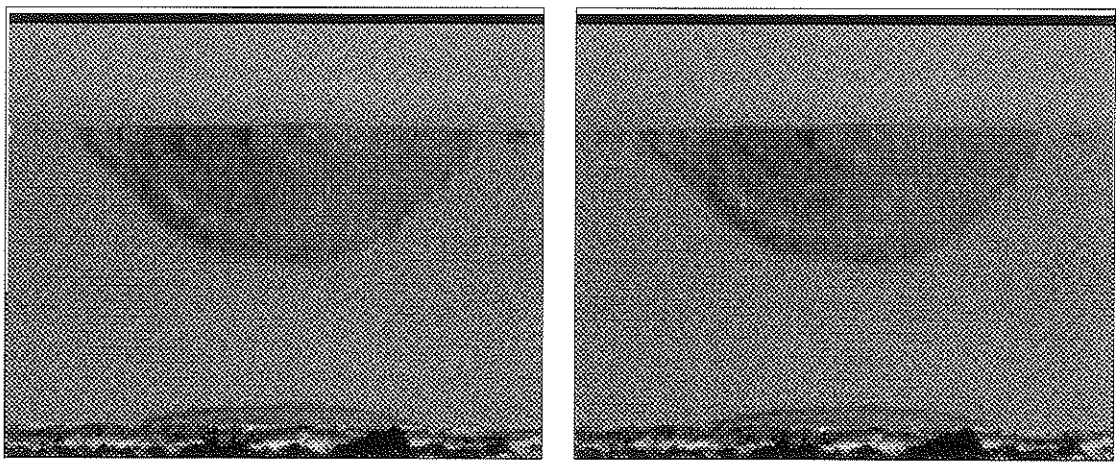


Figure 7.3: **Observer Locations**



9797m 2000m 9797m -2000m

Figure 7.4: **Symmetry Check**

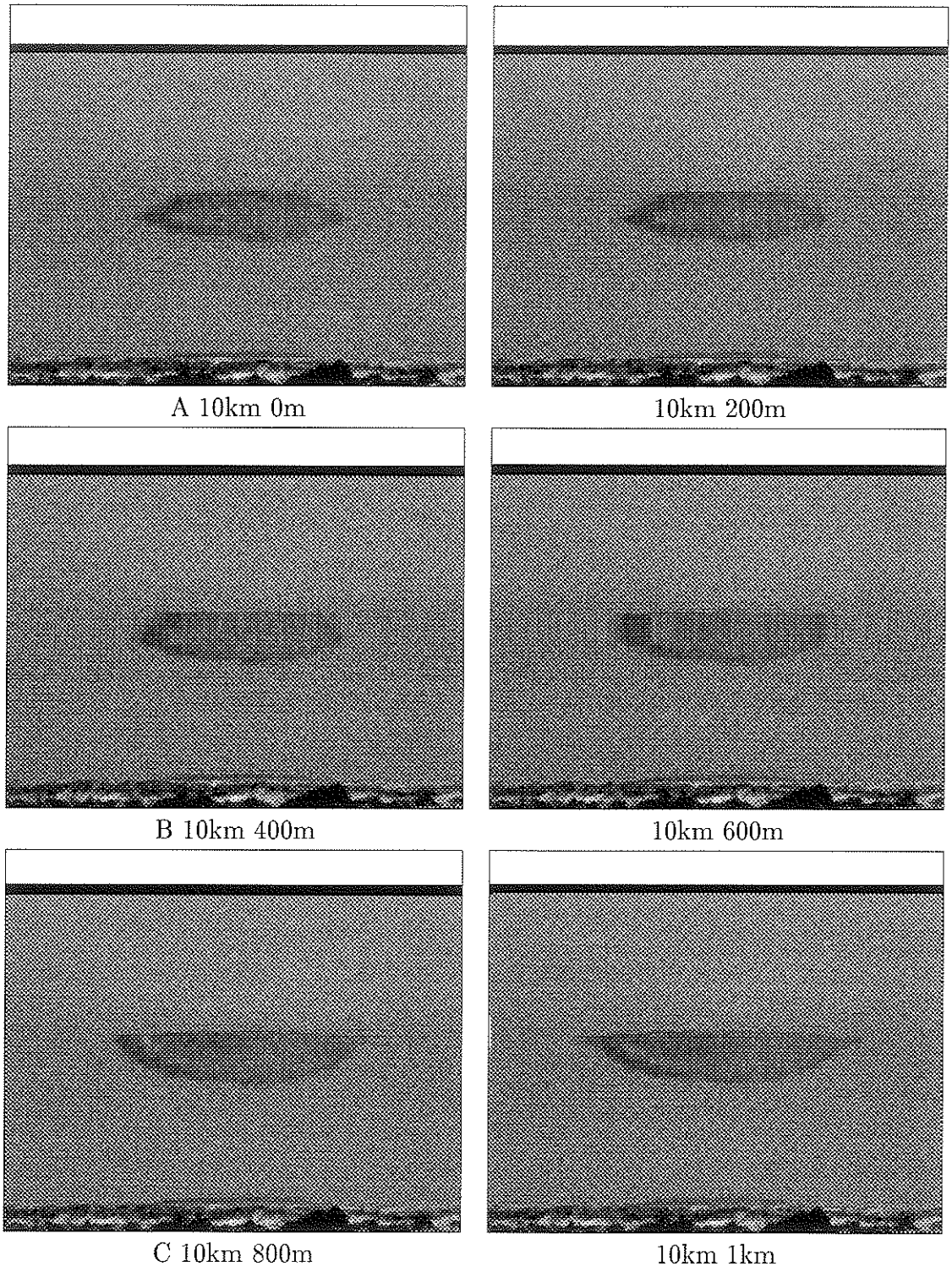
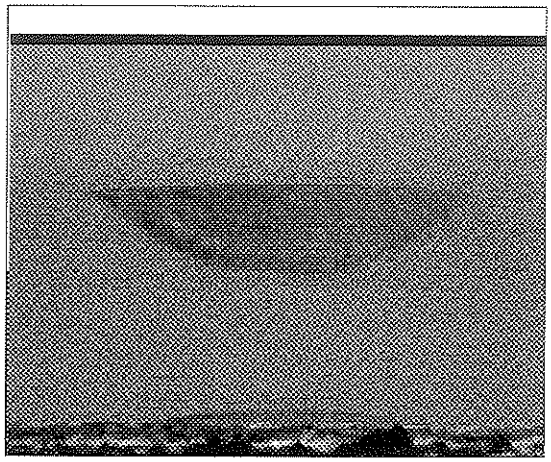
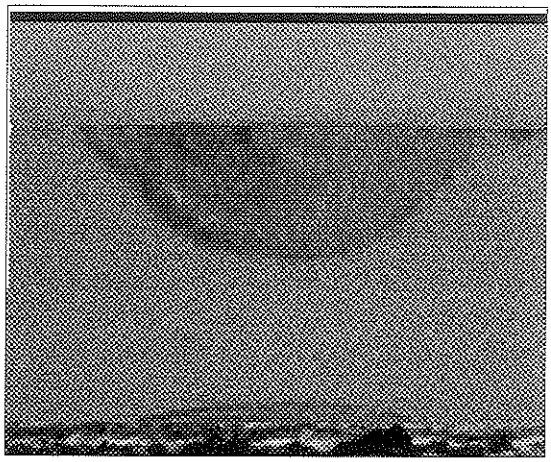


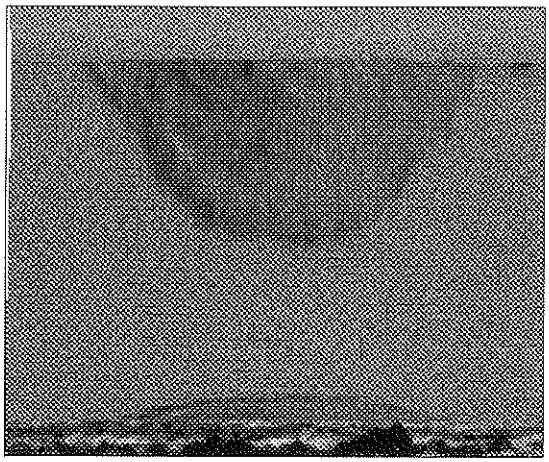
Figure 7.5: **Different Observer Locations**



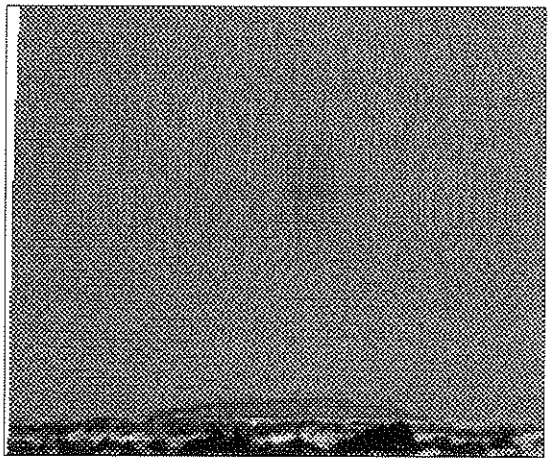
D 10km 1.5km



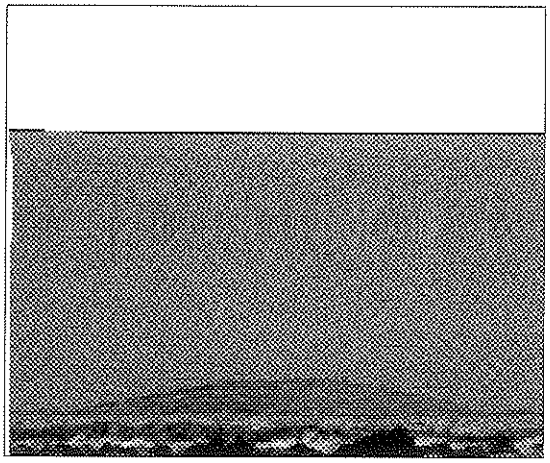
E 9797m 2000m



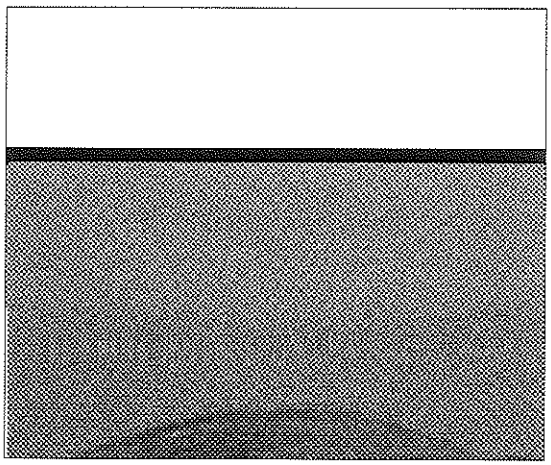
9743m 2250m



F 9682m 2500m



G 9539m 3000m



H 8660m 5000m

Figure 7.6: **Different Observer Locations Continued**

outer ellipse of the atmosphere. The observer was approximately 10km form the origin at each location. Figure 7.5 contains the first six observer locations and Figure 7.6 contains the other six. Under each picture is a letter and two numbers. The first number is the x location and the second number is the y location. The letter refers to an observer location in Figure 7.3 if present. At 200m the mirage is still essentially the same. By 400m the shape has begun to shift. At 600m it no longer looks like the original. In the 800m and 1km images a new mirage has emerged with a small upright image over a larger inverted image. In each case the observer is approximately 10km from the origin.

The series continues in Figure 7.6. The inverted image continues to grow in the next three images. Then somewhere between 2250m and 2500m the mirage has disappeared. From Figure 7.3 this appears to be at the edge of the atmosphere. At 3000m and 5000m more of the hill is exposed which leads to a normal appearing image. All the images used a transfer characteristic with the same ray groups. In the 5000m image the hill is appearing lower than it did in the earlier pictures.

Also to verify that symmetry applied for the observer's location an image was processed for -2000m. This image is compared to the 2000m image in Figure 7.4. As expected the images verify that symmetry exists.

Over a narrow range of a few hundred meters essentially the same mirage will be seen. Over a larger range the image gradually altered shape until a point was reached at which the sky image disappeared.

7.3 Sunset Effects

This final experiment will try to replicate aspects of the Novaya Zemlya effect. The Novaya Zemlya mirage exhibits a ducting window showing some object beyond the horizon. The case we will experiment with is when the sun's image is trapped in the duct and transported from beyond the horizon to the observer's view. This effect is explored in Lehn's paper "The Novaya Zemlya effect: analysis of an observation". [17]

The temperature profiles used in the original paper are given in Appendix D as data set TU8.DAT and TG4.DAT. The only difference is the observer's height which was adjusted until transfer characteristics similar to the original was produced for both Phase I & II profiles. In our case for Phase I the observer height was 27.98m and for Phase II it was 100.1m. The originals were 24.5m and 104.5m respectively. The earlier atmosphere model included pressure variations in its ray path calculations while our does not which could account for some of the difference in heights.

In Figures 7.7 & 7.11 the eye angle is the local uncorrected angle of the ray start point. The exit angle is the angle of the ray as it reaches the object distance with respect to the imposed rectangular coordinate system. Both of Lehn's plots feature corrected eye angles to include atmospheric tilt and atmosphere escape angles with respect to the local observer position. To realign our transfer characteristics we need to add 6 arcmin to both the observer and exit angles to account for atmospheric tilt. Next to make the exit angle with respect to the observer location we need to subtract the base angle of the observer. For the Phase I case the observer stands at 26400m which corresponds to a base angle of 14.2 arcmin and in the Phase II case

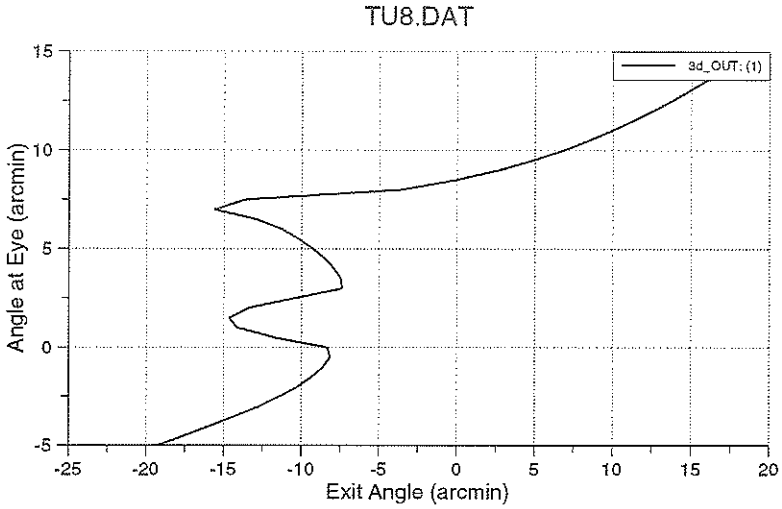


Figure 7.7: **Phase I Transfer Characteristic**

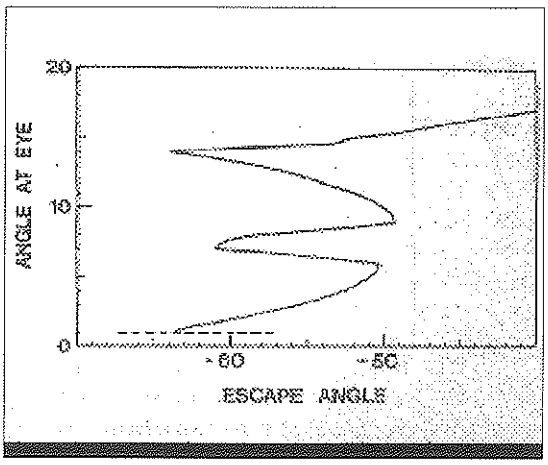
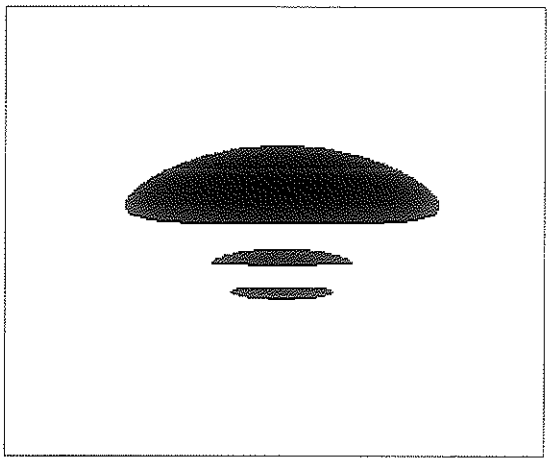
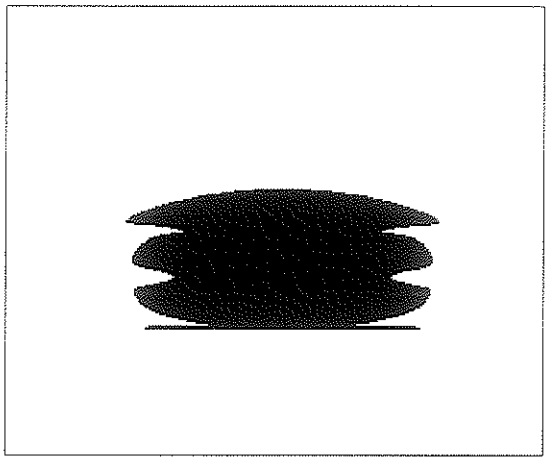


