Ray Tracing in Atmospheric Gravity Waves

by

Hung D. Pham

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

Winnipeg Manitoba 1989

ISBN 0-315-54854-1

Canada

RAY TRACING IN ATMOSPHERIC GRAVITY WAVES

BY

HUNG D. PHAM

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.

Hung D. Pham

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

This thesis deals with forward and inverse problems of ray tracing in the refracting atmosphere. In the forward problem, a mathematical model of atmospheric gravity wave shells under a measurable condition of the atmospheric temperature profile is developed. Light rays of different initial angles are traced from the observer's eyes through the developed model of the atmosphere and the effects of ray bending are also discussed. These ray paths can be used to predict the image formation which is perceived as a mirage if distortion is present. The inverse problem involves determination of the atmospheric temperature profile under an observed condition of multiple image mirages.

Acknowledgement

I wish to thank Professor Waldemar H. Lehn for his guidance and insight throughout the course of work presented in this thesis.

Table of Contents

List of figures

List of Tables

Chapter 1
Introduction and Background

1.1 Introduction

Transmission of images through a refracting atmosphere has been studied for more than 400 years. The most important step in this process is to develop a mathematical model for the earth's atmosphere. In the past, many mathematical models of the atmosphere have been developed by scientists, however these models are generally complicated to solve and some of them only work for some special cases. In 1985, a simple but sufficient model of the earth's atmosphere[1] namely "Parabolic model for the optics of the atmospheric surface layer" was developed by Professor W. H. Lehn. In this model, the earth's atmosphere consists of a small number of spherical shells which are concentric with the earth's surface as shown in figure 1.1.

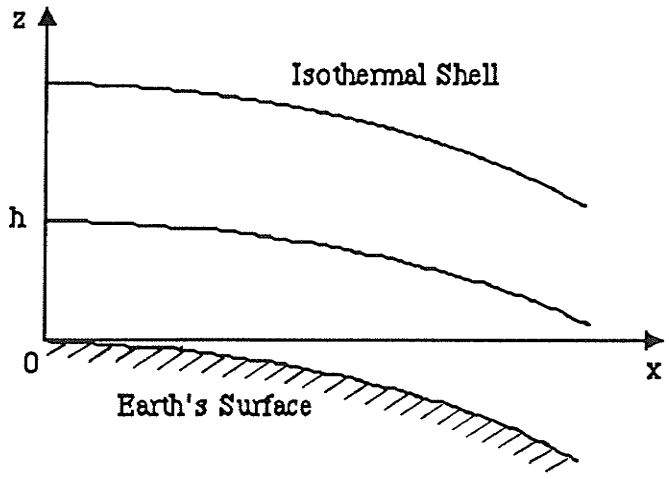


Figure 1.1 Concentric Spherical Shell Model

Each shell represents an isothermal surface and the temperature gradient with respect to elevation is assumed to be constant within a layer (between two shells). In other words, the atmospheric temperature curve is approximated by piecewise linear temperature profile and every point in this temperature profile represents an isothermal concentric shell. In two dimensional cartesian coordinates, the earth's surface can be represented by

$$x^2 + (z + R_E)^2 = R_E^2$$

where R_E is the earth radius. Expansion of the above expression gives

$$z + \frac{z^2}{2 R_E} = - \frac{x^2}{2 R_E} .$$

Since we are only interested in a small range of z (less than 200 meters), the term $z^2/2R_E$ is quite small. Thus the earth's surface equation can be approximated by parabolas as follows:

$$z = - \frac{x^2}{2 R_E} . \qquad (1.1)$$

Then concentric parabolic shells can be represented by

$$z = - \frac{x^2}{2 (R_E + h)} + h$$

where h is the elevation of the shell above the earth's surface at x=0. Since $h << R_E$, h has a very small contribution to the term (R_E +h); therefore we can further approximate the equation of a concentric parabolic shell as follows:

$$z = - \frac{x^2}{2 R_E} + h . \qquad (1.2)$$

Thus equations (1.1) and (1.2) can approximately represent a complete concentric spherical shell model for the atmosphere.

This thesis is considerd as an extension of Lehn's work to further improve the accuracy of atmospheric model by introducing atmospheric waves into his model of the atmosphere. The study of atmospheric waves is one of the oldest studies in fluid dynamics. In the early work on atmospheric waves, photographs of clouds are the most convincing evidence of "waves" in the atmosphere[2]. With improvement in science

technology, more advanced meteorological instruments have become available. In 1929, a study of pressure fluctuations in Britain[2] proved that waves travel on density interfaces within the atmosphere. With further improvement in pressure sensors, more information about atmospheric waves such as their speed, direction and wave length is now available. However, there still remains a large number of wave models due to different hypotheses of wave structures and the complexity of environment effects. In this paper, a simple wave model namely "gravity wave" is chosen to serve the computational purpose of ray tracing.

This new atmospheric model namely "Concentric Parabolic Gravity Wave Shell" consists of a small number of concentric gravity wave shells. Each parabolic shell in Lehn's model is replaced by a gravity wave shell. The equations representing these waves are found by solving fundamental wave equations which are derived from the laws of thermodynamics, gas equations and motion equations of air molecules. The amplitude of these gravity waves is a function of height which is largely dependent on the atmospheric temperature profile. More details of this model will be discussed later in the modeling section.

1.2 Mirages and their causes

Consider the transmission of images through a refracting atmosphere. If distortions due to refracting effects occur during the transmission process then the images perceived by observers are identified as mirages. The main cause of this distortion is due to the fact that light rays are bent as a result of strong temperature gradient (the rate of change of temperature with respect to height). In order to trace the path of a light ray, a series of consecutive ray arcs are projected. The curvature of these ray arcs is directly proportional to the temperature gradient, thus the stronger the temperature gradient, the greater the refractive ray bending effect.

There are basically two types of mirages[3] namely "inferior" and "superior" mirage. These two types of mirages are created under two different atmospheric temperature conditions. If the atmospheric temperature decreases with height, the refraction due to this type of temperature causes inferior mirage. Under this condition, objects perceived by an observer will appear lower than their actual elevation. This type of mirage is usually seen in the desert when pools of water appear on the horizon; these are actually portions of the sky, perceived lowered. In the case of the superior mirage, the atmosphere temperature increases with height. This type of mirage is often found in the polar region, and northern

and southern mid-latitudes. Horizontal surfaces then appear to be concave upward thus the object perceived by the observer will appear higher.

In the two cases, if there exits an inflection point in their temperature profile, multiple images of the object can be perceived by the observer at a certain distance away between the observer and the object. Figure 1.2 shows a case of superior mirage with multiple images at different distances. The multiple image case is more interesting to investigate, therefore in this thesis, we only consider the atmospheric model which has a temperature profile with an inflection point.

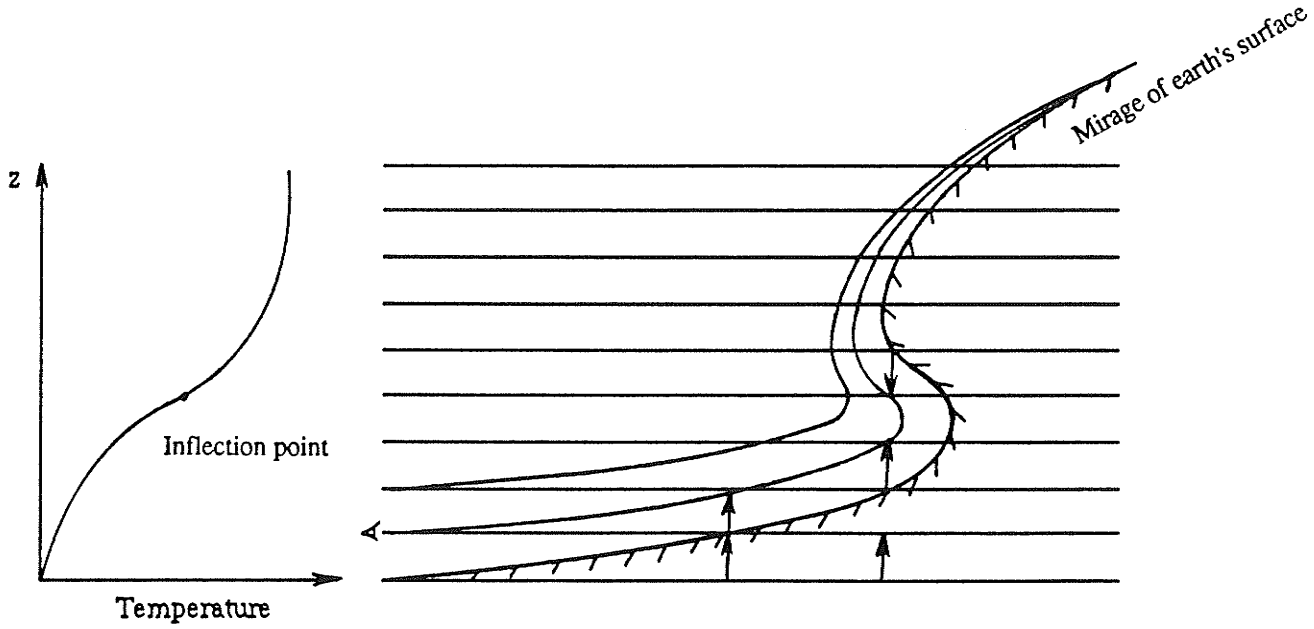


Figure 1.2 Superior Mirage

1.3 Scope

In this paper, the forward and inverse problems of ray tracing are investigated and implemented on digital computers. In the forward problem of ray tracing, first a section of modelling the earth's atmosphere is presented, followed by the sections that concern ray tracing through the developed atmospheric model, implementations and results. Finally, the inverse problem of ray tracing which utilizes an optimization method namely "Rosenbrock Method of Rotating Coordinates" is included.

Chapter 2
Modelling The Earth's Atmosphere

2.1 Gravity Waves

The term "internal gravity wave" is generally used to describe the type of wave in which the restoring force acting on the wave's elements is gravity. This term is sometimes confused with "gravitational wave" which is predicted by the theory of relativity, therefore the term "buoyancy waves" is sometime used for this type of wave to avoid confusion. Before going into mathematical details, it is convenient to define the symbols of commonly used parameters. Table 2.1 defines various symbols and constants used in this thesis.

Symbol	Definition	Value	Dimensions
g	acceleration of gravity	9.81	$m\ s^{-2}$
β	1/ specific gas constant	0.00348	$^0K\ s^2\ m^{-2}$
ε	constant	226×10^{-6}	$m^3\ kg^{-1}$
p	atmospheric pressure	—	$N\ m^{-2}$
k	ray curvature = 1/ radius	—	m^{-1}
R_E	earth's radius	6378	km
ρ	air density	—	$kg\ m^{-3}$
T	air temperature	—	^0K

Table 2.1 Symbol Definitions

2.1.1 Elementary physics of internal gravity wave

In this section, we will investigate the restoring force[4] of a gravity wave which is the buoyancy force exerted on a displaced air element in a stably stratified atmosphere.

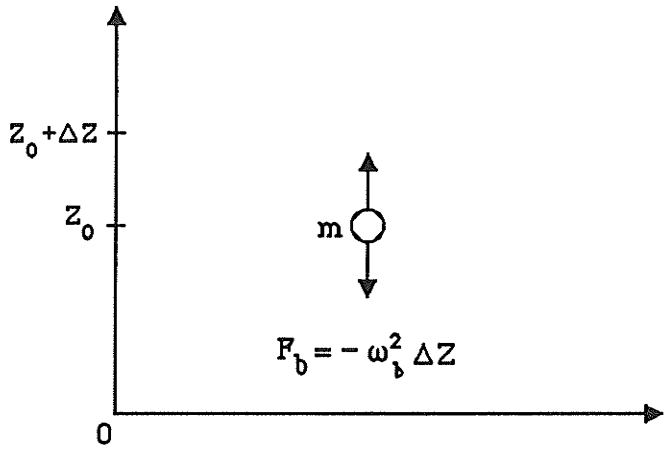


Figure 2.1 Buoyancy Force on an Air Element

Consider an air element of mass m located at an elevation of z_0 . Let the density of air be denoted by ρ. It has been known that air density is decreasing with height at a rate $-d\rho/dz$. By the law of physics, the mass of air element can be expressed as follows

$$m = \rho\,(z_0)\,v$$

where v is the volume of air element, considered for the moment to be fixed. If m is displaced by a small distance Δz, it will be subjected to a buoyancy force of

$$-\,g\,(m_{z_0} - m_{z_0+\Delta z}) = -\,g\left[\rho(z_0) - \left(\rho(z_0) + \frac{d\rho(z_0)}{dz}\,\Delta z\right)\right] v$$

acting to return m to z_0 . The differential equation describing the motion of m is

$$\rho(z_0)\,v\,\frac{d^2\Delta z}{dt^2} = g\,\frac{d\rho(z_0)}{dz}\,\Delta z\,v$$

or

$$\frac{d^2\Delta z}{dt^2} = \frac{g}{\rho(z_0)}\,\frac{d\rho}{dz}\,\Delta z$$

The density gradient can be replaced by the temperature gradient, using the state equation

$$\rho = \frac{\beta p}{T} .$$

Then

$$\frac{d\rho}{dz} = \beta \left(\frac{1}{T} \frac{dp}{dz} - \frac{p}{T^2} \frac{dT}{dz} \right)$$

but the equation for pressure in an atmosphere situated in gravitational field is

$$\frac{dp}{dz} = - g \rho(z)$$

therefore

$$\frac{d\rho}{dz} = - \frac{\rho}{T} \left[\beta g + \frac{dT}{dz} \right] .$$

The air element is not incompressible; its behavior is assumed adiabatic. This adds a correction term[4] to the buoyancy force, giving an equation of motion

$$\frac{d^2 \Delta z}{dt^2} = - \frac{g}{T} \left(\frac{dT}{dz} + \beta g \left[1 - \frac{1}{\gamma_s} \right] \right) \Delta z ,$$

where

γ_s is the ratio of specific heats c_p/c_v.