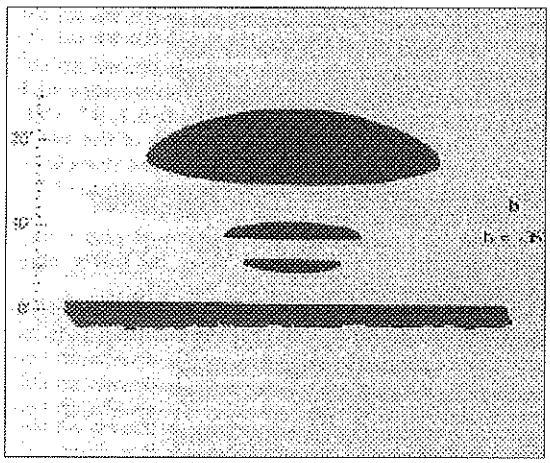
Figure 7.8: **Lehn Phase I Transfer Characteristic**



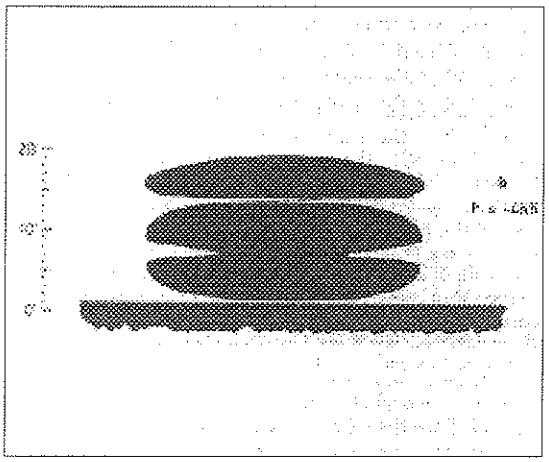
Simulation with h=-34.7 arcmin



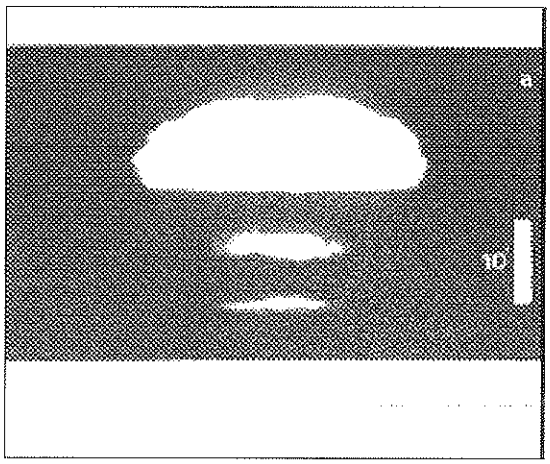
Simulation with h=-44.7 arcmin



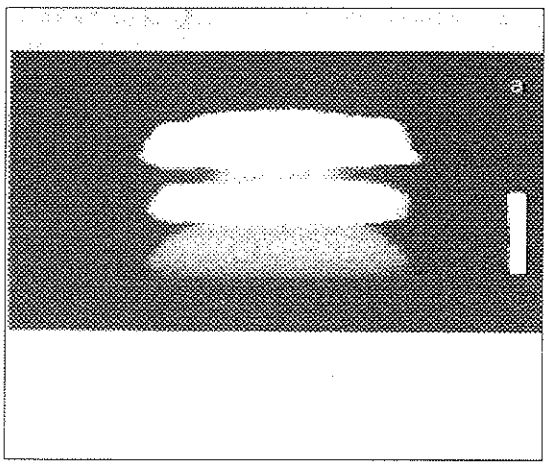
Lehn sim with h=-35 arcmin



Lehn sim with h=-46.5 arcmin

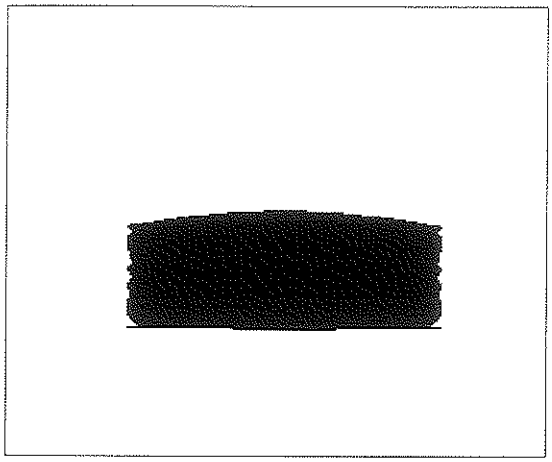


Mirage with h=-35 arcmin

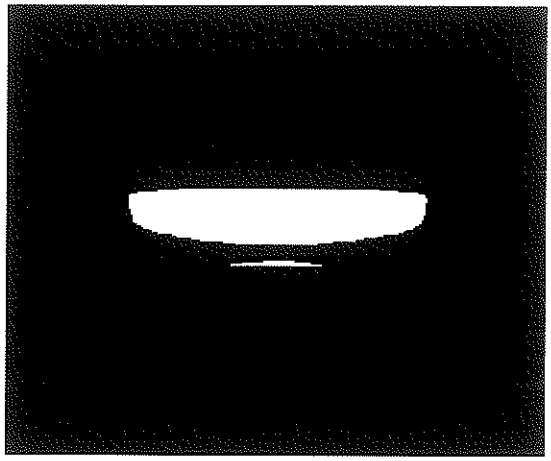


Mirage with h=-46.5 arcmin

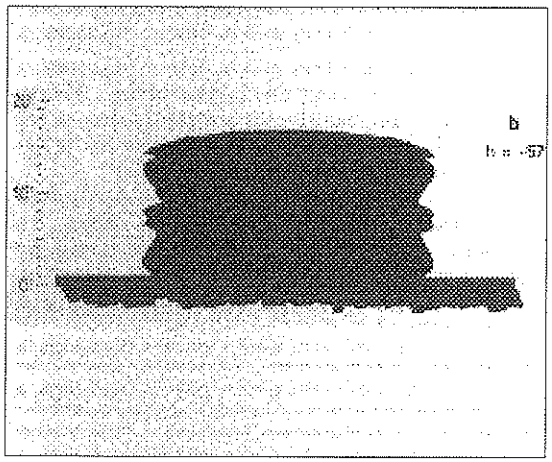
Figure 7.9: **Left side is the 1:34am series. Right side is 1:41am series.**



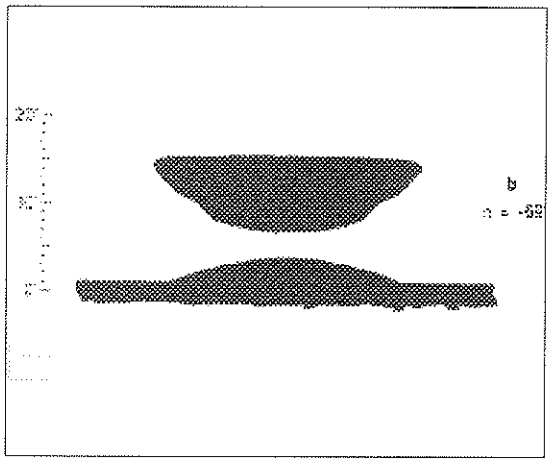
Simulation with h=-52.7 arcmin



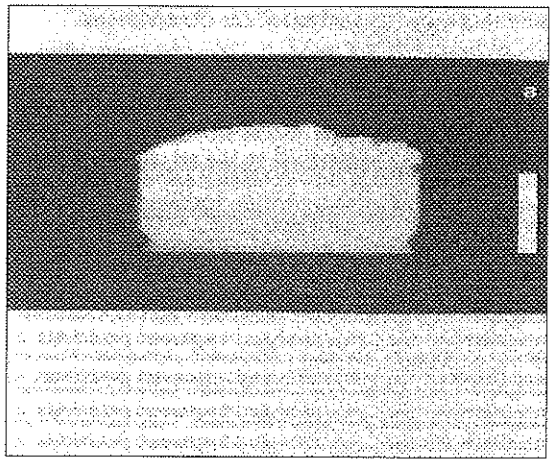
Simulation with h=-73 arcmin



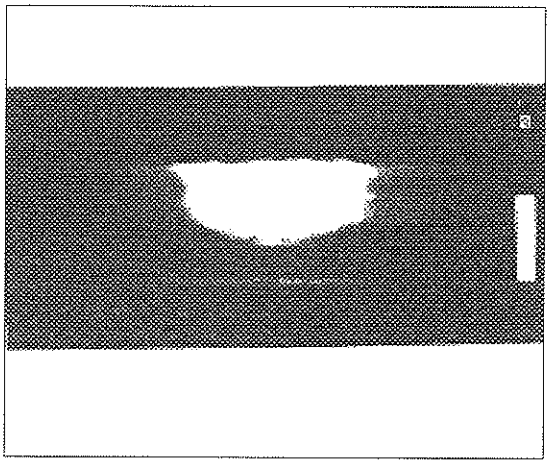
Lehn sim with h=-57 arcmin



Lehn sim with h=-75 arcmin



Mirage with h=-57 arcmin



Mirage with h=-75 arcmin

Figure 7.10: **Left side is the 1:49am series. Right side is Phase II series at 2:06am.**

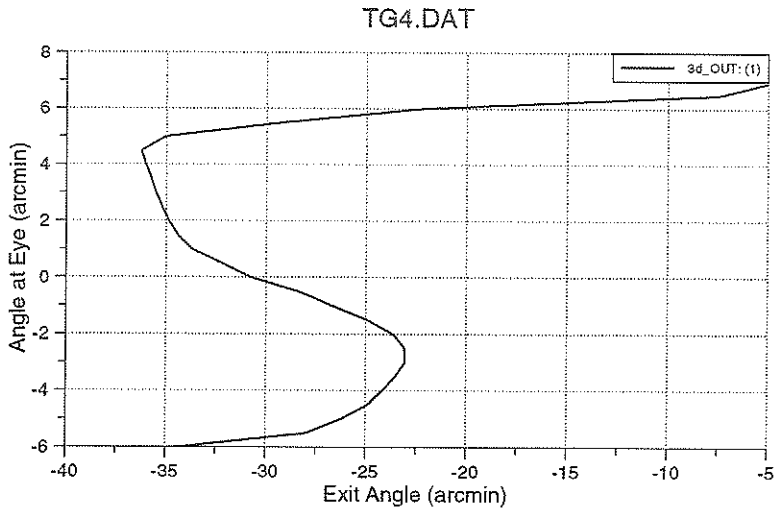


Figure 7.11: **Phase II Transfer Characteristic**

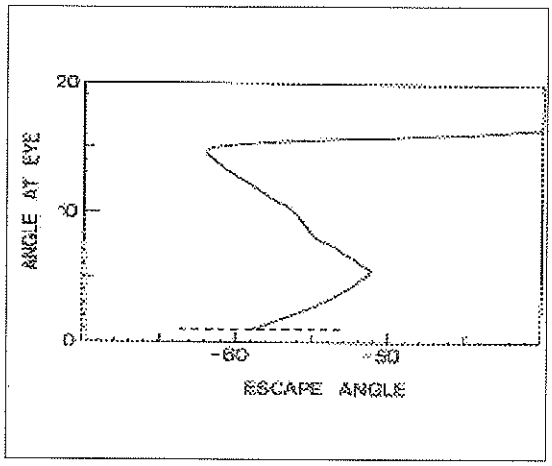


Figure 7.12: **Lehn Phase II Transfer Characteristic**

the observer stands at 0m which corresponds to 0 arcmin. Finally the additional atmospheric refraction required to exit the atmosphere is 33.5 arcmin for the Phase I case and 39 arcmin for the Phase II case. In summary for the Phase I case add 6 arcmin to the eye angle and subtract 41.7 arcmin from the exit angle to give escape angle. In summary for the Phase II case add 6 arcmin to the eye angle and subtract 33 arcmin from the exit angle to give escape angle. Both transfer characteristics are close to the earlier version once compared on the same reference system.

Figures 7.9 & 7.10 show four image sequences. The upper image is the current simulation with the sun center at h arcmin with respect to the horizon, the middle image is the simulation from Lehn's paper, and the lower image is a photo with the sun at h arcmin. The first sequence shows the sun at h=-35 arcmin and corresponds to 1:34am. The second sequence shows the sun at h=-46.5 arcmin and corresponds to 1:41am. The final Phase I sequence shows the sun at -57 arcmin and corresponds to 1:49am. The Phase II sequence shows the sun at h=-75 arcmin and corresponds to 2:06am. All the simulations correspond closely to the actual mirage and earlier simulation.

It has proven possible to replicate the Novaya Zemlya effect with some minor modifications to *3d_ray* and *mir* to output angles instead of position.

Chapter 8

Conclusions and Recommendations

The preceding seven chapters have covered a lot of ground, from developing fundamentals to discuss specific details.

8.1 Conclusions

In chapter 1 six goals where outlined as the scope of this project. The first goal involved developing an understanding of ray tracing in a three dimensional atmosphere model. This was accomplished specifically by incorporating plotting routines in *3d_ray* and by observer movement studies in chapter 7. In a more general sense both programs contributed to this better understanding.

Our second goal was to improve on previous work. This was accomplished through the successful implementation of predictor corrector methods in the ray tracing program. Improved accuracy was achieved.

The third goal was to have a three dimensional ray path plotter and was achieved

by the incorporation of *Plot* into *3d_ray*.

Fourth, to develop a method of applying ray path data to normal images. This was accomplished by implementing the *mir* program and validated by the comparisons in chapter 6.

The fifth goal was to analysis several typical data sets and was accomplished by the work discussed in chapter 6.

Finally, goal six, which was to explore the implications of the ellipsoidal atmosphere, was accomplished through the completion and verification of the main programs, and also through the experiments of chapter 7.

We can conclude that the major goals of this project were accomplished. The two programs *3d_ray* and *mir* have been verified as far as possible and have proved reliable. Also, that many new and interesting possibilities for study exist as a result of this work.

8.2 Recommendations

We have verified the two programs and done some basic exploration. Based on this the following recommendations are proposed:

1. Further study of atmospheric shape and corresponding phenomena be made.

2. Further study of observer location and corresponding phenomena be made.

3. Refine the mirage simulator in terms of foreground modeling and its interpolation process.

4. Seek to use larger image files which will result in more detailed mirages.

5. Study the ray paths themselves to develop a better understanding of the interaction of ray path and atmospheric shape.

Appendix A

3d_ray.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <string.h>
#include <malloc.h>

/*************************************************************************/
/* RAY TRACING                                                           */
/* Working date: OCT 11, 1990                                             */
/* Includes new ellipse model                                            */
/*                                                                       */
/* This program traces the path light rays travel through the atmosphere. */
/* The atmosphere is modelled as a series of confocal ellipsoids         */
/* which are concave toward the earth and represent isothermal layers of  */
/* air.  A three dimensional cartesian coordinate system is imposed such  */
/* that x planes are parallel to the object plane, the z planes          */
/* represent height, and the y planes are perpendicular to object plane.  */
/* The center of the coordinate system is embedded below the surface of   */
/* the earth in such a way that where the major axis of the ellispoids    */
/* touchs the earth is at zero height, and that the center is below the   */
/* highest point of the earths spherical surface.                        */
/*    The observor can be arbitrarily placed on the earth's surface. The  */
/* ellipsoids are defined by specifing the size of the outer ellipsoid    */
/* and using temperature profile data to determine the interior ellipsoids*/
/*                                                                       */
/* Written by Wesley J Friesen, University of Manitoba, Winnipeg         */
/*                                                                       */
/*************************************************************************/

/* Definition of Constants */
```

```
#define erad 6.37e6   /* earth radius */
#define kelvin 273.15   /* add to celcuis to get kelvin */
#define eyel 1.8        /* eye level of observor */
#define minutes 3437.746770784938954435   /* minutes/rad */
#define pi 3.14159265358979323846
#define SIZE 60      /* maximum number of ellipsoids */
#define INDEX(value) (1+(0.0799/((value)+kelvin)))
#define SQR(value) ((value)*(value))
#define SQRT(value) (pow((value),0.5))
#define TRUE 1
#define FALSE 0
#define LARGE 1.0e12  /* radius for a straight line */

/* Definition of Structures */

struct vector
{
double x;
double y;
double z;
};

struct ellipsoidata
{
double a;
double b;
double c;
double temp;
};  /* a,b,c, equation parameters */

struct planenum
{
double a;
double b;
double c;
double d;
};      /* a,b,c binormal direction numbers of plane ax+by+cz+d=0 */

/* Definition of New Types */

typedef struct vector vec;
typedef int bool;
typedef struct ellipsoidata elp;
typedef struct planenum plane;