This equation describes a harmonic oscillation in z direction with a frequency which is known as Vaisala-Brunt (VB) frequency denoted by ω_b:

$$\omega_b^2 = \frac{g}{T} \left(\frac{dT}{dz} + \beta g \left[1 - \frac{1}{\gamma_s} \right] \right) ,$$

or (2.1.1)

$$\omega_b^2 = \frac{g}{T} \left(\frac{dT}{dz} + \beta g \right) - \frac{g^2}{c_s^2} .$$

where

c_s is the speed of sound.

2.1.2 Gravity wave equation

Considering two dimensional propagation in the x and z directions, a set of partial differential equations describe the wave motion is derived from fundamental wave equations[2] by Gossard and Hooke as follows:

$$\frac{\partial u}{\partial t} = -\frac{1}{\rho_0}\frac{\partial p}{\partial x} \tag{2.1.2}$$

$$\frac{\partial w}{\partial t} = -\frac{1}{\rho_0}\frac{\partial p}{\partial z} - g\frac{\rho}{\rho_0} \tag{2.1.3}$$

$$\frac{\partial \rho}{\partial t} + w\frac{\partial \rho_0}{\partial z} = 0 \quad , \quad \frac{\partial u}{\partial x} + \frac{\partial w}{\partial z} = 0 \tag{2.1.4}$$

where

w is the z component of wave velocity

u is the x component of wave velocity

$\rho(z) = \rho_0(z) + \xi\,\rho_1(z) + \ldots$

ξ is a small ordering parameter proportional to the deviation from zero order state.

Taking the partial derivative of (2.1.3) w.r.t t gives

$$\frac{\partial^2 w}{\partial t^2} = -\frac{1}{\rho_0}\frac{\partial}{\partial t}\left(\frac{\partial p}{\partial z}\right) - \frac{g}{\rho_0}\frac{\partial \rho}{\partial t}. \tag{2.1.5}$$

Taking the partial derivative of (2.1.2) w.r.t x gives

$$\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial t}\right) = -\frac{1}{\rho_0}\frac{\partial^2 p}{\partial x^2}. \tag{2.1.6}$$

Taking the partial derivative of (2.1.4) w.r.t t gives

$$\frac{\partial}{\partial t}\left(\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial t}\left(\frac{\partial w}{\partial z}\right) = 0. \tag{2.1.7}$$

Substitute the partial derivative of ρ w.r.t t from (2.1.4) into (2.1.5) and eliminate u between (2.1.6) and (2.1.7). Then we have the following set of partial differential equations:

$$\frac{\partial^2 w}{\partial t^2} + \frac{1}{\rho_0}\frac{\partial}{\partial t}\left(\frac{\partial p}{\partial z}\right) - \frac{g}{\rho_0}\frac{\partial \rho_0}{\partial z} w = 0 \qquad (2.1.8)$$

$$\frac{1}{\rho_0}\frac{\partial^2 p}{\partial x^2} = \frac{\partial}{\partial t}\left(\frac{\partial w}{\partial z}\right). \qquad (2.1.9)$$

These equations are linear and homogeneous in the variables w and p, therefore the following types of solutions are suggested:

$$w = w_z\, e^{i(kx - \omega t)}$$

$$p = p_z\, e^{i(kx - \omega t)}$$

where

w_z and p_z are functions of z only

$k = 2\pi/\lambda$: wave number in the x direction

λ is the wave length

$\omega = 2\pi/T$: angular frequency

T is the time period.

Substitute these solutions into (2.1.9):

$$p_z = i\,\omega\frac{\rho_0}{k^2}\frac{\partial w_z}{\partial z}. \qquad (2.1.10)$$

Substitute these solutions into (2.1.8):

$$-\omega^2 e^{i(kx - \omega t)} w_z + \frac{1}{\rho_0}\frac{\partial}{\partial t}\left[e^{i(kx - \omega t)}\frac{\partial}{\partial z}(p_z)\right] - \frac{g}{\rho_0}\frac{\partial \rho_0}{\partial z} w_z\, e^{i(kx - \omega t)} = 0 .$$

(2.1.11)

Substitute (2.1.10) into (2.1.11) and take partial derivative w.r.t t

$$\frac{\partial^2 w_z}{\partial z^2} + \frac{1}{\rho_0}\frac{\partial \rho_0}{\partial z}\frac{\partial w_z}{\partial z} + \frac{k^2}{\omega^2}\left(-\frac{g}{\rho_0}\frac{\partial \rho_0}{\partial z} - \omega^2\right) w_z = 0 . \qquad (2.1.12)$$

Recall

$$\omega_b^2 = -\frac{g}{\rho}\frac{\partial \rho}{\partial z} \approx -\frac{g}{\rho_0}\frac{\partial \rho_0}{\partial z} .$$

Thus (2.1.12) becomes

$$\frac{\partial^2 w_z}{\partial z^2} + \frac{1}{\rho_0}\frac{\partial \rho_0}{\partial z}\frac{\partial w_z}{\partial z} + \frac{k^2}{\omega^2}\left(\omega_b^2 - \omega^2\right) w_z = 0 . \qquad (2.1.13)$$

This partial differential equation can be further simplified if

$$\frac{1}{\rho_0}\frac{\partial \rho_0}{\partial z} = -\alpha \quad \text{(constant)}$$

(ie. ρ_0 varies exponentially with height $\rho_0 = \rho_s e^{-\alpha z}$). Then we can transform variable w_z as follows.