/* Function Definitions */
```

```
void inputdata(elp [],vec *,double *,double [],int *,FILE *);
/* read input data */
void setupshells(elp [],int *,double *);
/* calculate a b c for each ellipsoid */
void conv_coord(vec *,double *,double [],double *,double *);
/* convert surface coords to cartesian */
bool outside(elp *,vec *,double);
/* determine if point is outside given ellipsoid */
bool on(elp *,vec *,double);
/* determine if point is on a given ellipsoid */
void get_destn(vec *,vec *,double *,double *,vec *,double *,double *);
/* where are we looking to */
bool line_intercept(elp *,vec *,vec *,vec *,double,double);
/* find intercept with outer ellips */
void dir_nums(vec *,vec *, vec *);
/*find direction nums of line ab */
void ellip_norm(vec *,elp *,vec *,double);
/*-norm direct of ellip at point */
void normalize(vec *);
/* normalize given vector */
void plane_of_sight(vec *,vec *,plane *,vec *);
/* find plane of sight nums */
void sphere_centre(double *,plane *,vec *,vec *,vec *);
/* determine sphere center */
void fit_ellip(vec *,elp [],elp *,int,double);
/* find ellipsoid nums for ellipsoid through point */
void sphere(plane *,vec *,vec *,elp *,elp *,elp [],double *,double *,int,
vec *,bool,double);
/* find new direction for ray path and radius */
void tangent(plane *,vec *,vec *,vec *);
/* find ray path tangent */
void new_plane(plane *,vec *,vec *,vec *);
/* new plane of sight nums */
void angle(vec *,vec *,double *);
/* find angle between ray path tangent and refraction gradient */
void inter_ellip_dist(vec *,vec *,elp [],elp *,double *,double *,int,double);
/* distance from point to next ellipsoid along gradient */
bool grounded(vec *,double *);
/* flag if point below ground level */
bool exor(bool,bool);
/* find exclusive or */
bool hit_object_plane(double *,vec *);
/* determine if we're there */
bool intercept(elp *,vec *,bool,double,plane *,double *,vec *,double);
/* test for intercept with next ellipse */
void sphere_intercept(vec *,vec *,vec *,double *);
```

```
/* grad line intercept with ground */
void move(vec *,vec *,plane *,double *);
/* advance the ray along path */
void locate_isotherm(elp *,vec *,bool,vec *,plane *,double *,double,double);
/* find intersection with given ellipse */
void correction(double *,double,plane *,plane *,vec *);
/* average old value with new to provide corrected value */
void plot(char [],vec far *,int *,int []);
/* function to plot the ray paths */
void DrawAxes(float,float,float,float,float,float,float,float);
/* draw the plot axes */
void RayMenu(float,float,float,float,float,float);
/* display menu options for plot screen */
float RoundOff(float);
/* round off function */
void Convert(vec far *,int,int [25],float,float,float,float);
/* convert 3d to 2d visual coordinates */
void Conv_to_earth(vec *,double);
/* convert point coordinates to earth surface point */
void find_ground(double,vec *,double *,vec *,plane *,double *);
/* ground find function */


/* Mainline */

main()
{

 /* Declare Variables */

FILE *out;          /* file to store output */
elp ellip[SIZE];    /* ellipsoidic shell data */
elp nexelp,curelp;  /* vector containing next, current ellipsoid data */
elp interelp;       /* vector for an intermediate fit ellipse */
vec obs;            /* observor location */
vec map[2];    /* actual loc and obs loc */
vec destn;          /* point the eye believes it is seeing */
vec point,oldpt;    /* current location on ray path */
vec line;           /* direction numbers for ray path line */
vec centre;         /* centre of sphere on which ray travels */
vec norm;           /* norm direction nums */
vec tan,oldtan;     /* tangent to ellipse at point */
vec huge *rays;      /* all points calc for current ray group*/
double sight[6];    /* sight angle ranges */
double obj;         /* object plane location */
double base_angle;  /*sighting angle due to earth curvature */
double radius,oldradius; /* radius of ray path arc */
```

```
double shift;       /* stepping variable for locating intercepts */
double hor,ver;     /* loop variables */
double iterate;     /* iteration step size */
double origin_depth; /* distance to earth surface at origin */
bool finished;      /* current ray traced has been terminated flag */
bool upward;        /* flag to indicate is rising through the shells */
int ellipnum;       /* number of shells in atmosphere */
int c,index;        /* counters */
int lastindex;      /* count for last point of previous ray path */
int NumberOfRay;    /* ray count */
int NumPoint[25];   /* count of points in each ray */
int plotfl,print;   /* flags to invoke these features */
int output;         /* flag to invoke storage of output */
plane plofs,oldplos;/* plane of sight data */
char title[80];     /* title to place over ray plot */

 printf("Would you like to plot the ray paths? 0=NO\n");
 scanf("%d",&plotfl);
 if (plotfl!=0)
 {
  /* give opportunity to exit a set mode */
  printf("RAY TRACING WITH HALO\n");
  printf("HAS HERC GRAPHICS MODE BEEN SET ? 0=NO\n");
  scanf("%d",&c);
  if(c == 0)
  {
   return(-1);
  }
 }
 printf("Would you like to see the ray points? 0=NO\n");
 scanf("%d",&print);
 out=NULL;
 /* open storage file */
 out=fopen("3d_OUT","w");
 if (out==NULL)
 {
  output=0;     /* file could not be opened */
  printf("The storage file could not be opened. No storage will occur.\n");
 }

 /* make sure memory is allocated */
 if ((print!=0)||(plotfl!=0))
 {
  rays=(vec huge *) halloc(3500L,24);
  if (rays==NULL)
   return(-1);       /* maxium # of ray points is 3500 */
 }
```

```
 /* get the data and do setup work */
 printf("Enter iteration step size(in m).\n");
 scanf("%lf",&iterate); /* step size */
 printf("step size=%f (m)\n",iterate);
 fprintf(out,"step size=%f (m)\n",iterate);
 strcpy(title,"Ray Path Plot 3d");
 inputdata(ellip,&obs,&obj,sight,&ellipnum,out);
 setupshells(ellip,&ellipnum,&origin_depth);
 conv_coord(&obs,&obj,sight,&origin_depth,&base_angle);

 /* display working data */
 printf("ellip: a      b      c     temp     index\n");
 fprintf(out,"ellip: a      b      c     temp     index\n");
 for (c=0;c<ellipnum;c++)
 {
  printf("%8.3f %8.3f %6.3f %8.5f %10.8f\n",ellip[c].a,ellip[c].b,ellip[c].c,
ellip[c].temp,INDEX(ellip[c].temp));
  fprintf(out,"%8.3f %8.3f %6.3f %8.5f %10.8f\n",ellip[c].a,ellip[c].b,ellip[c].c,
ellip[c].temp,INDEX(ellip[c].temp));
 }
 printf("Base Angle=%2.8f (rad) Origin Depth=%2.8f (m)\n",base_angle,
origin_depth);
 printf("Observer:%f %f %f (m)\n",obs.x,obs.y,obs.z);
 printf("Object Plane:%f (m)\n",obj);
 printf("Vertical Sight Angles: Start=%f Stop=%f Increment=%f (rad)\n",
 sight[0],sight[1],sight[2]);
 printf("Horizontal Sight Angles: Start=%f Stop=%f Increment=%f (rad)\n",sight[3],
sight[4],sight[5]);
 printf(
  "\nTransfer Charateristic coordinates with respect to earth's surface.\n");
 printf("\n              apparent      versus        actual location\n");
 printf(
  "      x            y           z              x            y          z\n");
 /* output to storage */
 fprintf(out,"Base Angle=%2.8f (rad) Origin Depth=%2.8f (m)\n",base_angle,
origin_depth);
 fprintf(out,"Observer:%f %f %f (m)\n",obs.x,obs.y,obs.z);
 fprintf(out,"Object Plane:%f (m)\n",obj);
 fprintf(out,"Vertical Sight Angles: Start=%f Stop=%f Increment=%f (rad)\n",
 sight[0],sight[1],sight[2]);
 fprintf(out,
  "Horizontal Sight Angles: Start=%f Stop=%f Increment=%f (rad)\n",sight[3],
sight[4],sight[5]);
 fprintf(out,
  "\nTransfer Charateristic coordinates with respect to earth's surface.\n");
 fprintf(out,"\n              apparent      versus        actual location\n");
```

```
 fprintf(out,
  "      x            y           z              x            y          z\n");
 radius=LARGE;

 /* begin ray tracing/outer loop */
 for (hor=sight[3];(hor<=sight[4]&&sight[5]>0.0)||(hor>=sight[4]&&sight[5]<0.0);
hor+=sight[5])
 {
  index=0;             /* initialize various counters */
  lastindex=0;
  NumberOfRay=0;
  printf("Hor Sight Angle=%f (min)\n",hor*minutes);
  fprintf(out,"Hor Sight Angle=%f (min)\n",hor*minutes);

  /* inner loop */
  for (ver=sight[0];(ver<=sight[1]&&sight[2]>0.0)||(ver>=sight[1]&&sight[2]<0.0);
ver+=sight[2])
  {
   upward=FALSE;
   get_destn(&destn,&line,&hor,&ver,&obs,&obj,&base_angle);
   if (grounded(&destn,&origin_depth))  /* then find earth intercept */
    sphere_intercept(&line,&destn,&obs,&origin_depth);
   map[0]=destn;

   /* check if ray missed atmosphere */
   if (outside(&ellip[0],&obs,origin_depth)&&
!line_intercept(&ellip[0],&line,&point,&obs,origin_depth,iterate))
   {
    map[1]=destn;
    printf("%f %f %f\t%f %f %f\n",map[0].x,map[0].y,map[0].z,
       map[1].x,map[1].y,map[1].z);
   }
   else  /* super big else statement */
   {     /* ray does hit atmosphere */
    if (outside(&ellip[0],&obs,origin_depth))
    { /* obs outside atmos */
     curelp=ellip[0];
     nexelp=ellip[1];
     if ((print!=0)||(plotfl!=0))
     {
      rays[index++]=obs;
      rays[index++]=point;
     }
    }
    else
    { /* obs inside atmos */
     point=obs;
```

```
      fit_ellip(&point,ellip,&curelp,ellipnum,origin_depth);
      point.x+=10*line.x; /* project along line */
      point.y+=10*line.y;
      point.z+=10*line.z;
      if (outside(&curelp,&point,origin_depth))
      { /* initial angle is upward */
       c=ellipnum-1;
       point=obs;
       while(outside(&ellip[c],&point,origin_depth))
        c--;
       nexelp=ellip[c];
       upward=!upward;
      }
      else
      { /* initial angle downward */
       c=0;
       point=obs;
       while(!outside(&ellip[c],&point,origin_depth))
        c++;
       nexelp=ellip[c];
      }
      if ((print!=0)||(plotfl!=0))
       rays[index++]=point;
    } /* end obs inside atmos */

    /* compute initial ray information */
    ellip_norm(&point,&curelp,&norm,origin_depth);  /* downward pointing norm */
    plane_of_sight(&line,&norm,&plofs,&point);  /* interchange line & norm
for upward pointing norm */
    sphere_centre(&radius,&plofs,&line,&point,&centre);
    sphere(&plofs,&centre,&point,&curelp,&nexelp,ellip,&radius,&origin_depth,
      ellipnum,&oldtan,upward,iterate);

    /* advance light ray location towards object plane */
    oldpt=point;
    oldplos=plofs;
    oldradius=radius;
    point.x-=iterate;
    move(&point,&centre,&plofs,&radius);

    /* ray iteration loop */
    finished=FALSE;
    while (!finished)
    { /* begin while loop */

     /* check if ray is outside last isotherm, if so */
     /* advance ray to the object plane */
```

```
     if (curelp.c==ellip[0].c&&upward)
     { /* begin loop */
      while((!hit_object_plane(&obj,&point) &&!grounded(&point,&origin_depth)))
      {
       if ((print!=0)||(plotfl!=0))
rays[index++]=point;
       point.x-=iterate;
       move(&point,&centre,&plofs,&radius);
      }
     } /* end loop */
     else

     /* check if ray is inside all isotherms, if so advance till repenetration
or grounded or hit the object */
     if ((curelp.c==ellip[ellipnum-1].c)&&!upward)
     { /* begin loop */
      while (!hit_object_plane(&obj,&point)&&!grounded(&point,&origin_depth))
      {
       if ((print!=0)||(plotfl!=0))
 rays[index++]=point;
       point.x-=iterate;
       move(&point,&centre,&plofs,&radius);
       if (outside(&ellip[ellipnum-1],&point,origin_depth)&&(point.z>0.0))
       {
locate_isotherm(&curelp,&point,upward,&centre,&plofs,&radius,
 (oldpt.x-point.x),origin_depth);
nexelp=ellip[ellipnum-2];
upward=TRUE;
break;
       }
      }
     } /* end loop */
     else

     /* check if ray has penetrated the next elp */
     if (intercept(&nexelp,&point,upward,iterate,&plofs,&radius,&centre,
origin_depth))
     { /* begin penetration */
      locate_isotherm(&nexelp,&point,upward,&centre,&plofs,&radius,
       (oldpt.x-point.x),origin_depth);
      sphere(&plofs,&centre,&point,&nexelp,&curelp,ellip,&radius,&origin_depth,
 ellipnum,&tan,upward,iterate);
      correction(&radius,oldradius,&plofs,&oldplos,&oldpt);
      /* curelp and nexelp are interchanged so that the correct values are fnd */
      sphere_centre(&radius,&plofs,&oldtan,&oldpt,&centre);
      point=oldpt;
      point.x-=iterate;
```

```
      move(&point,&centre,&plofs,&radius);
      if (grounded(&point,&origin_depth))
       find_ground(iterate,&point,&origin_depth,&centre,&plofs,&radius);

      if (intercept(&nexelp,&point,upward,iterate,&plofs,&radius,&centre,
origin_depth))
      { /* corrected point still intercepts */
       locate_isotherm(&nexelp,&point,upward,&centre,&plofs,&radius,
(oldpt.x-point.x),origin_depth);
       curelp=nexelp;
       c=0;
       while(ellip[c].c>=curelp.c&&c<ellipnum)
c++;
       if (curelp.c!=ellip[ellipnum-1].c) /* inner shell */
if (!upward)
 if (curelp.c!=ellip[c].c)
  nexelp=ellip[c];
 else
  nexelp=ellip[c+1];
else
 if(curelp.c!=ellip[c-1].c)
  nexelp=ellip[c-1];
 else
  nexelp=ellip[c-2];
       if (curelp.c==ellip[0].c&&upward) /* outer shell */
nexelp=ellip[0];
      }
      else  /* cannot find intercept so drop through to repenetrate test */
      {
       oldradius=radius;
       oldplos=plofs;
       goto rep;
      }
     } /* end penetration */
     else

     /* check if ray repenetrated the current isotherm */
rep: if (exor(outside(&curelp,&point,origin_depth),upward))
     { /* begin repenetration */
      upward=!upward; /* change direction flag */
      locate_isotherm(&curelp,&point,upward,&centre,&plofs,&radius,
       (oldpt.x-point.x),origin_depth);
      sphere(&plofs,&centre,&point,&curelp,&nexelp,ellip,&radius,&origin_depth,
 ellipnum,&tan,upward,iterate);
      correction(&radius,oldradius,&plofs,&oldplos,&oldpt);
      sphere_centre(&radius,&plofs,&oldtan,&oldpt,&centre);
      point=oldpt;
```

```
      point.x-=iterate;
      move(&point,&centre,&plofs,&radius);
      if (grounded(&point,&origin_depth))
       find_ground(iterate,&point,&origin_depth,&centre,&plofs,&radius);

      if (exor(outside(&curelp,&point,origin_depth),!upward))
      {  /* corrected ray still repenetrates */
       locate_isotherm(&curelp,&point,upward,&centre,&plofs,&radius,
(oldpt.x-point.x),origin_depth);
       c=0;
       while(!outside(&ellip[c],&point,origin_depth))
c++;
       if (curelp.c!=ellip[ellipnum-1].c) /* inner shell */
if (!upward)
 if (curelp.c!=ellip[c].c)
  nexelp=ellip[c];
 else
  nexelp=ellip[c+1];
else
 if(curelp.c!=ellip[c-1].c)
  nexelp=ellip[c-1];
 else
  nexelp=ellip[c-2];
       if (curelp.c==ellip[0].c&&upward) /* outer shell */
nexelp=ellip[0];
      }
      else  /* ray does not repenetrate so fit ellipse instead */
      {
       fit_ellip(&point,ellip,&curelp,ellipnum,origin_depth);
       upward=!upward;
      }
     } /* end repenetrations */
     else

     { /* ray did not penetrate or repenetrate */
       /* begin fit */
      fit_ellip(&point,ellip,&interelp,ellipnum,origin_depth);
      /* interelp is first since point is through it and then curelp */
      sphere(&plofs,&centre,&point,&interelp,&curelp,ellip,&radius,&origin_depth,
 ellipnum,&tan,upward,iterate);
      correction(&radius,oldradius,&plofs,&oldplos,&oldpt);
      sphere_centre(&radius,&plofs,&oldtan,&oldpt,&centre);
      point=oldpt;
      point.x-=iterate;
      move(&point,&centre,&plofs,&radius);
      if (grounded(&point,&origin_depth))
       find_ground(iterate,&point,&origin_depth,&centre,&plofs,&radius);
```