Let

$$W_z = e^{-0.5\,\alpha z} w_z \quad ==> \quad w_z = \sqrt{\frac{\rho_s}{\rho_0}}\, W_z .$$

Then

$$\frac{\partial w_z}{\partial z} = \sqrt{\frac{\rho_s}{\rho_0}}\frac{\partial W_z}{\partial z} - \frac{1}{2}\frac{\sqrt{\rho_s}}{\sqrt{\rho_0^3}}\frac{\partial \rho_0}{\partial z} W_z$$

and

$$\frac{\partial^2 w_z}{\partial z^2} = \sqrt{\frac{\rho_s}{\rho_0}}\frac{\partial^2 W_z}{\partial z^2} - \frac{\sqrt{\rho_s}}{\sqrt{\rho_0^3}}\frac{\partial W_z}{\partial z}\frac{\partial \rho_0}{\partial z} + \frac{3}{4}\frac{\sqrt{\rho_s}}{\sqrt{\rho_0^5}} W_z \left(\frac{\partial \rho_0}{\partial z}\right) - \frac{1}{2}\frac{\sqrt{\rho_s}}{\sqrt{\rho_0^5}}\left(\frac{\partial \rho_0}{\partial z}\right)^2 W_z .$$

Substitute into (2.1.13) and simplify to get

$$\frac{\partial^2 W_z}{\partial z^2} + \left[\frac{k^2}{\omega^2}\left(\omega_b^2 - \omega^2\right) - \left(\frac{1}{2\rho_0}\frac{\partial \rho_0}{\partial z}\right)^2\right] W_z = 0 . \qquad (2.1.14)$$

The quantity inside the square bracket is known as "vertical wave number" denoted by

$$n^2 = \frac{k^2}{\omega^2}\left(\omega_b^2 - \omega^2\right) - \left(\frac{1}{2\rho_0}\frac{\partial \rho_0}{\partial z}\right)^2 . \qquad (2.1.15)$$

Then (2.1.14) becomes

$$\frac{\partial^2 W_z}{\partial z^2} + n^2 W_z = 0 \qquad (2.1.16)$$

Depending on the value of n^2, which we will assume to be a constant, W_z will have different forms of solution.
If $n^2 < 0$ (say $n = i\gamma$) then the solution of W_z is

$$W_z = e^{\gamma z} + e^{-\gamma z}$$

Gravity waves' amplitudes are evanescent, dying off exponentially.
If $n^2 > 0$ (say $n = \beta$) then the solution of W_z is

$$W_z = e^{i\beta z} + e^{-i\beta z}$$

Gravity waves' amplitudes are oscillating with height.
The solution for the variable w is thus a simple plane wave:

$$w = \sqrt{\frac{\rho_s}{\rho_0}}\, W_z\, e^{i(kx - \omega t)} .$$

2.2 Three Section Model

This model[2] divides the earth's atmosphere into three sections. It is chosen such that temperature and density of the atmosphere are continuous within each section and as well as at section boundaries. For simplicity, the wind factor will not be included and the "vertical wave number" n is assumed to be constant within each section. Figure 2.2 shows the three section model and the corresponding temperature profile.

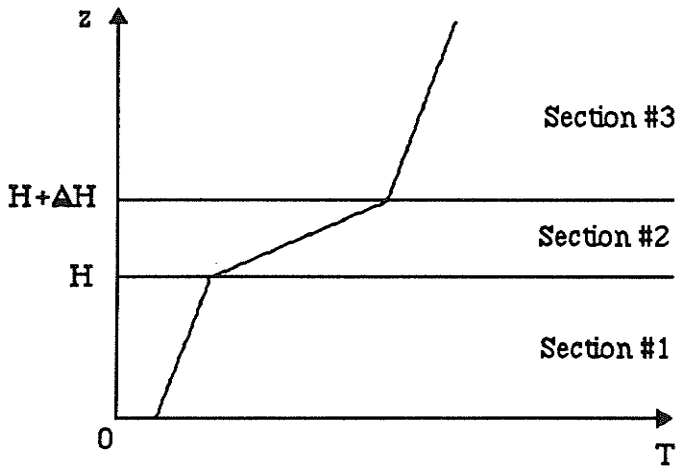


Figure 2.2 Three Section Model and Corresponding Temperature Curve

From the equation of "vertical wave number" (2.1.15), the value of

$$\left(\frac{1}{2\rho_0}\frac{\partial \rho_0}{\partial z}\right)^2$$

is usually small, therefore the sign of n^2 is largely dependent on the ratio of ω_b / ω. From the expression of ω_b (eq. 1.1.1) above and the three section model temperature gradients of figure 1, we can predict the value for the vertical wave number of each section:

$$n_1^2 < 0$$
$$n_2^2 > 0$$
$$n_3^2 < 0 .$$

Applying the gravity wave amplitude solution derived earlier to this model, we have the following solutions for each section. The notation has been changed: n_1 and n_3 will now represent the roots of $-n_1^2$ and $-n_3^2$ respectively.

$$W_3 = C\, e^{-n_3 z} \qquad \text{for } z > H+\Delta H$$

$$W_2 = B_1\, e^{in_2 z} + B_2\, e^{-in_2 z} \qquad \text{for } H < z < H+\Delta H$$

$$W_1 = A_1\, e^{n_1 z} + A_2\, e^{-n_1 z} \qquad \text{for } 0 < z < H$$

These solutions define a wave system which is evanescent in the upper and lower layers satisfying the boundary conditions of

$$W(0) = 0$$
$$W(\infty) = 0 .$$

Since $W(0) = 0$, the amplitude solution for section # 1 reduces to

$$W_1 = A \sinh n_1 z$$

Using the following relation

$$D_1 \cos x + D_2 \sin x = \frac{1}{2}(D_1 - iD_2)\, e^{ix} + \frac{1}{2}(D_1 + iD_2)\, e^{-ix} = E_1\, e^{ix} + E_2\, e^{-ix} ,$$

the solution for section # 2 can be expressed as

$$W_2 = D_1\left[\cos n_2 z + \frac{D_2}{D_1}\sin n_2 z\right].$$

The kinematic boundary condition requires continuity in gravity wave amplitude. For convenience, we refer the solution in all sections to the wave amplitude at z=H, say W_H. Then the solution can be expressed as follows:

$$W_1 = W_H\left(\frac{\text{Sinh } n_1 z}{\text{Sinh } n_1 H}\right)$$

$$W_2 = W_H\left[\cos n_2 (z\text{-}H) + A \sin n_2(z - H)\right]$$

$$W_3 = W_H\left[\cos\left(n_2 \Delta H\right) + A \sin\left(n_2 \Delta H\right)\right] e^{-n_3[z - (H+\Delta H)]}.$$

The dynamic boundary condition requires that the height derivatives of W_i be continuous across the section interfaces:

at z = H

$$n_1 \coth n_1 H = n_2 A ;$$

at $z = H + \Delta H$

$$n_2 A \cos (n_2 \Delta H) - n_2 \sin (n_2 \Delta H) = - n_3\left[A \sin (n_2 \Delta H) + \cos (n_2 \Delta H)\right].$$

The coefficient A can be found from either dynamic boundary condition above:

$$A = \frac{n_1 \coth n_1 H}{n_2}.$$

Thus the complete solution of gravity wave amplitude expressed in term of W_H

$$W_1 = W_H\left(\frac{\sinh n_1 z}{\sinh n_1 H}\right)$$

$$W_2 = W_H\left[\cos (n_2(z\text{-}H)) + \frac{n_1}{n_2}\coth (n_1 H) \sin (n_2(z\text{-}H))\right]$$

$$W_3 = W_H\left[\cos\left(n_2 \Delta H\right) + \frac{n_1}{n_2}\coth (n_1 H) \sin\left(n_2 \Delta H\right)\right] e^{-n_3[z-(H+\Delta H)]}. \qquad (2.2.1)$$

These equations (2.2.1) define gravity wave amplitudes as functions of elevation in their corresponding section.

2.3 Three Section Concentric Gravity Wave Shell model

In this model, the gravity wave is added to the concentric parabolic shell of Lehn's model. Recall the solution for gravity wave

$$W = W_z e^{i(kx - \omega t)} ;$$

if we replace the time dependent variable t by a phase shift α and take only the real part of x variable, then the solution for gravity wave is simplified to

$$W = W_z \cos (kx + \alpha) .$$

Recall that this equation describes the vertical velocity of gravity wave in the atmosphere, thus by performing integration, the solution for gravity wave positions can be obtained. However, we will use this equation as the approximated equation for the gravity wave position. The reasons that make this approximation possible are as follows:

. Since this equation is exponential in the z direction and sinusoidal in the x direction, integration will simply reproduce exponential and sinusoidal terms

. The parameter values for the gravity waves such as α, λ, W_H are not known. We only need the form of the function to build the gravity wave model for the atmosphere

. For the first analysis of the problem, this approximation is sufficiently accurate.

To represent a concentric spherical gravity wave shell, the circular term (which is approximated by parabola) is added to the expression of gravity wave

$$W = W_z \cos (kx+\alpha) - \frac{x^2}{2 R_E} + h \qquad (2.3.1)$$

where W_z is the amplitude of the gravity wave shell which is a function of elevation and defined by three section gravity wave amplitude equations of (2.2.1). Figure 2.3 shows the graph of the model for "three section concentric gravity wave shells" which was derived from the wave equation of (2.3.1) with arbitrarily chosen parameters. In this figure, the solid lines represent the isothermal shells distorted by the gravity waves; the dotted lines define the boundaries of the three sections.

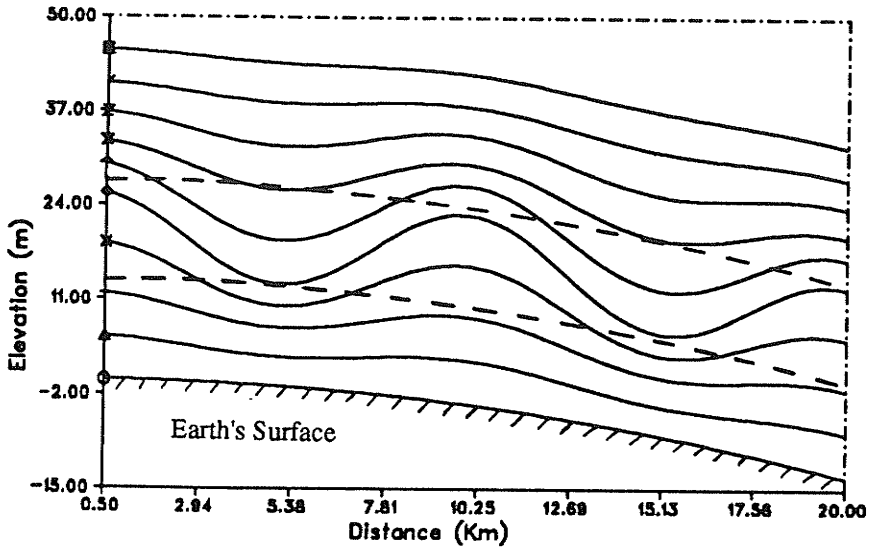


Figure 2.3 Three Section Concentric Gravity Wave Shells model

Chapter 3
Ray Tracing

The law of geometrical optics states that in any constant medium, light rays obey the law of rectilinear propagation. However, in the case of mirage, light rays certainly do not travel in straight line manner but rather in the form of curves. This chapter investigates the causes of ray bending.

3.1 Ray Curvature

Consider a light ray that travels in the "concentric gravity wave shell model" of the atmosphere; the curvature[1] of such ray is given by

$$k = \frac{\varepsilon \rho}{(1 + \varepsilon \rho) T(z)} \sin\theta \left(\frac{dT}{dz} + \beta g\right) \tag{3.1.1}$$

where

$$\rho(z) = \frac{\beta p_0}{T(z)} e^{-g\beta \int_0^z \frac{dT(z')}{T(z')}}$$

θ is the angle between the light ray and the normal to the gravity wave shell.

p_0 is the pressure of air at ground level.

From the expression of light ray curvature, k is directly proportional to temperature gradient. When temperature gradient is strong, k is large thus the ray bending effect becomes more evident, whereas, in the case of weak temperature gradient, the ray travels in a nearly straight line as predicted by the law of geometrical optics. Figure 3.1 shows a light ray travelling between gravity wave shells.

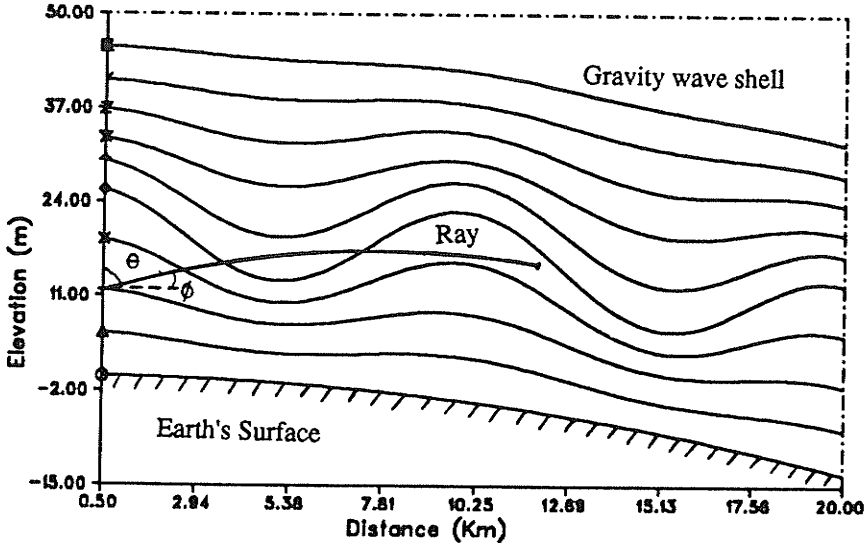


Figure 3.1 A light ray travels between gravity wave shells

3.2 Ray Path

Ray paths are also approximated by parabolas of the form

$$z = a\,x^2 + b\,x + c\,.$$

Using the boundary conditions, we can determine the coefficients a, b and c as follows:

$z = z_0$ at $x = 0$ thus

$$c = z_0$$

$$\tan \phi_0 = \frac{dz}{dx}(x=0) = b\,.$$

the coefficient "a" is approximated as in the case of parabolic shells discuss earlier:

$$a = -\frac{1}{2r}$$

where

$r = 1/k$ =radius of the ray arc.