```
      if (exor(!outside(&nexelp,&point,origin_depth),upward))
      {   /* after correction the ray path did penetrate the nexelp */
       locate_isotherm(&nexelp,&point,upward,&centre,&plofs,&radius,
(oldpt.x-point.x),origin_depth);
       curelp=nexelp;
       c=0;
       while(ellip[c].c>=curelp.c&&c<ellipnum)
c++;
       if (curelp.c!=ellip[ellipnum-1].c) /* inner shell */
if (!upward)
 if (curelp.c!=ellip[c].c)
  nexelp=ellip[c];
 else
  nexelp=ellip[c+1];
else
 if(curelp.c!=ellip[c-1].c)
  nexelp=ellip[c-1];
 else
  nexelp=ellip[c-2];
       if (curelp.c==ellip[0].c&&upward) /* outer shell */
nexelp=ellip[0];
      }   /* end of penetration */

      else if (!outside(&ellip[0],&point,origin_depth)) /* normal condition */
       fit_ellip(&point,ellip,&curelp,ellipnum,origin_depth);

      else /* the new point is outside atmosphere */
      {
       upward=!upward;   /* treat like a repenetration */
       locate_isotherm(&ellip[0],&point,upward,&centre,&plofs,&radius,
(oldpt.x-point.x),origin_depth);
       curelp=nexelp=ellip[0];
      }
     } /* end fit */

     /* check if light ray has hit object */
     if (hit_object_plane(&obj,&point))
     { /* begin object */
      finished=TRUE;
      point.x=obj;
      move(&point,&centre,&plofs,&radius);
      if (grounded(&point,&origin_depth))
      { /* point at object plane was below ground */
       find_ground(iterate,&point,&origin_depth,&centre,&plofs,&radius);
       get_destn(&destn,&line,&hor,&ver,&obs,&point.x,&base_angle);
       map[0]=destn;
```

```
      }
      if ((print!=0)||(plotfl!=0))
       rays[index++]=point;
     } /* end object */
     else

     /* check if light ray has hit the ground */
     if (grounded(&point,&origin_depth))
     { /* begin ground */
      finished=TRUE;
      find_ground(iterate,&point,&origin_depth,&centre,&plofs,&radius);
      get_destn(&destn,&line,&hor,&ver,&obs,&point.x,&base_angle);
      map[0]=destn;
      if ((print!=0)||(plotfl!=0))
       rays[index++]=point;
     } /* end ground */
     else

     /* otherwise advance ray until at inner surface*/
     if (curelp.c!=ellip[ellipnum-1].c)
     {
      sphere(&plofs,&centre,&point,&curelp,&nexelp,ellip,&radius,&origin_depth,
 ellipnum,&oldtan,upward,iterate);
      if ((print!=0)||(plotfl!=0))
       rays[index++]=point;
      oldpt=point;             /* save current values for use in predictor */
      oldplos=plofs;           /* corrector corrections to ray projections */
      oldradius=radius;
      point.x-=iterate;
      move(&point,&centre,&plofs,&radius);
      if (grounded(&point,&origin_depth))
       find_ground(iterate,&point,&origin_depth,&centre,&plofs,&radius);
     }
    } /* end while loop */

    NumPoint[NumberOfRay]=index-lastindex;
    NumberOfRay++;
    lastindex=index;
    map[1]=point;
    /* convert to earth surface coordinates */
    Conv_to_earth(&map[0],origin_depth);
    Conv_to_earth(&map[1],origin_depth);
    printf("%+11.5f %+11.5f %+11.6f\t%+11.5f %+11.5f %+11.6f\n",map[0].x,
map[0].y,map[0].z,map[1].x,map[1].y,map[1].z);
    fprintf(out,"%+11.5f %+11.5f %+11.6f\t%+11.5f %+11.5f %+11.6f\n",map[0].x,
map[0].y,map[0].z,map[1].x,map[1].y,map[1].z);
   }   /* end of if else loop for inside atmosphere intercept */
```

```
  } /* end of inner counter loop */

    if ((print!=0)||(plotfl!=0))
     for(c=0;c<index;c++)
     {
      rays[c].z+=erad-sqrt(SQR(erad)-SQR(rays[c].x)-SQR(rays[c].y))-origin_depth;
      if (print!=0)
      {
       printf("%f %f %f\n",rays[c].x,rays[c].y,rays[c].z);
       fprintf(out,"%f %f %f\n",rays[c].x,rays[c].y,rays[c].z);
      }
     }

   if (plotfl!=0)
   {
    if (NumberOfRay <= 25)
    {
     plot(title,rays,&NumberOfRay,NumPoint);
    }
    else
     printf("Error: Too many Rays to plot\n");
   }
/* rem call plot */
 } /* end of outer counter loop */