Thus the equation of the ray path becomes

$$z = -\frac{1}{2r} x^2 + x \tan \phi_0 + z_0 \qquad (3.2.1)$$

where

z_0 is the initial elevation of the ray arc (observer eye's elevation) at x=0.
ϕ_0 is the angle between light ray and the horizontal at x=0.

3.3 Flat Earth's Surface Model

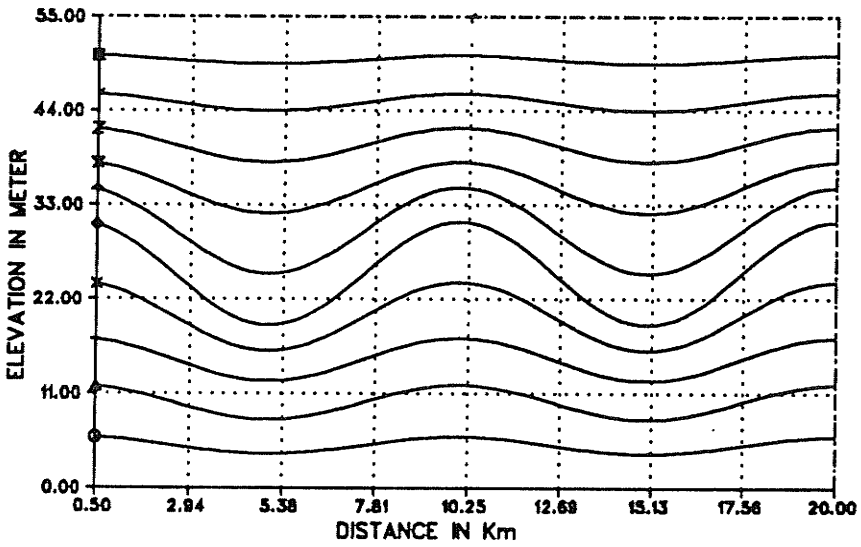


Figure 3.2 Flat earth's surface model

From the equations of gravity wave shell and ray path, we can simplify our atmospheric model by flattening the earth's surface mathematically. This is done by adding to the previously developed equations the amount of

$$-\frac{x^2}{2R_E}.$$

Thus, the equation for gravity wave shell and ray path for "flat earth's surface" model are:
Gravity wave shell

$$z = W_z \cos(kx + \alpha) + h \tag{3.3.1}$$

Ray path

$$z = -\frac{x^2}{2}\left(\frac{1}{r} - \frac{1}{R_E}\right) + x \tan \phi_0 + z_0 \tag{3.3.2}$$

Figure 3.2 shows the "flat earth's surface" model of gravity wave shells.

3.4 Ray Transition Between Gravity Wave Shells

The ray curvature value (eq. 3.1.1) is largely dependent on temperature gradient. From the gravity wave shell model that we have seen earlier, every point in x direction between two gravity wave shells will have a different temperature gradient. In order to complete a ray path, a series of consecutive parabolic ray arcs with small arc length are projected to minimize the error due to variation of temperature gradient in x direction. At the beginning of every ray arc, the new ray curvature is calculated based on its present temperature gradient, density and ray angles ϕ and θ. Having the current values of ray angles and ray curvature, the end point's elevation coordinate of parabolic ray arc can be found using equation (3.3.2). During the transition between the two shells (within a layer), the angle θ between light ray and normal of gravity wave shell is kept constant and has its new value when intersection with gravity wave shell occurs.

3.4.1 Normal intersection

Consider a parabolic ray arc with angle ϕ_1 and θ_1 which starts at x=0.5 and ends at $x=x_p$ as shown in figure 3.3. This ray arc enters the next layer and intersects with the next layer boundary at the point I. The x coordinate of this intersection point denoted x_I can be found by equating the two equations of ray arc and gravity wave shell in the z variable:

$$W_i \cos(kx_I) + h_i = -\frac{x_I^2}{2}\left(\frac{1}{r} - \frac{1}{R_E}\right) + x_I^2 \tan \phi_1 + h_{i-1} .$$

Unfortunately, this equation cannot be solved analytically; therefore, a numerical method namely "Golden Section Search" will be used to find x_I. At the intersection point I, the

new angle ϕ_2, θ_2 and ray curvature are found; then a new ray arc of length $(x_I - x_P)$ is projected to continue the ray path. The reason for arc length reduction is to maintain the ray path x-coordinate interval at the preset value which is used to find the "transfer characteristic" discussed later in this chapter.