 return(0);  /* return good signal */
} /* end of mainline program */
```

Appendix B

3d_subro.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <string.h>
#include <malloc.h>

/*************************************************************************/
/* RAY TRACING                                                           */
/*                                                                       */
/* Written by Wesley J Friesen, University of Manitoba, Winnipeg         */
/* Working date October 11, 1991.                                            */
/* Oct 11, modified lateral range to start,stop,increment   */
/*                                                                       */
/*************************************************************************/

/* Definition of Constants */

#define erad 6.37e6   /* earth radius */
#define kelvin 273.15   /* add to celcuis to get kelvin */
#define eyel 1.8        /* eye level of observor */
#define minutes 3437.746770784938954435   /* minutes/rad */
#define pi 3.14159265358979323846
#define SIZE 60      /* maximum number of ellipsoids */
#define INDEX(value) (1+(0.0799/((value)+kelvin)))
#define SQR(value) ((value)*(value))
#define SQRT(value) (pow((value),0.5))
#define TRUE 1
#define FALSE 0
#define LARGE 1.0e12  /* radius for a straight line */
```

```
/* Definition of Structures */

struct vector
{
double x;
double y;
double z;
};

struct ellipsoidata
{
double a;
double b;
double c;
double temp;
};  /* a,b,c, equation parameters */

struct planenum
{
double a;
double b;
double c;
double d;
};      /* a,b,c binormal direction numbers of plane ax+by+cz+d=0 */

/* Definition of New Types */

typedef struct vector vec;
typedef int bool;
typedef struct ellipsoidata elp;
typedef struct planenum plane;


/* Function Definitions */

void inputdata(elp [],vec *,double *,double [],int *,FILE *);
/* read input data */
void setupshells(elp [],int *,double *);
/* calculate a b c for each ellipsoid */
void conv_coord(vec *,double *,double [],double *,double *);
/* convert surface coords to cartesian */
bool outside(elp *,vec *,double);
/* determine if point is outside given ellipsoid */
bool on(elp *,vec *,double);
/* determine if point is on a given ellipsoid */
void get_destn(vec *,vec *,double *,double *,vec *,double *,double *);
/* where are we looking to */
```

```
bool line_intercept(elp *,vec *,vec *,vec *,double,double);
/* find intercept with outer ellips */
void dir_nums(vec *,vec *, vec *);
/*find direction nums of line ab */
void ellip_norm(vec *,elp *,vec *,double);
/*-norm direct of ellip at point */
void normalize(vec *);
/* normalize given vector */
void plane_of_sight(vec *,vec *,plane *,vec *);
/* find plane of sight nums */
void sphere_centre(double *,plane *,vec *,vec *,vec *);
/* determine sphere center */
void fit_ellip(vec *,elp [],elp *,int,double);
/* find ellipsoid nums for ellipsoid through point */
void sphere(plane *,vec *,vec *,elp *,elp *,elp [],double *,double *,int,
vec *,bool,double);
/* find new direction for ray path and radius */
void tangent(plane *,vec *,vec *,vec *);
/* find ray path tangent */
void new_plane(plane *,vec *,vec *,vec *);
/* new plane of sight nums */
void angle(vec *,vec *,double *);
/* find angle between ray path tangent and refraction gradient */
void inter_ellip_dist(vec *,vec *,elp [],elp *,double *,double *,int,double);
/* distance from point to next ellipsoid along gradient */
bool grounded(vec *,double *);
/* flag if point below ground level */
bool exor(bool,bool);
/* find exclusive or */
bool hit_object_plane(double *,vec *);
/* determine if we're there */
bool intercept(elp *,vec *,bool,double,plane *,double *,vec *,double);
/* test for intercept with next ellipse */
void sphere_intercept(vec *,vec *,vec *,double *);
/* grad line intercept with ground */
void move(vec *,vec *,plane *,double *);
/* advance the ray along path */
void locate_isotherm(elp *,vec *,bool,vec *,plane *,double *,double,double);
/* find intersection with given ellipse */
void correction(double *,double,plane *,plane *,vec *);
/* average old value with new to provide corrected value */
void find_ground(double,vec *,double *,vec *,plane *,double *);
/* ground find function */


/*************************************************************************/
/* Read in program data
```

```
inputs:         none

outputs:        ellip[] - contains height and temperature profile
obs - locations of observer wrt earth surface
obj - distance along earth surface to object plane
sight - limits of sighting angles in minutes
cnt - number of ellipsoidic shells
*/
/*****************************************************************************/
void inputdata(elp ellip[SIZE],vec *obs,double *obj,double sight[6],int *cnt,
FILE *out)
{
 char input[80];
 int c;
 FILE *fp;
 do
 {
  printf("Enter name of file containing temperature profile(include path)\n");
  scanf("%s",input);
  if ((fp=fopen(input,"r")) != NULL)
  {
   fgets(input,80,fp);
   printf("%s",input);
   fprintf(out,"%s",input);
   fprintf(out,"Enter the a,b coordinates for the outer ellipsoid with a space");
   fprintf(out," between each number\n");
   fscanf(fp,"%lf %lf",&ellip[0].a,&ellip[0].b);
   fprintf(out,"ellip parameters a=%f b=%f\n",ellip[0].a,ellip[0].b);
   fprintf(out,"Enter the coordinates of the observor wrt temperature ");
   fprintf(out,"measurement\nlocation(x y z) with space between numbers.\n");
   fscanf(fp,"%lf %lf %lf",&obs->x,&obs->y,&obs->z);
   fprintf(out,"obs: %f %f %f\n",obs->x,obs->y,obs->z);
   fprintf(out,"Enter distance to object plane from temp measurement loc(+ve)\n");
   fscanf(fp,"%lf",obj);
   fprintf(out,"obj: %f\n",*obj);
   *obj=-*obj;
   fscanf(fp,"%i",cnt);
   if (*cnt>60)
   {
    printf("Number of data points exceeds data limit of 60 pairs\n");
    exit;
   }
   for (c=1;c <=*cnt; c++)
    fscanf(fp,"%lf %lf",&ellip[*cnt-c].temp,&ellip[*cnt-c].c);
   fclose(fp);
  }
```

```
  else
   printf("Error: Couldn't retrieve file.\n");
 } while ( fp == NULL);
 printf("Enter the elevation start, stop, and increment angles(min):\n");
 scanf("%lf %lf %lf",&sight[0],&sight[1],&sight[2]);
 fprintf(out,"elev start:%f stop:%f incr:%f\n",sight[0],sight[1],sight[2]);
 printf("Enter the positive lateral start, stop and increment angles(min):\n");
 scanf("%lf %lf",&sight[3],&sight[4],&sight[5]);
 fprintf(out,"lateral start:%f stop=%f incr:%f\n",sight[3],sight[4],sight[5]);
}

/*************************************************************************/
/* Use data to determine shell parameters in cartesian system. Add a shell at
   zero height if none exists.

inputs:         ellip[] - height and temp profile
cnt - number of shells

outputs:        origin_depth - depth of origin beneath earth's surface
ellip[] - ellipsoid parameters and associated
  temperature
cnt - updated number of shells
Modified July 27 to remove earth height from shell height
Modified Nov 6, 1990 to correct conversion process                    */
/*************************************************************************/
void setupshells(elp ellip[SIZE],int *cnt,double *origin_depth)
{
 double t,t1,t2;
 int c;

 /* add an ellip at height zero if none exists */
 if (ellip[*cnt-1].c!=0.0)
 {
  ellip[*cnt].c=0.0;
  ellip[*cnt].temp=(ellip[*cnt].c-ellip[*cnt-1].c)*(ellip[*cnt-2].temp-
ellip[*cnt-1].temp)/(ellip[*cnt-2].c-ellip[*cnt-1].c)+
ellip[*cnt-1].temp;
 (*cnt)++;
 }

 t=ellip[0].a;    /* x distance */
 t1=ellip[0].b;   /* y distance */
 if (t1 > t)
 {
  *origin_depth= erad*(1-cos(t1/erad));
  ellip[0].b=erad*sin(t1/erad);
  ellip[0].a=erad*sin(t/erad);
```

```
 }
 else
 {
  *origin_depth=erad*(1-cos(t/erad));
  ellip[0].a=erad*sin(t/erad);
  ellip[0].b=erad*sin(t1/erad);
 }
 t=SQR(ellip[0].a)-SQR(ellip[0].c);  /* focal point length */
 t1=SQR(ellip[0].b)-SQR(ellip[0].c);  /* focal point length */
 for (c=1;c<*cnt;c++)
 {
  ellip[c].a = SQRT((SQR(ellip[c].c)+t));
  ellip[c].b = SQRT((SQR(ellip[c].c)+t1));
 }
}

/*************************************************************************/
/* Convert observer location, and object location from earth surface
   distances to cartersian locations. Convert sight angles to radians.

inputs:         obs - observer loaction
obj - object location
sight - angles
origin_depth -

outputs:        obs,obj,sight - modified data
base-angle - angle of earth tangent at observer wrt
to coordinate system
Modified Oct 10, 1991 to correct observer location conversion.

*/
/*************************************************************************/
void conv_coord(vec *obs,double *obj,double sight[6],double *origin_depth,
 double *base_angle)
{
 int c;
 double temp,gamma;

 *base_angle=obs->x/erad; /* calc base angle from arc length and radius */
 gamma=obs->y/erad;
 temp=erad+obs->z;

 obs->x=temp*sin(*base_angle);
 obs->y=temp*cos(*base_angle)*sin(gamma);
 obs->z=temp*cos(*base_angle)*cos(gamma)-erad+*origin_depth;

 *obj=erad*sin(*obj/erad);  /* convert distance along earth's curvature
```

```
 to straight line */

 for (c=0;c<6;c++)
  sight[c]=sight[c]/minutes; /* convert minutes to radians */

}

/*************************************************************************/
/* Test if point outside given ellipsoidic surface
Modified July 19 to use new shell system                              */
/*************************************************************************/
bool outside(elp *shell,vec *position,double origin_depth)
{
 bool flag;
 double zprime;
 zprime=position->z-origin_depth+(SQR(position->x)+SQR(position->y))/2.0/erad;
 if (shell->c==0)
  flag=(fabs(position->x)>shell->a)||(fabs(position->y)>shell->b)||
  (zprime>=0);
 else if (zprime>0.0)
   flag = ((SQR(position->x/shell->a)+SQR(position->y/shell->b)+
    SQR(zprime/shell->c)) > 1.0)||(fabs(position->x)>shell->a);
 else
  flag = (SQR(position->x/shell->a)+SQR(position->y/shell->b) > 1.0);
 return flag;
}

/*************************************************************************/
/* Test if point within tolerance of ellipsoidic surface
Modified July 19 to use new shell system                              */
/*************************************************************************/
bool on(elp *shell,vec *position,double origin_depth)
{
 double test;
 double zprime;
 bool flag;
 zprime=position->z-origin_depth+(SQR(position->x)+SQR(position->y))/2.0/erad;
 if (shell->c==0)
  flag=(fabs(position->x)<shell->a)&&(fabs(position->y)<shell->b)&&
(fabs(zprime)<1.0e-5);
 else
 {
  test = SQR((position->x/shell->a))+SQR((position->y/shell->b))+
    SQR((zprime/shell->c))-1.0;
  flag = (fabs(test) <1.0e-5);
 }
 flag = flag&&(zprime > 0.0); /* must be on upper half of ellipse */
```

```
 return flag;
}

/*************************************************************************/
/* Find point that the eye thinks its seeing.

inputs:         line - tangent vector at point on ray path
hor - horizontal angle of view
ver - vertical angle of view
obs - observer location
obj - location of object plane
base_angle - view offset due to earth curvature

outputs:        destn - location in object plane the eyes believes it
is seeing
*/
/*************************************************************************/
void get_destn(vec *destn,vec *line,double *hor,double *ver,vec *obs,double *obj,
 double *base_angle)
{
 destn->x=*obj;
 destn->y=obs->y+tan(*hor)*(obs->x-*obj)/cos(*ver+*base_angle);
 destn->z=obs->z+tan(*ver+*base_angle)*(obs->x-*obj)/cos(*hor);

 dir_nums(obs,destn,line);
}

/*************************************************************************/
/* Find line intercept of line form observer along angles ver and hor in
   direction line with outer ellip surface

inputs:         ellip - given outer ellip
line - direction numbers
obs - start location of line
origin_depth - earth surface
iterate - step size

outputs:        point - location of intercept
*/
/*************************************************************************/
bool line_intercept(elp *ellip,vec *line,vec *point,vec *obs,
double origin_depth,double iterate)
{
 double t;
 double shift;
 bool test;
```

```
 test=TRUE;
 t=0.0;
 shift=iterate;
 *point=*obs;

 if (outside(ellip,point,origin_depth))
  while(outside(ellip,point,origin_depth)&&(t<erad))
  {
   t+=shift;
   point->x=obs->x+t*line->x;
   point->y=obs->y+t*line->y;
   point->z=obs->z+t*line->z;
  }
 else
  {
   shift=-iterate;
   while(!outside(ellip,point,origin_depth)&&(t>-erad))
   {
    t+=shift;
    point->x=obs->x+t*line->x;
    point->y=obs->y+t*line->y;
    point->z=obs->z+t*line->z;
   }
  }
 shift=iterate;
 if (fabs(t)<erad)
  while(!on(ellip,point,origin_depth))
  {
   shift/=2.0;
   if(outside(ellip,point,origin_depth))
    t+=shift;
   else
    t-=shift;
   point->x=obs->x+t*line->x;
   point->y=obs->y+t*line->y;
   point->z=obs->z+t*line->z;
  }
 else
  test=FALSE;

 return test;
}

/*************************************************************************/
/* Normalized direction numbers fron a to b stored in dir */
/*************************************************************************/
void dir_nums(vec *a,vec *b,vec *dir)
```

```
{
 dir->x=b->x-a->x;
 dir->y=b->y-a->y;
 dir->z=b->z-a->z;
 normalize(dir);
}

/*************************************************************************/
/* Find magnitude of a and normalize its components */
/*************************************************************************/
void normalize(vec *a)
{
 double temp;
 temp=SQRT(SQR(a->x)+SQR(a->y)+SQR(a->z));
 a->x/=temp;
 a->y/=temp;
 a->z/=temp;
}

/*************************************************************************/
/* Normal vector to ellip at point
Modified July 23 to use new shell system

inputs:         pt - point
el - ellip parameters
origin_depth - earth's surface

outputs:        nm - normalized norm vector
*/
/*************************************************************************/
void ellip_norm(vec *pt,elp *el,vec *nm,double origin_depth)
{
 vec temp;
 double z_earth;    /* distance from origin plane to earth surface at pt */
 if (el->c==0.0)
 {
  nm->x=-2*pt->x;
  nm->y=-2*pt->y;
  nm->z=-2*(pt->z-origin_depth+erad);
 }
 else
 {
  z_earth=origin_depth-(SQR(pt->x)+SQR(pt->y))/2/erad;
  nm->x=-2*pt->x/SQR(el->a)-2*(pt->z-z_earth)/SQR(el->c)*pt->x/erad;
  nm->y=-2*pt->y/SQR(el->b)-2*(pt->z-z_earth)/SQR(el->c)*pt->y/erad;
  nm->z=-2*(pt->z-z_earth)/SQR(el->c);
 }
```

```
 normalize(nm);
}

/*************************************************************************/
/* Plane of sight is determined by the binormal directions numbers A,B,C and
   D which is dependent on a point in the plane. Ax+By+Cz+D=0.

inputs:         a - normal to ellip
b - tangent to ray path
pt - point in plane

outputs:        plane - plane parameters A,B,C,D
*/
/*************************************************************************/
void plane_of_sight(vec *a,vec *b,plane *c,vec *pt)
{
 vec temp;
 /* normal*tangent=binormal */
 temp.x=a->y*b->z-a->z*b->y;
 temp.y=a->z*b->x-a->x*b->z;
 temp.z=a->x*b->y-a->y*b->x;
 normalize(&temp);
 c->a=temp.x;
 c->b=temp.y;
 c->c=temp.z;
 c->d=-(c->a)*pt->x-c->b*pt->y-c->c*pt->z;
}

/*************************************************************************/
/* The centre if found by determining a vector perpendicular to the ray path
   in the negative direction. Moving from the point along this vector a total
   of radius determines the centre point.

inputs:         radius - radius of ray path
pl - current plane of sight
line - tangent vector to ray path
pt - location on ray path

outputs:        cnt - centre for ray path
*/
/*************************************************************************/
void sphere_centre(double *radius,plane *pl,vec *line,vec *pt,vec *cnt)
{
 vec ppd; /* perpendicular - local direction of -norm */

 /* evaluate binormal * tan = -norm cross product */
 ppd.x=pl->b*line->z-line->y*pl->c;
```

```
 ppd.y=pl->c*line->x-line->z*pl->a;
 ppd.z=pl->a*line->y-line->x*pl->b;
 normalize(&ppd);

 cnt->x=pt->x+ppd.x*(*radius);
 cnt->y=pt->y+ppd.y*(*radius);
 cnt->z=pt->z+ppd.z*(*radius);
}

/*****************************************************************************/
/* Given a point find the ellip parameters that fit through this point and
   interpolate temperature data to assign a temperature to this ellip.

inputs:         pt - point
base_ellip - array of shells
ellipnum - number of ellipsoids
origin_depth - earth's surface

outputs:        newelp - new ellip data
*/
/*****************************************************************************/
void fit_ellip(vec *pt,elp baseelp[SIZE],elp *newelp,int ellipnum,
double origin_depth)
{
 double t,t1; /* use t,t1 to store foci */
 double shift;
 int c;

 /* focii of ellipsoids */
 t=SQR(baseelp[0].a)-SQR(baseelp[0].c);
 t1=SQR(baseelp[0].b)-SQR(baseelp[0].c);

 c=0;
 /* find ellips surrounding point */
 while((!outside(&baseelp[c],pt,origin_depth)&&(c<ellipnum)))
  c++;
 if (c==0)
  c++;
 else if (c==ellipnum)
 {
  c--;
  shift=baseelp[c].c-baseelp[c-1].c;
 }
 else
  shift=baseelp[c-1].c-baseelp[c].c;
 newelp->a=baseelp[c].a;
 newelp->b=baseelp[c].b;
```

```
 newelp->c=baseelp[c].c;
 newelp->temp=baseelp[c].temp;
 /* move in parameters until pt is on ellip */
 do
 {
  shift/=2.0;
  if (outside(newelp,pt,origin_depth))
   newelp->c+=shift;
  else
  newelp->c-=shift;
  newelp->b=SQRT((SQR(newelp->c)+t1));
  newelp->a=SQRT((SQR(newelp->c)+t));
 } while (!on(newelp,pt,origin_depth));

 newelp->temp=(newelp->c-baseelp[c].c)/(baseelp[c-1].c-baseelp[c].c)*
  (baseelp[c-1].temp-baseelp[c].temp)+baseelp[c].temp;
}

/*************************************************************************/
/* Update variables to reflect new location given pt, curelp, nexelp

outputs:        Updated plos(plane of sight), cnt(centre of ray path),
and radius(of ray path).
*/
/*************************************************************************/
void sphere(plane *plos,vec *cnt,vec *pt,elp *scurelp,elp *snexelp,
 elp ellip[SIZE],double *radius,double *origin_depth,int ellipnum,
 vec *oldtan,bool upward,double iterate)
{
 vec grad; /* ellipsoid gradient direction nums (inward pointing) */
 elp temp;
 double gamma,distance;

 ellip_norm(pt,scurelp,&grad,*origin_depth);  /* ellip normal */
 tangent(plos,pt,oldtan,cnt);  /* ray path tangent */
 new_plane(plos,oldtan,&grad,pt);  /* find plane numbers */
 angle(&grad,oldtan,&gamma);  /* angle between normal and tangent */
 temp=*snexelp;
 inter_ellip_dist(&grad,pt,ellip,&temp,&distance,origin_depth,ellipnum,iterate);
 if (scurelp->temp == temp.temp)
 {
  if (scurelp->c <= temp.c)
   *radius=LARGE;
  else
   *radius=-LARGE;
 }
 else
```

```
  *radius=distance*INDEX(scurelp->temp)/
   (INDEX(temp.temp)-INDEX(scurelp->temp))/sin(gamma);
 if (upward)
  *radius=-*radius;
 sphere_centre(radius,plos,oldtan,pt,cnt);
}

/*************************************************************************/
/* Find tangent to ray path at pt.

inputs:         plos - plane of transit
pt - location on ray path
cnt - centre for ray path

outputs:        tan - tangent vector
*/
/*************************************************************************/
void tangent(plane *plos,vec *pt,vec *tan,vec *cnt)
{
 vec norm; /* sphere norm */

 norm.x=3*pt->x-3*cnt->x;    /* norm is a point arbitrarily extended */
 norm.y=3*pt->y-3*cnt->y;    /* from the centre through pt */
 norm.z=3*pt->z-3*cnt->z;
 normalize(&norm);   /* normalize norm */
 if (cnt->z>pt->z)
 { /* this gives a negative norm which is corrected below */
  norm.x=-norm.x;
  norm.y=-norm.y;
  norm.z=-norm.z;
 }
 /* evaluate b*n=t cross product */
 tan->x=plos->b*norm.z-norm.y*plos->c;
 tan->y=plos->c*norm.x-norm.z*plos->a;
 tan->z=plos->a*norm.y-norm.x*plos->b;
 normalize(tan);
}

/*************************************************************************/
void new_plane(plane *plos,vec *tan,vec *grad,vec *pt)
{
 /* tangent*-ellip.norm=binormal */
 plane_of_sight(tan,grad,plos,pt);
}

/*************************************************************************/
/* Angle between ellip gradient and ray path tangent
```

```
inputs:         grad - gradient vector
tab - tangent vector

outputs:        gamma - angle in radians
*/
/*************************************************************************/
void angle(vec *grad,vec *tan,double *gamma)
{
 /* remember grad is in negative direction */
 double sqrlength,w;
 sqrlength=SQR((grad->x+tan->x))+SQR((grad->y+tan->y))+SQR((grad->z+tan->z));
 w=(2-sqrlength)/2; /* apply cosine law with a=1,b=1,c=sqrlength */
 *gamma=acos(w);
}

/*************************************************************************/
/* Distance between two ellip surfaces along normal vector.

inputs:         grad - normal direction
pt - current location on path
ellip - array of ellip shells
nexelp - next ellip in direction of travel
origin_depth - earth's surface
ellipnum - number of ellipsoids
iterate - step size

outputs:        dist - distance
*/
/*************************************************************************/
void inter_ellip_dist(vec *grad,vec *pt,elp ellip[SIZE],elp *nexelp,
 double *dist,double *origin_depth,int ellipnum,double iterate)
{
 vec inter; /* intercept point */
 line_intercept(nexelp,grad,&inter,pt,*origin_depth,iterate);
/* find intercept of gradient line
  with next elp */
 if (grounded(&inter,origin_depth))
 {
  sphere_intercept(grad,&inter,pt,origin_depth);
  *nexelp=ellip[ellipnum-1];
 }
 *dist=SQRT(SQR((pt->x-inter.x))+SQR((pt->y-inter.y))+SQR((pt->z-inter.z)));
}

/*************************************************************************/
/* Test if point is below earth surface */
```

```
/*************************************************************************/
bool grounded(vec *pt,double *origin_depth)
{
 bool flag;
 flag = ((SQR(pt->x)+SQR(pt->y)+SQR(pt->z-*origin_depth+erad)) < (SQR(erad)));
 return flag;
}

/*************************************************************************/
/* Intercept between a line and a sphere. t is the parameter for the line.

inputs:         grad - direction of line
pt - start of line
origin_depth -  earth's surface

outputs:        inter - location of intercept
*/
/*************************************************************************/
void sphere_intercept(vec *grad,vec *inter,vec *pt,double *origin_depth)
{
 double a,b,c,d,t;
 d=*origin_depth-erad;
 a=SQR(grad->x)+SQR(grad->y)+SQR(grad->z);
 b=2*(grad->x*pt->x+grad->y*pt->y+grad->z*pt->z-grad->z*d);
 c=SQR(pt->x)+SQR(pt->y)+SQR(pt->z)-SQR(erad)+SQR(d)-2*pt->z*d;

 if (grad->z >= 0.0 )           /* gradient pos x dir then t should be pos*/
  t=(-b+SQRT(SQR(b)-4*a*c))/(2*a);
 else
  t=(-b-SQRT(SQR(b)-4*a*c))/(2*a);

 inter->x=pt->x+t*grad->x;
 inter->y=pt->y+t*grad->y;
 inter->z=pt->z+t*grad->z;
}

/*************************************************************************/
bool exor(bool a,bool b)
{
 bool c;
 c=(a && !b) || (!a && b);
 return c;
} /* exor a b */

/*************************************************************************/
/* Test if point past object plane */
/*************************************************************************/
```

```
bool hit_object_plane(double *obj,vec *pt)
{
 bool a;
  if (pt->x <= *obj)
  a=TRUE;
 else
  a=FALSE;
 return a;
}

/*************************************************************************/
/* Move along ray path to new point determined from new x coordinate.

inputs:         pt - current location
cnt - centre of ray path arc
plos - plane of travel of ray path
radius - ray path radius

outputs:        pt - new location
*/
/*************************************************************************/
void move(vec *pt,vec *cnt,plane *plos,double *radius)
{
 double a,b,c;
 a=1+SQR(plos->c/plos->b);
 b=2*(plos->c*(plos->d+cnt->y*plos->b+plos->a*pt->x)-SQR(plos->b)*cnt->z)/
    SQR(plos->b);
 c=SQR(pt->x-cnt->x)+SQR(plos->d/plos->b+plos->a*pt->x/plos->b+cnt->y)-
    SQR(*radius)+SQR(cnt->z);
 /* check if radius is positive, if so curvature is concave earthward */
 /* otherwise it's convave skyward and z is the lesser value */
 if (*radius > 0.0)
  pt->z=(-b+SQRT(SQR(b)-4*a*c))/(2*a);
 else
  pt->z=(-b-SQRT(SQR(b)-4*a*c))/(2*a);
 pt->y=(-plos->d-plos->a*pt->x-plos->c*pt->z )/plos->b;
}
/*************************************************************************/
/* Locate the intersection of ray path and given ellipse

inputs:         ellip - ellip to intersect with
point - current ray path location
upward - direction flag
center - ray path center
plofs - plane of sight
radius - ray path radius
iterate - step size
```

```
origin_depth - earth's surface

outputs:        point - located on ellipse
*/
/*************************************************************************/
void locate_isotherm(elp *ellip,vec *point,bool upward,vec *center,
plane *plofs,double *radius,double iterate,double origin_depth)
{
 double shift;
 shift=iterate;
 while(!on(ellip,point,origin_depth)&&(shift>1.0e-7))
 {
  shift/=2.0;
  if (exor(!outside(ellip,point,origin_depth),upward))
   point->x+=shift;
  else
   point->x-=shift;
  move(point,center,plofs,radius);
 }
}

/*************************************************************************/
/* correction of ray path using predictor/corrector philosophy
  */
/*************************************************************************/
void correction(double *radius,double oldradius,plane *plos,plane *oldplos,
vec *oldpt)
{
 vec temp;
 *radius=(oldradius-*radius)/2.0;
 temp.x=(plos->a+oldplos->a)/2.0;
 temp.y=(plos->b+oldplos->b)/2.0;
 temp.z=(plos->c+oldplos->c)/2.0;
 normalize(&temp);
 plos->a=temp.x;
 plos->b=temp.y;
 plos->c=temp.z;
 plos->d=-(plos->a)*oldpt->x-plos->b*oldpt->y-plos->c*oldpt->z;
}

/*************************************************************************/
/* Test for an intercept with the next ellipse. Also try to reduce chance
   of missing intercept by testing three successively closer points to the
   starting point. If all fail then there was no intercept otherwise declare
   intercept as soon as detected.

inputs:         nexelp - next elp
```

```
point - current test location
upward - direction flag
iterate - step length
centre - path sphere centre
radius - path sphere radius
plos - plane of ray transit
origin_depth earth's surface

outputs:        bool true or false
  */
/*************************************************************************/
bool intercept(elp *nexelp,vec *point,bool upward,double iterate,plane *plos,
double *radius,vec *centre,double origin_depth)
{
 int cnt;
 vec temp;      /* test point */

 if (exor(!outside(nexelp,point,origin_depth),upward))
  return TRUE;
 temp=*point;
 for (cnt=1;cnt<4;cnt++)
 {
  temp.x+=iterate/4;
  move(&temp,centre,plos,radius);
  if (exor(!outside(nexelp,&temp,origin_depth),upward))
  {
   *point=temp;
   return TRUE;
  }
 }
 return FALSE;
}

/*************************************************************************/
/* Convert the coordinates of a point from cartesian system to a point
   relative to the earth's surface.

inputs:         pt - point to be converted
origin_depth - locates earth surface

outputs:        pt - new coordinates
  */
/*************************************************************************/
void Conv_to_earth(vec *pt,double origin_depth)
{
 double temp;
 /* determine height of point above earth's surface */
```

```
 temp=erad-sqrt(SQR(erad)-SQR(pt->x)-SQR(pt->y))-origin_depth;
 pt->z=pt->z+temp;
}

/*************************************************************************/
/* find_ground is a function which successively converges to ground      */
/*************************************************************************/

void find_ground(double iterate,vec *point,double *origin_depth,vec *centre,
plane *plofs,double *radius)
{
 double shift;

 shift=iterate;
 do
 {
  if (grounded(point,origin_depth))
   point->x+=shift;
  else
   point->x-=shift;
  shift/=2;
  move(point,centre,plofs,radius);
 } while(shift>.001);
}
```

Appendix C

3d_plot.c

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <float.h>
#include <conio.h>

/*************************************************************************/
/*
   3d_PLOT.C working date AUG 17, 1990.