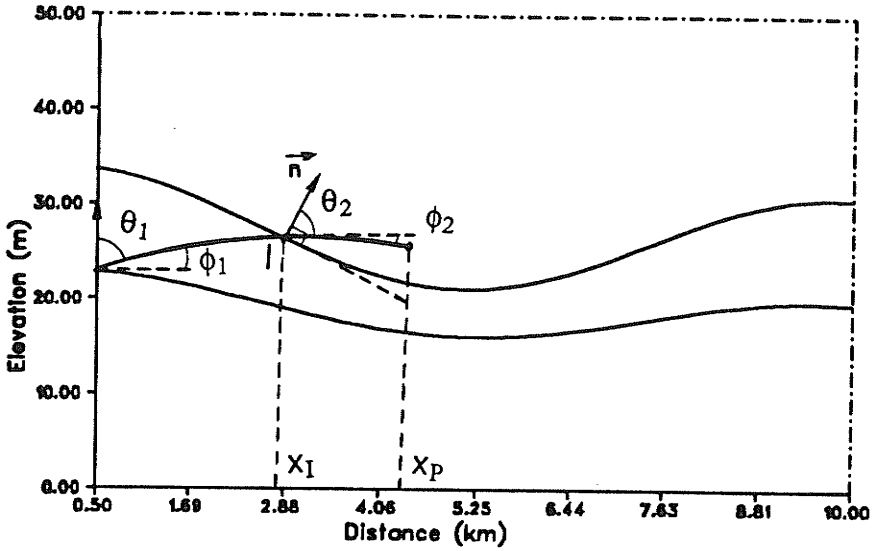


Figure 3.3 Normal Intersection

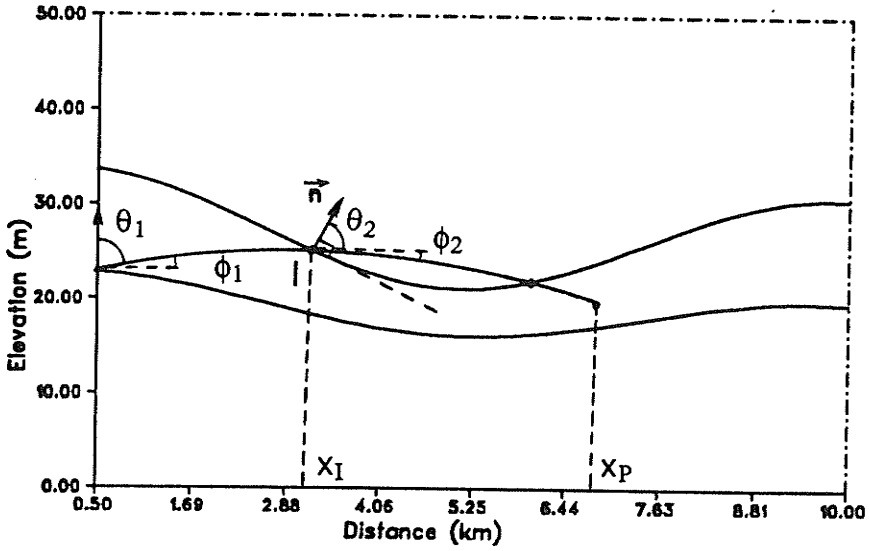


Figure 3.4 Re-entry Intersection

3.4.2 Re-entry intersection

Consider a parabolic ray arc with angle ϕ_1 and θ_1 which starts at x=0.5 and ends at $x=x_p$ as shown in figure 3.4. This ray arc intersects twice with the upper shell. In this case, the first intersection at x_I must be found then the new ray arc of length $(x_I - x_P)$ is projected. This new ray arc may or may not intersect with the shell depending on its new ray angles and ray curvature.

3.4.3 Ray Angles

In previous sections, the ray angle ϕ is said to have different value at every point in the atmosphere and the angle θ is said to remain constant within each layer and be recalculated when intersection with gravity shell occurs. In this section, we will look at the variations in these angles. Figure 3.5 shows variation of ray angles during their transitions between gravity wave shells.

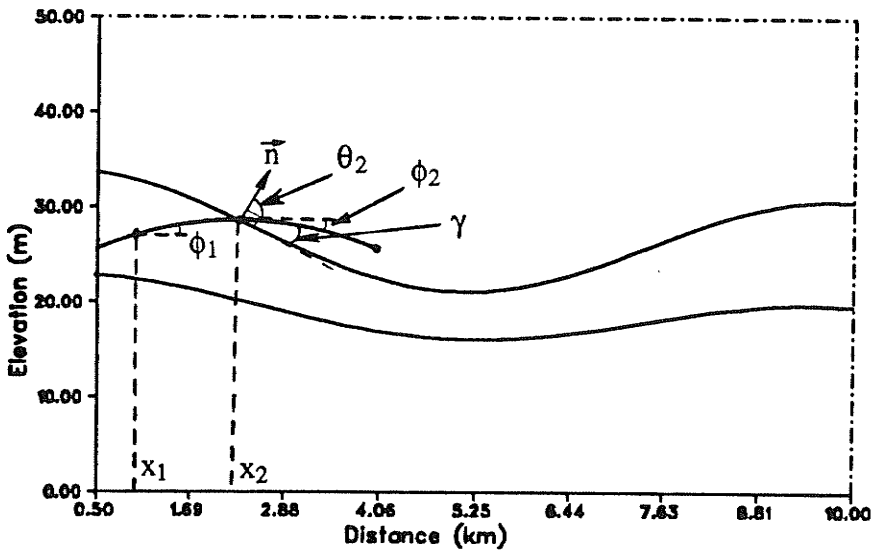


Figure 3.5 Ray Angles

At the beginning of the ray path (ie. x=0) the angle ϕ_0 is given as initial ray angle. At the beginning of each ray arc, the new ray angle ϕ will be calculated based on the equation of previous ray arc. Consider the ray angle ϕ_2 at $x=x_2$, its value can be found as follows:

$$\phi_2 = \tan^{-1} \text{(ray slope at } x=x_2)$$

where

$$\text{ray slope at } (x=x_2) = \frac{dz(x=x_2)}{dx} = -x_2\left(\frac{1}{r} - \frac{1}{R_E}\right) + \tan\phi_1$$

At every ray-gravity wave shell intersection, the new value of ray angle θ is found as follows:

Let γ be the angle between light ray and tangent of gravity wave shell at $x=x_2$. Then we have the relation (since $\gamma + \theta_2 = 90^0$)

$$\sin\theta_2 = \cos\gamma\,.$$

Apply the series expansion to $\cos\gamma$

$$\cos\gamma = 1 - \frac{\gamma^2}{2!} \ldots$$

where γ can can be found by

$$\gamma = \tan^{-1} \text{(slope of ray)} - \tan^{-1} \text{(slope of gravity wave shell)}$$

which can can be approximated by

$$\gamma \approx \tan\gamma \approx \text{slope of ray - slope of gravity wave shell}\,.$$

3.4.4 Golden Section Search:

Golden Section Search[5] is a single dimension search technique which is based on function evaluations to find the small interval that contains the minimum in the least number of evaluations. Consider a unimodal function f(x) in figure 3.6.

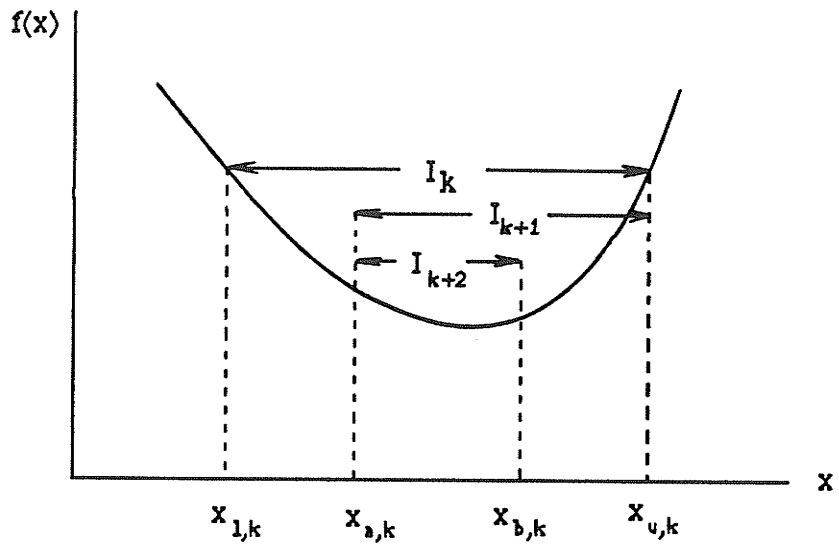


Figure 3.6 Golden Section Search

At iteration number k, the interval which contains the minimum is I_k. The interval for the next iterations is such that

$$I_k = I_{k+1} + I_{k+2} \,. \tag{3.4.1}$$

This can only be achieved if the ratio of successive intervals is held constant.