   The following are routines that support Halo '88 graphic calls. These
   routines will plot the ray paths in a given bundle of rays.  The view is
   a perspective view projected onto to a 2-d surface.  The view can be
   resolved and rotated to give a clear view of the rays.
   */
/*************************************************************************/

struct vector
{
 double x;
 double y;
 double z;
};

typedef struct vector vec;

void plot(char [],vec far *,int *,int []);
void DrawAxes(float,float,float,float,float,float,float,float);
void RayMenu(float,float,float,float,float,float);
float RoundOff(float);
```

```
void Convert(vec far *,int *,int [25],float,float,float,float);
static float X_RayPlot[25][100],Z_RayPlot[25][100];
static int pattr[]={-1,-1,0,0,1,1,0,0,-1,0,0,-1,0,-1,-1,-1,0};
/* for large plot use 960,576 in pattr[0],[1] */
#define convert 57.29577951 /* convert radians to degrees */



/*************************************************************************/
/* Write title to top of plot display                                    */
/*************************************************************************/

void WriteTitle(char title[])
{
 int mode;
 float x1, x2, y1, y2, tx, ty, height, width;
 static char device[]="HALOHERC.DEV";

 SETDEV(device);
 mode=0;
 INITGRAPHICS(&mode);

 x1=0.0; y1=0.0; x2=640.0; y2=200.0;
 SETWORLD(&x1,&y1,&x2,&y2);

 INQTSIZE(title,&height,&width);
 tx=(x2-width)/2.0;
 ty=182.0;
 MOVTCURABS(&tx,&ty);   /* center */
 BTEXT(title);
 DELTCUR();
}


/*************************************************************************/
/* Display outer box and coordinates axises                              */
/*************************************************************************/


void DrawAxes(float xorgin,float yorgin,float xmax,float ymax,
float phi,float theta,float d,float rho)
{
 int i, border, black, index;
 float x1, x2, y1, y2;
 static char label[5], unit1[]="(Km)", unit2[]="(m)";
```

```
 x1=0.05; y1=0.10; x2=0.95; y2=0.80; border=1; black=-1;
 SETVIEWPORT(&x1,&y1,&x2,&y2,&border,&black);

 SETWORLD(&xorgin,&yorgin,&xmax,&ymax);
 x1=0.0;
 y1=0.0;
 MOVABS(&x1,&y1);
 x1=0.0;
 y1=d*ymax/4*sin(phi)/(ymax/4*cos(phi)+rho);
 LNREL(&x1,&y1);     /* y axis */
 x1=0.0;
 y1=0.0;
 MOVABS(&x1,&y1);
 index=3;
 SETLNSTYLE(&index);
 x1=d*xorgin/4*sin(theta)/(-xorgin/4*sin(phi)*cos(theta)+rho);
 y1=d*-xorgin/4*cos(theta)*cos(phi)/(-xorgin/4*sin(phi)*cos(theta)+rho);
 LNREL(&x1,&y1);     /* x axis */
 x1=0.0;
 y1=0.0;
 MOVABS(&x1,&y1);
 index=1;
 SETLNSTYLE(&index);
 x1=d*-xorgin/4*cos(theta)/(-xorgin/4*sin(theta)*sin(phi)+rho);
 y1=d*-xorgin/4*sin(theta)*cos(phi)/(-xorgin/4*sin(theta)*sin(phi)+rho);
 LNREL(&x1,&y1);     /* z axis */

}

/*************************************************************************/
/* Display menu options and status informations                          */
/*************************************************************************/

void RayMenu(float rho,float d,float phi,float theta,float xscale,
float yscale)
{
 int i, border, black;
 float x1, x2, y1, y2, tx, ty;
 static char list1[]=
 "Q=QUIT P=PRN SCRN A=ASPECT SCALE R=ROTATE OBJT S=SCALE V=VIEW SIZE";
 static char list2[]=
 " Theta=";
 char label[5];

 list2[7]='\0';         /* construct status string */
 gcvt((phi*convert),4,label);
 strcat(list2,label);
```

```
 strcat(list2," Phi=");
 gcvt((theta*convert),4,label);
 strcat(list2,label);
 strcat(list2," Xscale=");
 gcvt(xscale,4,label);
 strcat(list2,label);
 strcat(list2," Yscale=");
 gcvt(yscale,4,label);
 strcat(list2,label);
 strcat(list2," Rho=");
 gcvt(rho,4,label);
 strcat(list2,label);
 strcat(list2," D=");
 gcvt(d,4,label);
 strcat(list2,label);

 x1=0.05; y1=0.82; x2=0.95; y2=0.95; border=1; black=-1;
 SETVIEWPORT(&x1,&y1,&x2,&y2,&border,&black);

 x1=0.0; y1=0.0; x2=220; y2=40;
 SETWORLD(&x1,&y1,&x2,&y2);

 ty=22.0; tx=5.0;
 MOVTCURABS(&tx,&ty);
 BTEXT(list1);
 ty=7.0;
 MOVTCURABS(&tx,&ty);
 BTEXT(list2);

 DELTCUR();

 x1=0.0; y1=0.0; x2=1.0; y2=1.0;
 SETVIEWPORT(&x1,&y1,&x2,&y2,&black,&black);

}


void plot(char title[],vec far *rays,int *NumberOfRay,int NumPoint[])
{
 int result, i, k, PlotPoints;
 float  RoundOff(),aspect;
 char menu, filename[]="HALOEPSN.PRN";
 float d,rho,phi,theta;
 int border,black,x2,y2;
 float xorginRay,yorginRay,xmaxRay,ymaxRay;
```

```
 d=8000.0;
 rho=8000.0;
 phi=1.3;
 theta=0.7;
 xorginRay=0.0;


 for(;;)
 {
  Convert(rays,NumberOfRay,NumPoint,d,rho,theta,phi);
/* convert 3-d data to 2-d data */
  WriteTitle(title);

  if(xorginRay==0.0)
  {
   for(k=0;k<*NumberOfRay;k++)
   {
    for(i=0;i<NumPoint[k];i++)
    {
     if(fabs(X_RayPlot[k][i]) > xorginRay)
      xorginRay=fabs(X_RayPlot[k][i]);
    }
   }
   xorginRay=RoundOff(xorginRay);
   xmaxRay=-xorginRay;

   INQASP(&aspect);
   INQDRANGE(&x2,&y2);
   yorginRay=-xorginRay/aspect*(y2+1)/(x2+1);
   ymaxRay=-yorginRay;  /* aspect scale y axis to match x axis */
  }

  DrawAxes(xorginRay,yorginRay,xmaxRay,ymaxRay,phi,theta,d,rho);

  for(k=0;k<*NumberOfRay;k++)
  {
   MOVABS(&X_RayPlot[k][0],&Z_RayPlot[k][0]);
   PlotPoints=NumPoint[k];
   POLYCABS(X_RayPlot[k],Z_RayPlot[k],&PlotPoints);   /* plot k'th ray */
  }


  RayMenu(rho,d,phi,theta,xorginRay,ymaxRay);   /* update menu and status */
  do
  {
   menu = getch();
   switch (menu)
```

```
{
case 'a':
 /* aspect scale view */
 CLOSEGRAPHICS();
 yorginRay=-xorginRay/aspect*(y2+1)/(x2+1);
 ymaxRay=-yorginRay;
 break;
case 's':
 /* manual scaling of each axis */
 CLOSEGRAPHICS();
 printf("X:%5.1f\nENTER NEW X-AXIS LIMIT (in m)\n",xorginRay);
 scanf("%f",&xorginRay);
 printf("Y:%5.1f\nENTER NEW Y-AXIS LIMIT (in m)\n",ymaxRay);
 scanf("%f",&ymaxRay);
 xmaxRay=-xorginRay;
 yorginRay=-ymaxRay;
 break;
case 'r':
 /* rotate viewing angles */
 CLOSEGRAPHICS();
 printf("Theta:%4.3f\nENTER NEW Theta (in degrees)\n",(phi*convert));
 scanf("%f",&phi);
 phi=phi/convert;
 printf("Phi:%4.3f\nENTER NEW Phi (in degrees)\n",(theta*convert));
 scanf("%f",&theta);
 theta=theta/convert;
 break;
case 'v':
 /* change viewing distances */
 CLOSEGRAPHICS();
 printf("d:%5.0f\nENTER NEW d\n",d);
 scanf("%f",&d);
 printf("rho:%5.0f\nENTER NEW rho\n",rho);
 scanf("%f",&rho);
 break;
case 'p':
 /* print plot */
 SETPRN(filename);
 SETPATTR(pattr);
 GPRINT();
 break;
case 'q':
 CLOSEGRAPHICS();
 return;
 break;
}
inqerr(&i,&k);
```

```
   if (k!=0)
   {
    CLOSEGRAPHICS();
    printf("func err=%d,error =%d\n pa[16]=%d",i,k,pattr[16]);
   }
  } while(menu=='p');
 }
}

/*************************************************************************/
/* Round off a floating point number to an integer and reassign to floating
   point number.                                                         */
/*************************************************************************/

float RoundOff(float InputNumber)
{
 int integer;
 float OutputNumber;
 if(InputNumber > 0.0)
  integer=InputNumber+0.5;
 else
  integer=InputNumber-0.5;
 OutputNumber=integer;
 return OutputNumber;
}

/*************************************************************************/
/* Convert ray points in 3-d to data points in 2-d

inputs:         rays - ray data
NumberOfRay - ray count
NumPoints - number of points in each ray
d - viewing distance
rho - viewing distance
theta,phi - viewing angles

outputs:        X_RayPlot - x axis data
Y_RayPlot - y axis data
   */
/*************************************************************************/

void Convert(vec far*rays,int *NumberOfRay,int NumPoint[25],float d,
float rho,float theta,float phi)
{
 int a,b,cnt;
 double s1,s2,c1,c2;
 double xe,ye,ze;
```

```
 s1=sin(theta);
 c1=cos(theta);
 s2=sin(phi);
 c2=cos(phi);
 for (a=0,cnt=0;a<*NumberOfRay;a++)
  for (b=0;b<NumPoint[a];b++)
  {
   xe=rays[cnt].x*s1-rays[cnt].y*c1;
   ye=-rays[cnt].x*c1*c2-rays[cnt].y*s1*c2+rays[cnt].z*s2;
   ze=-rays[cnt].x*s2*c1-rays[cnt].y*s1*s2-rays[cnt++].z*c2+rho;
   X_RayPlot[a][b]=(float) (d)*xe/ze;
   Z_RayPlot[a][b]=(float) (d)*ye/ze;
  }
}
```

Appendix D

DATA SETS

```
Comment line describing data set and relevant information
Outer Shell Size Length Outer Shell Size Width
Observor Location (X Y Z)
Object Plane Location (Y)
Number of Temperature Profile Points
Temperature        Height (in increasing height order)
```

Table D.1: **Data Set File Template**

```
set1.dat from Isaak
5000 1000
5000 0 1.8
2500
2
0       0
6.0     30
```

Table D.2: **Data Set SET1**

```
Set #2 5000 1000 phi -2 24 2 theta 240 2 obs 5000 0 1.8 obj -2500
5000 1000
5000 0 1.8
2500
7
0.0     0.0
1.5     11
2.5     15
7.0     20
12.0    21
13.0    22
15.0    30
```

Table D.3: **Data Set SET2**

```
Set #3 5000 1000 phi -2 24 2 theta 240 2 obs 5000 0 1.8 obj -2500
5000 1000
5000 0 1.8
2500
7
8.0     0.0
8.0     11
8.0     15
8.0     20
8.0     21
8.0     22
8.0     30
```

Table D.4: **Data Set SET3**

```
Test Data Set #4 5000 1000 obs 5350 0 1.8 obj -5350
5000 1000
5350 0 1.8
5350
7
0.0     0.0
1.5     11
2.5     15
7.0     20
12.0    21
13.0    22
15.0    30
```

Table D.5: **Data Set SET4**

```
set6.dat using function 2.0e-z-0.02z+30
100000 100000
0 0 4
8000
17
32      0
31.8    0.1
31.6    0.2
31.47   0.3
31.33   0.4
31.2    0.5
31.08   0.6
30.98   0.7
30.88   0.8
30.80   0.9
30.7    1
30.0    3
29.88   6
29.8    10
29.6    20
29.4    30
29.2    40
```

Table D.6: **Data Set SET6**

```
PROFILE JM3  (JM1 WITH EXTRA POINTS, RE DT/DZ SENSITIVITY)
100000 100000
0 0 4
21500
  18
     0.00      0.0
     0.30      5.0
     0.75     10.0
     0.88     11.0
     1.06     12.0
     1.26     13.0
     1.50     14.0
     2.40     17.0
     3.85     20.0
     6.00     24.0
     7.10     26.0
     8.37     30.0
     9.18     34.0
     9.60     39.0
     9.80     45.0
     9.80     60.0
     9.68     80.0
     9.56    100.0
```

Table D.7: **Data Set SETJM3**

```
Data Set for 79-4-27
17000 3000
10000 0 2.5
10000
22
-2.39   0
-2.0    2.48
-1.43   10
-0.98   13.4
-0.60   14.8
0.08    17.7
0.3     18.7
0.4558  19.361
1.1047  21.4687
1.8589  23.4820
2.6460  25.3026
3.5198  27.0575
4.1135  28.0714
4.7405  29.0405
5.5020  30.1363
6.4160  31.3440
7.3870  32.4983
7.8598  32.9960
8.4560  33.9852
8.6889  34.3555
8.8712  35.0109
9.3400  46.9000
```

Table D.8: **Data Set 79-4-27**

```
TB8: linear + parabolic temp profile to reconstruct slide 79-4-20:step 1km
17000 3000
10000 0 2.5
10000
 46
    -2.30       0.00
    -2.0        2.498
    -1.81625    5.000
    -1.44125   10.000
    -1.06625   15.000
    -0.69125   20.000
    -0.31625   25.000
     0.05875   30.000
     0.43375   35.000
     0.82000   40.150
     0.84944   40.400
     0.93774   40.640
     1.08492   40.900
     1.29096   41.150
     1.55588   41.400
     1.87966   41.650
     2.26232   41.900
     2.70384   42.150
     3.20424   42.400
     3.76350   42.650
     4.38164   42.900
     5.05865   43.150
     5.79452   43.400
     6.58927   43.650
     7.44288   43.900
     8.35537   44.150
     9.32673   44.400
     9.9861    44.56
    10.0374    44.59
    10.1845    44.70
    10.293     44.80
    10.44      44.95
    10.575     45.10
    10.745     45.30
    10.903     45.50
    11.055     45.70
    11.203     45.90
    11.333     46.10
    11.463     46.40
    11.57      46.80
    11.67      47.50
    11.75      49.00
    11.75      50.00
    11.69      60.00
    11.57      80.00
    11.45     100.00
```

Table D.9: **Data Set TB8-79-4**

```
tu8 Sun Mirage data set 1
6000000 6000000
26400 0 27.98
26400
48
        -1.32   0
        -1.32   8
        -1.35   12
        -1.39   16
        -1.49   20
        -1.57   22
        -1.66   24
        -1.75   26
        -1.85   28
        -1.95   30
        -2.04   32
        -2.08   34
        -2.08   35
        -2.08   36
        -2.08   36.6
        -2.03   37.2
        -1.935  37.6
        -1.73   38
        -1.44   38.2
        -1.0    38.4
        -0.12   38.8
        0.76    39.2
        1.66    39.6
        1.97    40
        2.10    40.4
        2.21    40.8
        2.35    41.4
        2.47    42
        2.64    43
        2.78    44
        3.03    46
        3.44    50
        3.84    54
        4.44    60
        4.84    64
        5.44    70
        6.94    85
        8.43    100
        9.43    110
        10.28   120
        10.83   130
        11.15   140
        11.23   150
        11.23   160
        11.00   200
        10.40   300
        9.20    500
        7.40    800
```

Table D.10: **Data Set TU8**

```
TG4 Sun Mirage data set 2
6000000 6000000
0 0 100.1
54000
40
        -1.20   0
        -1.33   16
        -1.44   32
        -1.59   52
        -1.71   68
        -1.80   80
        -1.85   88
        -1.88   92
        -1.92   96
        -1.94   100
        -1.94   102
        -1.92   104
        -1.88   106
        -1.82   108
        -1.69   110
        -1.32   112.8
        -0.93   114.4
        -0.37   116
        0.55    117.6
        1.27    118.4
        1.65    118.8
        1.92    119.2
        2.33    120
        2.6     121.2
        2.88    123.2
        3.08    125.2
        3.28    128
        3.49    132
        3.67    136
        3.85    142
        3.87    150
        3.766   166
        3.636   186
        3.376   226
        2.856   306
        2.206   406
        1.556   506
        0.906   606
        -0.394  806
        -2.344  1106
```

Table D.11: **Data Set TG4**

Appendix E

mir.c

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "ansi.h"
#include "itexpfg.h"
#include "stdtyp.h"
/* Working date Sept 18, 1991*/
/*
Mirage Image Simulation written by Wes J Friesn
This program was written in partial fulfillment of the requirements
for the degree Master of Science in Electrical Engineering.
Done under the supervision of Professor W Lehn.
*/


/* definition section to declare constants */
#define HOR 511   /* image size horizontally 512 */
#define VER 511   /* image size vertically 512 */
#define SQR(value) pow((value),2.0)
#define TRUE -1
#define FALSE -1

/* declare variables */

char file_name[35];   /* maximum filename with paths is 35 char */
```

```
struct vector
{
 double x;
 double y;
 double z;
};

typedef struct vector vec;
typedef int bool;

unsigned char cross[10];  /* array to store data cursor over writes */
unsigned char huge *pic;  /* pointer to picture buffer */
unsigned char huge *p; /* pointers used for buffer access */
int mode; /* mode 0 is for Jack Sill's Board mode 1 is for itex board */
void DISPLAY2(unsigned char huge *);  /* display on itex board */
void DISPLAY1(unsigned char huge *);  /* display on Sill board */
void DISPLAY(int);

void Init_6845(void);  /* configure video board */
void Write_port(unsigned int,unsigned int),Get_picture(void);
void Getscale(int *,int *,float *);
void Marker(int,int);
void unMarker(int,int);
void Gettc(int [],int *,vec *,double *,double *);
void Map_(int *,int *,int [],float *,vec *,double *);
void interpolate(double,double,double,vec,vec,vec,vec,vec,
vec,vec,vec,double *,double *,double *);
bool end_of_region(double,double,int *,int,bool,double *,double);
bool belowz(vec,vec,double,double);
bool belowx(vec,vec,double,double);
int roundoff(float);

vec map[15][2][50];   /* array of transfer characteristic */

main()
{
  int refel; /* reference elevation */
  int hor; /* horizon */
  int nor[20]; /* # of rays in transfer characteristic */
  int nog; /* # of groups in transfer map */
  float scale; /* number of meters per line in image */
  double obj; /* object location */
  vec obs; /* observer location */
  double orgn_dpth; /* depth of origin below earth's surface */
  int cnt1;
  FILE *fp; /* file pointer */
  char command;
```

```
  char cmdline[35];


  /* set up buffers */
  pic=(unsigned char huge *)halloc(262144L,sizeof(char));
  if (pic == NULL)
  {
   printf("Insufficient memory for buffer\n");
   for(;;); /* loop infinitely */
  }
  do /* set display mode */
  {
   printf("Select display mode <0> for Sill' board <1> for itex board\n");
   scanf("%d",&mode);
  } while (mode != 0 && mode != 1);
  Init_6845(); /* initialize video stuff */
  Get_picture(); /* load picture */
  DISPLAY(mode); /* display on screen */
  Getscale(&refel,&hor,&scale);
  do /* Processing loop */
  {
   Gettc(nor,&nog,&obs,&obj,&orgn_dpth);
   Map_(&hor,&nog,nor,&scale,&obs,&obj); /* apply mapping */
   CLR_SCRN;
   printf("Loading image...\n"); /* load resulting image */
   fp=fopen(file_name,"rb");
   p=pic;
   for (cnt1=0;cnt1<8;cnt1++)
   { /* loads in 32k sections */
    fread(p,sizeof(char),32768,fp);
    printf("Sec=%d\n",cnt1);
    CUR_UP(1);
    p=(p+32768L);
   }
   fclose(fp);
   CLR_SCRN;
   DISPLAY(mode);
   printf("\nWould you like to process another transfer characteristic? <Y/N>");
   do
    command= (char) tolower (getchar());
   while (command!='y' && command != 'n');
  if (command == 'y')
   Get_picture(); /* reload source for next tc */
  } while (command == 'y');
  hfree(pic);
  return 0;
}
```

```
void Init_6845()
{
  char x;
  if (mode == 0)
  {
   x =(char) inp(772);
   Write_port(0,157);     /* initialize the display board */
   Write_port(1,128);     /*port  read             write             */
   Write_port(2,140);     /*768 reset addr cntrs   crtc addr reg     */
   Write_port(3,102);     /*769 n/a                crtc data reg     */
   Write_port(4,31);      /*770 read image latch   write image latch */
   Write_port(5,5);       /*772 "read" display md    n/a             */
   Write_port(6,29);      /*774 "write" display md   n/a             */
   Write_port(7,30);
   Write_port(8,3);
   Write_port(9,14);
   Write_port(10,0);
   Write_port(11,0);
   Write_port(12,0);
   Write_port(13,0);
   Write_port(14,0);
   Write_port(15,0);
  }
  else
  {
   sethdw(0x300,0xD0000L,SINGLE); /* set address and mode */
   setdim(1024,512,8); /* set dimensions */
   fgon();
   initialize();
   sclear(0); /* blank screen */
  }
}
void Write_port(a,b)
unsigned int a,b;
{
  outp(768,a);         /* reset counters */
  outp(769,b);         /* offset first count */
}

void Map_(int *hor,int *nog,int nor[20],float *scale,vec *obs,double *obj)
{
 float zmax,zmin,xmax;
 double ym,zm,xm;
 double xob,yob,zob;
 double xscale,yscale;
 double low_z,low_x,high_x;
```

```
int ymaxi,ymini,zmaxi,zmini,xmaxi;
int a,b,d,diff,z,y;
int vert1,vert2,hort1,hort2;
int xline,yline,zline;
int rows;
long i;
bool foreground;
char str[HOR+1];
FILE *fp;

memset(str,255,HOR+1); /* white line */
xmax=(float) map[0][0][0].x;
zmax=zmin=(float) map[0][0][nor[0]].z;
*obj=(double) (roundoff(100000*(*obj))/100000.0);

for (a=0;a<=*nog;a++)        /* determine region covered by apparent map */
 for (b=0;b<=nor[a];b++)
 {
  if (zmax < map[a][0][b].z)  /* break z region into two areas */
   zmax=(float) map[a][0][b].z;
                              /* first area is normal object plane mapping */
  if (zmin > map[a][0][b].z&&(map[a][0][b].x-*obj)<1.0)
   zmin=(float) map[a][0][b].z;
  if (xmax < map[a][0][b].x)  /* second area is foreground remapping */
   xmax=(float) map[a][0][b].x;
 }
ymini=-(VER+1)/2;           /* convert to screen coordinates */
ymaxi=(VER+1)/2;
zmaxi=(int) (zmax/(*scale));
zmini=(int) (zmin/(*scale));
xmax=xmax-*obj;
xmax=-xmax/(obs->x-*obj)*(HOR-*hor);
xmaxi=(int) xmax;
if ((zmaxi-zmini)>(HOR))
 zmaxi=zmini+(HOR);
if ((zmaxi-xmaxi-zmini) > HOR)  /* limit z extent to screen height */
{
 xmaxi/=2;
 zmaxi=HOR+xmaxi+zmini;
}
printf("zmini=%d zmaxi=%d xmaxi=%d\n",zmini,zmaxi,xmaxi);
/* start the mapping prodedure */
fp=fopen(file_name,"wb"); /* open result file */
p=pic;
diff=HOR-zmaxi+xmaxi+zmini;
printf("diff=%d\n",diff);
if (diff < HOR && diff > 0)
```

```
   for (z=0;z<diff/2;z++)
    fwrite(str,1,HOR+1,fp); /* fill above image */
/*************************************/
 zm=zmaxi*(*scale);
 xm=*obj;
 xscale=1;
 rows=diff;
 foreground=FALSE;
 for (b=nor[0];b>0;b--) /* moved down regions to first region
in image limits */
   if (!end_of_region(zm,xm,nog,b,foreground,obj,xscale))
    break;
 do
 {
  printf("layer change\n");
  low_z=zm;
  low_x=*obj;
  high_x=*obj;
  for (a=*nog;a>0;a--)
  {
  if (map[a][0][b-1].z < low_z) /* find x and z extremes of region */
    low_z=map[a][0][b-1].z;
  if (map[a][0][b-1].x > high_x)
    high_x=map[a][0][b-1].x;
  if (map[a][0][b].x > low_x)
    low_x=map[a][0][b].x;
  }
  xscale=(high_x-low_x)/(zm-low_z)*(*scale); /* foreground in sky scale*/
  xm=low_x;
   do
   {
    y=ymini; /* start row */
    for (a=*nog;a>0;a--)
    {
     foreground=FALSE;
     hort1=b;
     vert1=a;
     hort2=b-1;
     vert2=a-1;
     /* if any of surrounding points is not a object plane point then consider
        the group to be foreground */
     if ((map[vert1][0][hort1].x > *obj)||
     (map[vert1][0][hort2].x > *obj) ||
     (map[vert2][0][hort2].x > *obj) ||
     (map[vert2][0][hort1].x > *obj))
      foreground=TRUE;
     if (!foreground)
```

```
    { /* all points in object plane */
     ym=y*(*scale);
     while ((map[vert2][0][hort2].y >= ym || map[vert2][0][hort1].y >= ym)&&
       (map[vert1][0][hort1].y <= ym || map[vert1][0][hort2].y <= ym)&&
       y<ymaxi)
     { /* use surrounding group */
     if (!belowz(map[vert1][0][hort2],map[vert2][0][hort2],ym,zm)||
       map[vert1][0][hort2-1].x!=*obj||map[vert2][0][hort2-1].x!=*obj)
      interpolate(0.0,ym,zm,map[vert1][0][hort1],map[vert1][1][hort1],
map[vert1][0][hort2],map[vert1][1][hort2],
        map[vert2][0][hort2],map[vert2][1][hort2],
        map[vert2][0][hort1],map[vert2][1][hort1],
        &xob,&yob,&zob);
     else /* use group below to do this point */
      interpolate(0.0,ym,zm,map[vert1][0][hort2],map[vert1][1][hort2],
        map[vert1][0][hort2-1],map[vert1][1][hort2-1],
        map[vert2][0][hort2-1],map[vert2][1][hort2-1],
        map[vert2][0][hort2],map[vert2][1][hort2],
        &xob,&yob,&zob);
      yline=roundoff((yob/(*scale)))+((VER+1)/2); /* convert to pixels*/
      zline=*hor-roundoff((zob/(*scale)));
      if ( yline <0 || yline > VER || zline <0 || zline > HOR)
       fputc(255,fp);   /* point is outside known area */
      else
      {
       i=(long) zline*(HOR+1);
       i+=yline;
       fputc(*(p+i),fp); /* output corresponding pixel to result file*/
      } /* transfer pixel */
      y++; /* increment column*/
      ym=y*(*scale);
     }
    }  /* object plane */
   else
    { /* foreground case */
     /* given xm */
     if (zm < ((zmini+1)*(*scale))) /* foreground scales*/
       xscale=(obs->x-*obj)/(HOR-*hor);
     yscale=(*scale)*(obs->x-xm)/(obs->x-*obj);
     ym=y*yscale;
     while ((map[vert2][0][hort2].y >= ym || map[vert2][0][hort1].y >= ym)&&
        (map[vert1][0][hort1].y <= ym || map[vert1][0][hort2].y <= ym)&&
       y<ymaxi)
     { /* loop till out of this column group */
     if (!belowx(map[vert1][0][hort2],map[vert2][0][hort2],ym,xm)||hort1<2)
      interpolate(xm,ym,0.0,map[vert1][0][hort1],map[vert1][1][hort1],
       map[vert1][0][hort2],map[vert1][1][hort2],
```

```
          map[vert2][0][hort2],map[vert2][1][hort2],
          map[vert2][0][hort1],map[vert2][1][hort1],
          &xob,&yob,&zob);
         else /* use group below */
         interpolate(xm,ym,0.0,map[vert1][0][hort2],map[vert1][1][hort2],
          map[vert1][0][hort1-2],map[vert1][1][hort1-2],
          map[vert2][0][hort2-1],map[vert2][1][hort2-1],
          map[vert2][0][hort2],map[vert2][1][hort2],
          &xob,&yob,&zob);
         yline=roundoff((yob/((*scale)*(obs->x-xob)/(obs->x-*obj))))+((VER+1)/2);
         xline=*hor+roundoff(((xob-*obj)/((obs->x-*obj)/(HOR-*hor))));
         if ( yline <0 || yline > VER || xline <0 || xline > HOR )
          fputc(255,fp);   /* point is outside known area */
         else
         {
          i=(long) xline*(HOR+1);
          i+=yline;
          fputc(*(p+i),fp); /* output pixel */
         } /* transfer pixel */
         y++; /* increment column */
         ym=yscale*y;
        }
       } /* end of foreground case */
    } /* column counter loop */
    xm+=xscale; /* increment row */
    zm-=*scale;
    rows+=1; /* row count */
   } while(!end_of_region(zm,xm,nog,b,foreground,obj,xscale) && (rows<=HOR));
   /* while region loop */
   printf("rows=%d\n",rows);
   b-=1;
} while(b > 0 && (rows<=HOR));  /* row counter loop */
/*****************************************************/
 if (diff < HOR && diff >0)
  {
   diff= roundoff(diff/2.0);
   for (z=0;z<diff;z++)
    fwrite(str,1,(HOR+1),fp); /* fill below image */
  }
 fclose(fp); /* all done */
}

bool end_of_region(zm,xm,nog,b,foreground,obj,xscale)
double zm;
double xm;
int *nog;
int b;
```

```
bool foreground;
double *obj;
double xscale;
{
 int a;
 bool flag;

 flag=TRUE; /* assume end of region */
 if (!foreground)
 {
  for (a=0;a <= *nog;a++)
   if(zm >= map[a][0][b-1].z) /* zm above all lower region pts */
    flag=FALSE;
 }
 else
 {
  for (a=0;a <= *nog;a++) /* xm region cases */
    if(((xm <= map[a][0][b-1].x)&&(xscale>0.0))||
(xscale<0.0&&xm>=map[a][0][b-1].x))
     flag=FALSE;
 }
 return flag;
}

bool belowz(pt1,pt2,ym,zm)
vec pt1;
vec pt2;
double ym,zm;
{
 double y;
 bool flag;

 y=(pt2.y-pt1.y)/(pt2.z-pt1.z)*(zm-pt1.z)+pt1.y;
 if (pt2.z >pt1.z)
  flag=ym>y; /* current point above surrounding group boundry*/
 else
  flag=ym<y; /* below boundry */

 return flag;
}


bool belowx(pt1,pt2,ym,xm)
vec pt1,pt2;
double ym,xm;
{
 double y;
```

```
 bool flag;

 y=(pt2.y-pt1.y)/(pt2.x-pt1.x)*(xm-pt1.x)+pt1.y;
 if (pt2.x >pt1.x)
  flag=ym>y;    /* current point above surrounding group boundry*/
 else
  flag=ym<y; /* below boundry */

 return flag;
}

void interpolate(xm,ym,zm,pt10,pt11,pt20,pt21,pt30,pt31,pt40,pt41,xob,yob,zob)
double xm;
double ym;
double zm;
vec pt10,pt11,pt20,pt21,pt30,pt31,pt40,pt41;
double *xob;
double *yob;
double *zob;
{
 double e,f,g,h;
 double ym1,ym2,ym3,ym4;
 double zm1,zm2,zm3,zm4;
 double xm1,xm2,xm3,xm4;

  g=(ym-pt40.y)/(pt10.y-pt40.y);
  h=(ym-pt30.y)/(pt20.y-pt30.y);
  ym1=g*pt11.y+(1-g)*pt41.y;
  ym2=h*pt21.y+(1-h)*pt31.y;

 if (xm==0.0)
 {
  e=(zm-pt20.z)/(pt10.z-pt20.z);
  f=(zm-pt30.z)/(pt40.z-pt30.z);
  ym3=e*pt11.y+(1-e)*pt21.y; /* actual perimeter points*/
  ym4=f*pt41.y+(1-f)*pt31.y;
  zm1=g*pt11.z+(1-g)*pt41.z;
  zm2=h*pt21.z+(1-h)*pt31.z;
  zm3=e*pt11.z+(1-e)*pt21.z;
  zm4=f*pt41.z+(1-f)*pt31.z;
  *zob=(zm3*(ym4-ym3)*(zm2-zm1)-zm1*(ym2-ym1)*(zm4-zm3)-(ym3-ym1)*(zm4-zm3)*
        (zm2-zm1))/((ym4-ym3)*(zm2-zm1)-(ym2-ym1)*(zm4-zm3));
  *yob=ym1+(ym2-ym1)*(*zob-zm1)/(zm2-zm1);
  *xob=0.0;
 }
 if (zm==0.0 && !(xm==0.0))
 {
```

```
  e=(xm-pt20.x)/(pt10.x-pt20.x);
  f=(xm-pt30.x)/(pt40.x-pt30.x);
  ym3=e*pt11.y+(1-e)*pt21.y; /* actual perimeter points*/
  ym4=f*pt41.y+(1-f)*pt31.y;
  xm1=g*pt11.x+(1-g)*pt41.x;
  xm2=h*pt21.x+(1-h)*pt31.x;
  xm3=e*pt11.x+(1-e)*pt21.x;
  xm4=f*pt41.x+(1-f)*pt31.x;
  *xob=(xm3*(ym4-ym3)*(xm2-xm1)-xm1*(ym2-ym1)*(xm4-xm3)-(ym3-ym1)*(xm4-xm3)*
        (xm2-xm1))/((ym4-ym3)*(xm2-xm1)-(ym2-ym1)*(xm4-xm3));
  *yob=ym1+(ym2-ym1)*(*xob-xm1)/(xm2-xm1);
  *zob=0.0;
 }
}

void Gettc(int nor[20],int *nog,vec *obs,double *obj,double *orgn_dpth)
{
 char chrstr[90];
 char c;
 char *str;
 double tempz;
 int result;
 int a,b;
 FILE *fp;
 CLR_SCRN;
 do
 {
 printf("Enter transfer characteristic file name.\n");
 scanf("%s",file_name);
 if ((fp=fopen(file_name,"r")) != NULL) /* check for exsistance*/
  {
   do
   {
   fgets(chrstr,90,fp);
   } while (strstr(chrstr,"Depth") == NULL);
   str=strrchr(chrstr,'=');
   str++;
   *orgn_dpth=atof(str);
   fscanf(fp,"Observer:%lf %lf %lf (m)\n",&obs->x,&obs->y,&obs->z);
   fscanf(fp,"Object Plane:%lf (m)\n",obj);
   do
   {
   if (fgets(chrstr,90,fp)== NULL) /* search for start of map data */
   {
    printf("NULL");
    a=ferror(fp);
    a=feof(fp);
```

```
   }
   } while (strstr(chrstr,"x") == NULL);
   a=0;
   do /* read groups */
   {
    fgets(chrstr,90,fp);
    b=0;
    do /* read region */
    {
     fscanf(fp,"%lf %lf %lf %lf %lf %lf",&map[a][0][b].x,&map[a][0][b].y,
 &map[a][0][b].z,&map[a][1][b].x,&map[a][1][b].y,&map[a][1][b].z);
     c=(char) fgetc(fp);
     if (c == '\n')
      c=(char) fgetc(fp);
     ungetc(c,fp);
     b++;
    } while (c != 'H'&&!feof(fp));
    nor[a]=b-1;
    a++;
    c=(char) fgetc(fp);
    if (c == '\n')
     c=(char) fgetc(fp);
    if (!feof(fp))
     ungetc(c,fp);
  } while(!feof(fp));  /* until end of file */
  *nog=a-1;
  fclose(fp);
  for(b=0;b<35;b++)
   if (file_name[b]=='.')
    file_name[b]='\0';
  strcat(file_name,".pic");
 }
 else
  printf("File was not found.\n");
 } while (fp==NULL);
 for (a=0;a <= *nog;a++) /* show mapping data */
  for (b=0;b <= nor[a];b++)
   printf("%f %f %f\t%f %f %f\n",map[a][0][b].x,map[a][0][b].y,
    map[a][0][b].z,map[a][1][b].x,map[a][1][b].y,map[a][1][b].z);

 printf("\nFile loaded.\n");
}

int roundoff(float number)
{
 int integer;
 if (number >(float) 0.0)
```

```
integer=(int) (number+0.5);
 else
integer=(int) (number-0.5);
 return integer;
}
void Getscale(int *refel,int *hor,float *scale)
{
 int x,y;
 float elev;
 char command;
 CLR_SCRN;
 printf("Next you must select the reference elevation ");
 printf("for this picture.  To do this inputrow and column numbers");
 printf(" with a space between in the region of the reference\nelevation.");
 printf(" Then refine these numbers until you locate the desired point.");
 printf("\nEnter <-1 -1> once you are done.\n");
 for(;;)
 { /* get reference level */
  CUR_MV(6,0);
  CLR_LINE;
  scanf("%d %d",&y,&x);
  if (y==-1)    /* keep cursor on image */
   break;
  if (x < 0)
   x=0;
  if (x > VER)
   x=511;
  if (y <0)
   y=0;
  if (y>HOR)
   y=511;
  Marker(x,y); /* add cursor to image */
  DISPLAY(mode);
  unMarker(x,y);  /* remove cursor and replace image covered */
  *refel=y;
 } /* get horizon level */
 CUR_MV(6,0);
 printf("Reference Elevation = %d\n",*refel);
 printf("\nNext find the Horizon Elevation using the same procedure.\n");
 for (;;)
 {
  CUR_MV(9,0);
  CLR_LINE;
  scanf("%d %d",&y,&x);
  if (y==-1)
   break;
  if (x < 0)
```

```
   x=0;
  if (x > VER)
   x=511;
  if (y <0)
   y=0;
  if (y>HOR)
   y=511;
  Marker(x,y);
  DISPLAY(mode);
  unMarker(x,y);
  *hor=y;
 }
 printf("\nHorizon = %d \n",*hor);
 printf("\nEnter the height of the reference elevation in meters > ");
 scanf("%f",&elev);
 *scale=elev/(*hor-*refel);
 printf("Scale is %f meters per line.\n",*scale);
 printf("Press <Enter> to continue....\n");
 command=(char) getch();
}

void Marker(int x,int y)
{ /* save image data and replace with bright pixels*/
 int a,b;
 long c;
 c=(long)(y);
 p=pic;
 p+=(512*(c-2))+x;
 for (a=0;a<5;a++)
 {
  cross[a]=*p;    /* save image data */
  if (a !=2)
   *p=255;   /* replace with cursor */
  else
   for (b=0;b<5;b++)
   {
    cross[b+5]=*(p+b-2);    /* save image data */
    *(p+b-2)=255;       /* replace with cursor */
   }
  p=p+512;
 }
}

void unMarker(int x,int y)
{
 int a,b;
 long c;
```

```
 c=(long)(y);
 p=pic;
 /* reverse Marker process */
 p+=(512*(c-2))+x;
 for (a=0;a<5;a++)
 {
  if (a !=2)
   *p=cross[a];
  else
   for (b=0;b<5;b++)
    *(p+b-2)=cross[b+5];
  p=p+512;
 }
}

void Get_picture()
{
 FILE *fp;
 int cnt1;
 p=pic;
 CLR_SCRN;
 do
 {
  printf("Enter the file name to be loaded...\n");
  scanf("%s",file_name);
  if ((fp=fopen(file_name,"rb")) != NULL)
  {
   printf("File opened\n");
   for (cnt1=0;cnt1<8;cnt1++)
   {
    fread(p,sizeof(char),32768,fp);  /* read picture into buffer */
    printf("Sec=%d\n",cnt1);
    CUR_UP(1);
    p=(p+32768L);
   }
   fclose(fp);
   DISPLAY(mode);       /* call display routine */
  }
  else
  {
   CLR_SCRN;
   printf("Error: Couldn't retrieve file.\n");
  }
 } while (fp == NULL);
}
/* itex board routine writes a row at a time */
void DISPLAY(int mode)
```

```
{
 if (mode == 0)
  DISPLAY1(pic);
 else
  DISPLAY2(pic);
}
/* assembly language display routine */
void DISPLAY2(unsigned char huge *pic)
{
 int a,b;
 p=pic;
 sclear(0);
 for (a=0;a<=VER;a++)
 {
  bwhline(64,a,512,p);
  p+=512;
 }
}

void DISPLAY1(unsigned char huge *pic)
{

 _asm \
 {
 ; okay lets load the pointer
           LES         DI,pic
 ;
           MOV         DX,768              ;point to counter port
           IN          AL,DX               ;reset counter
           MOV         DX,774              ;point to write mode port
           IN          AL,DX               ;set to write
           MOV         CX,0000H            ;initialize count
           MOV         DX,770              ;point to data latch
 ;for large arrays BX always starts with 0000H
 LP1:      MOV         AL,ES:[DI]          ;read array
   OUT        DX,AL              ;send to port
   NOP                           ;wait
           INC         DI                  ;next location
           JNZ         LP1                 ;loop up to FFFF
           MOV         AX,ES               ;copy DS
           ADD         AX,1000H            ;add increment
           MOV         ES,AX               ;return DS
           INC         CX                  ;increment outer loop
           CMP         CX,0004H            ;check count
           JL          LP1                 ;finished?
           MOV         DX,772              ;point to read mode port
           IN          AL,DX               ;set to display
```

```
 }
}
```

Appendix F

User's Guide

F.1 3d_ray

3d_ray requires both commandline entries and a data file containing a temperature profile. The data file format is outlined in Appendix D along with several examples of such files. The following questions are asked:

```
Would you like to plot the ray paths? 0=NO

RAY TRACING WITH HALO
HAS HERC GRAPHICS MODE BEEN SET ? 0=NO

Would you like to see the ray points? 0=NO

Enter iteration step size(in m).

Enter name of file containing temperature profile(include path)

Enter the elevation start, stop, and increment angles(min):

Enter the positive lateral start, stop and increment angles(min):
```

The first choice is whether to plot the ray paths. Entering 0 sets the plot flag to false and any other number will set the flag to true. If plotting was selected the next

prompt is a reminder to set the computer to a compatible graphics mode. Entering 0 here will terminate the program and allow the machine to be correctly configured. Any other number will continue the program.

The next question checks if ray points are to be listed. Selecting 0 sets this flag to false. Any other entry sets the flag to true. If either the plotting flag or ray points flag is true than a special array for containing this information is created.

The first setup question asks for the iteration step size. Any integer will be accepted. Suggested step sizes are from 100m to several kilometers depending on the distance to be covered and the desired degree on accuracy. In general smaller step sizes give better results. Step sizes much below 100m may have trouble converging on shell intersections and experience other processing limitations. After this the user is prompted for the complete data file name and path. If the file is not found the user will be reprompted until an acceptable file is found.

The next prompt asks for the elevation starting, stopping, and increment angles in arcmin. For example -4 20 2 would trace the ray starting at -4 arcmin below the observer's eye level and step by 2 arcmin upwards until 20 arcmin is reached or exceeded. The final question asks for the lateral starting, stopping, and increment angle in arcmin. An example of lateral angles might be 120 -120 -60 which would start at 120 and decrement lateral angles by -60 until -120 is reached or exceeded. The actual range of angles will depend on the distances involved. Only the file name is checked, all other inputs accept the given entry. This can result in the program terminating early or getting stuck in a loop if incorrect or contradictory data is entered.

Some of the output is echoed to the screen to indicate processing progress. All the relevant output data is placed in the standard output file 3d_OUT. This file can be renamed to save the contents for future use.

The combination of number of rays traced and number of points in each ray should not exceed 3500 if either plotting or printing was selected. To estimate the number of points divided the distance between observer and object plane by the iteration size and multiply by the number of elevation angles you are processing. This approximation does not include the additional points generated by intersections with the atmospheric shell.

If you are using the ray plotting option see Chapter 4 for a discussion of the commands.

F.2 mir

Input question sequence:

```
Select display mode <0> for Sill' board <1> for itex board

Enter the file name to be loaded...

Next you must select the reference elevation for this picture. To do
this input row and column numbers with a space between in the region
of the reference elevation. Then refine these numbers until you
locate the desired point.
Enter <-1 -1> once you are done.

Next find the Horizon Elevation using the same procedure.

Enter the height of the reference elevation in meters >

Press <Enter> to continue....

Enter transfer characteristic file name.

Would you like to process another transfer characteristic? <Y/N>
```

The first input selects the display procedure based on entered number. Next enter the complete file name of the source image. To select the reference elevation position the cross near the desired position. For examples entering "256 256" will put the cursor in the center of the image. The first number is a row position and the second a column position. The column number is not used by the program but is provided to help accurately determine the correct row by allowing the cursor to move to any position on the image. The select process is ended by entering "-1 -1". The current row then becomes the reference elevation. A similar procedure is used to select the horizon reference. To complete the data necessary to determine the source image scale enter the height in meters (include decimals). The program will respond by displaying the calculated scale. Finally enter the desired transfer characteristic file name. The program will give feed back on progress by displaying the transfer characteristic data and show row number at each layer transition. After the resulting image is displayed you will be asked if you would like to process another transfer characteristic. Answering yes will return you to the transfer characteristic file name prompt.

Bibliography

[1] Bertram, Sidney "Compensating for propagation errors in electromagnetic measuring systems.", *IEEE Spectrum* 8, (March 1971), p. 58-63.

[2] Bock, John "Light Refraction In The Elliptic Model", B.Sc. Thesis(1984), University of Manitoba.

[3] Born, Max and Wolf, E *Principles of Optics*, Pergamon Press, Oxford, 1964.

[4] de Veer, G. *The Three Voyages of William Barents to the Artic Region (1594, 1595, and 1596)*, 2nd ed., Hakluyt Society, London, 1876.

[5] Fleagle, Robert G. and Businger, J. A. *An Introduction To Atmospheric Physics*, Academic Press, New York, 1963.

[6] Fraser, Alistair B. "Solutions of the refraction and extinction integrals for use in inversion and image formation.", *Applied Optics*, vol. 16 no. 1 (1977), p. 160-165.

[7] Fraser, Alistair B. and Mach, W. H. "Mirages.", *Atmospheric Phenomena: Readings from Scientific America*, W.H. Freeman and Co., San Francisco, 1980, p. 81-89.

[8] Friesen, W. J. "Image Processing on the PC (Mirage Simulation).", B.Sc. Thesis(1988), University of Manitoba.

[9] Fujiwhara, S. et. al. "Sinkiro or the Japanese Fata Morgana.", *Geophysical Magazine (Japan)*, 1931, p. 317-373

[10] Greenler, Robert *Rainbows, Halos, and Glories.*, Cambridge University Press, London, 1980.

[11] Harris, Dennis. *Computer Graphics and Applications*, Chapman and Hall, London, 1984.

[12] Hidaka, K. "On a Theory of Sinkiro or the Japanese Fata Morgana.", *Geophysical Magazine (Japan)*, 1931, p. 375-389.

[13] Isaak, P. C. "The Optics of Three Dimensional Refractive Media.", B.Sc. Thesis(1986), University of Manitoba.

[14] Kaplan, Wilfred *Advanced Mathematics for Engineers.*, Addison-Wesley Publishing Co., Reading, Massachusetts, 1981.

[15] Kepler, Johannes "Astronomiae pars Optica Traditur", Frankfurt, 1604 (reprinted by Culture et Civilization, Brussels, 1968), p. 138-143.

[16] Lehn, W. H. and El-Arini, M. B. "Computer-graphics analysis of atmospheric refraction.", *Applied Optics*, vol. 17 no. 19 (1978), p. 3146-3151.

[17] Lehn, W. H. and German, B. A. "Novaya Zemlya effect: analysis of an observation.", *Applied Optics*, vol. 20 no. 12 (1981), p. 2043-2047.

[18] Lehn, W. H. and Sawatzky, H. L. "Image Transmission under Arctic Mirage Conditions.", *Polarforschung 45*, 1975, p. 120-128.

[19] Lehn, W. H. and Wallace, R. E. "Continouse-Tone Mirage Images computed from Digitised Source Photographs.", *Optical Society of America*(Topical Meeting on Meteorological Optics, Technical Digest), (1986), p. 36-38.

[20] Lehn, W. H. "A simple parabolic model for the optics of the atmospheric surface layer.", *Applied Mathematical Modeling*, vol. 9 (1985), p. 447-453.

[21] Lehn, W. H. "Inversion of superior mirage data to compute temperature profiles.", *Journal of the Optical Society of America.*, vol. 73 no. 12 (1983), p. 1622-1625.

[22] Lehn, W. H. "The Novaya Zemlya effect: An arctic mirage.", *Journal of the Optical Society of America*, vol. 69 no. 5 (1979), p. 776-781.

[23] Liljequist, G. H. "Refraction Phenomena in the Polar Atmosphere.", *Scientific Results, Norwegian-British-Swedish Antarctic Expedition, 1949-52*, vol. 2, Part 2, Oslo - Oslo University, 1964.

[24] Mach, William H. and Fraser, Alistair B. "Inversion of optical data to obtain a micrometeorological temperature profile.", *Applied Optics*, vol. 18 no. 11 (1979), p. 1715-1723.

[25] Meyers, Roy E. *Microcomputer Graphics for the IBM PC*, Addison-Wesley Pub. Co., Reading, Mass., 1984.

[26] Midford, Lorne "Analysis of Mirage Images using an Elliptical Model of the Atmosphere.", B.Sc. Thesis(1985), University of Manitoba.

[27] Minnaert, M. *The nature of LIGHT & COLOR in the open air.*, Dover Publications, New York, 1954.

[28] Moon, Parry and Spencer, Domina *Field Theory Handbook including coordinate systems.*, Springer-Verlag, Berlin, 1961.

[29] Morrish, John S. "Inferior Mirages and their Corresponding Temperature Structures." , M.Sc. Thesis(1985), University of Manitoba.

[30] Nansen, "In Night and Ice", The Norwegian Polar Expedition 1893-1896, I , F.A. Brockhaus, Leipzig, 1897, p. 315-316.

[31] O'Connell, D. J. K. "The Green Flash", *Atmospheric Phenomena: Readings from Scientific America*, W. H. Freeman and Co., San Francisco, 1980, p. 91-97.

[32] Shackleton, E. H. *The Heart of the Antarctic: Being the story of the British Antarctic Expedition 1907-1909*, J.B.Lippincott Company, Philadelphia, 1909. p. 367.

[33] Shackleton, E. H. *South - The Story of Shackleton's Last Expedition 1914-1917*, MacMillian, New York, 1920.

[34] Schiele, Wolf-Egbert *Zur Theorie die Luftspiegelungen insbesondere des elliptischen Falles*, Veröffentlichungen des Geophysikalischen Instituts der Univesität Leipzig, Leipzig, 1935

[35] Stravroudis, O. N. *The Optics of Rays, Wavefronts, and Caustics.* Academic Press, New York, 1972.

[36] Tipler, Paul *Physics.* Worth Publishers, Inc., New York, New York, 1976.

[37] Trim, Donald W. *Calculus and Analytic Geometry.* Addison-Wesley Publishing Company, Don Mills, Ontario, 1983.

[38] Wegener, Alfred "Elementare Theorie der atmosphärischen Spiegelungen.", *Annalen der Physik* (Leipzig) 57, 1918, p. 203-230.

[39] Wolf, A. *A History of Science, Technology and Philosophy.*, George Allen & Unwin Ltd, London, 1935.