$$\frac{I_k}{I_{k+1}} = \frac{I_{k+1}}{I_{k+2}} = K \tag{3.4.2}$$

Substitute (3.4.2) into (3.4.1)

$$I_k = \frac{I_k}{K} + \frac{I_k}{K^2};$$

multiply both sides by K^2 / I_k

$$K^2 = K + 1;$$

thus

$$K = \frac{1 \pm \sqrt{5}}{2} \,.$$

Taking positive root, K becomes

$$K = 1.61803 .$$

which is known as "Golden Ratio". Let $x_{l,k}$ and $x_{u,k}$ be the lower and upper limit respectively at the k^{th} iteration. Then the two extra points $x_{a,k}$ and $x_{b,k}$ are found using the "Golden Ratio" as follows:

$$x_{a,k} = x_{u,k} - \frac{1}{K}(x_{u,k} - x_{l,k}) \tag{3.4.3}$$

$$x_{b,k} = x_{l,k} + \frac{1}{K}(x_{u,k} - x_{l,k}); \tag{3.4.4}$$

then the function evaluations at $x_{a,k}$ and $x_{b,k}$ are calculated:

$$E_{a,k} = f(x_a,k)$$

$$E_{b,k} = f(x_b,k) .$$

The interval for next iteration k+1 will depend on the following two cases.
Case 1: $E_{a,k} > E_{b,k}$ (ie. the minimum lies between $x_{b,k}$ and $x_{u,k}$)
The reduced interval for next iteration is

$$x_{l,k+1} = x_{a,k}$$

$$x_{u,k+1} = x_{u,k}$$

$$x_{a,k+1} = x_{b,k}$$

$$x_{b,k+1} = x_{l,k+1} + (x_{u,k+1} - x_{l,k+1})/K .$$

Case 2: $E_{a,k} < E_{b,k}$ (ie. the minimum lies between $x_{l,k}$ and $x_{a,k}$)
The reduced interval for next iteration is

$$x_{l,k+1} = x_{l,k}$$

$$x_{u,k+1} = x_{b,k}$$

$$x_{a,k+1} = x_{u,k+1} + (x_{u,k+1} - x_{l,k+1})/K$$

$$x_{b,k+1} = x_{a,k} .$$

This process is repeated until the interval which contains the minimum is small enough to satisfy the preset value. Figure 3.7 summarizes the "golden search" procedure.

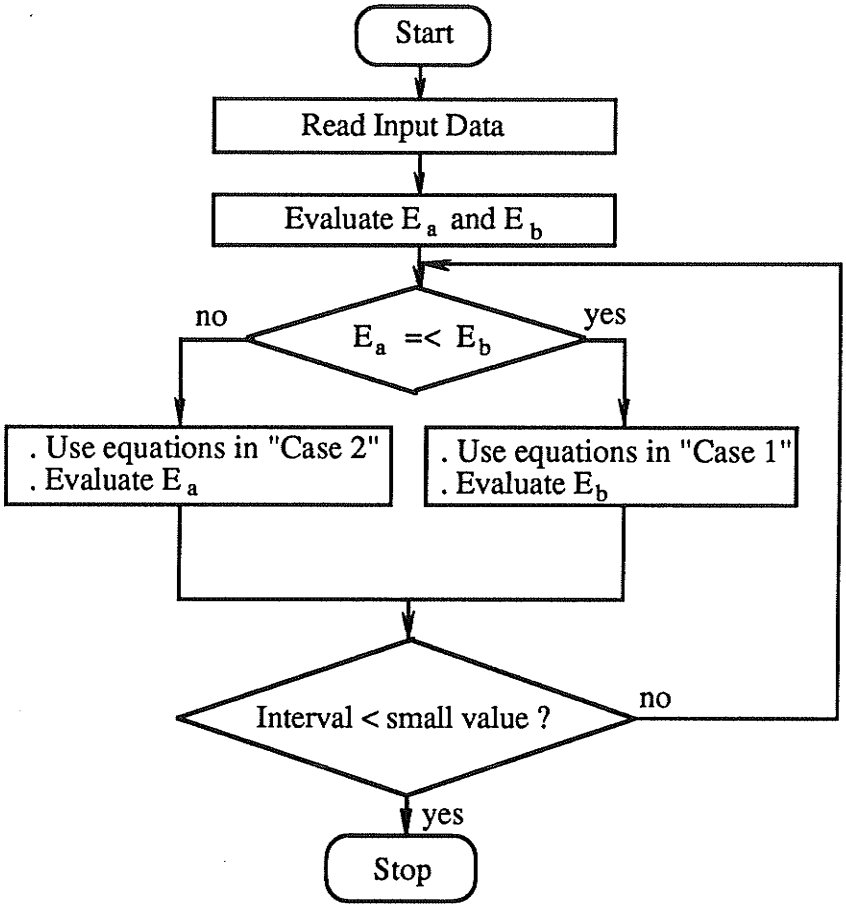


Figure 3.7 "Golden Search" procedure

Chapter 4
Implementation and Results

In this chapter, explanatory comments along with a flowchart are used to explain the operation of the ray tracing program.

4.1 Implementation steps

The ray tracing program is written in *Microsoft C* [6] and utilizes *Halo 88* [7] software package for graphics purposes. There are 5 main sections in this ray tracing process.

4.1.1 Reading input parameters

The following parameters are necessary to the ray tracing program:

. Observer elevation: the elevation of the observer's eyes which is also the starting elevation of ray paths.

. Initial ray angle ϕ_0: starting angles of light rays with respect to horizontal (in minutes). The total number of initial angles corresponds to the number of light rays to be traced.

. Limitation ranges: vertical and horizontal limits of light rays.

. Atmospheric temperature profile: temperature (in °C) of the atmosphere as a function of elevation.

. Gravity wave's amplitude at z=H (see Figure 2.2 for definition of H).

. Ray arc step size: the larger the step size the less accurate will be the ray path calculation, due to misses of intersections with gravity wave shells. However, the larger the step size the smaller will be the number of points calculated on each light ray, thus less computation time is required.

4.1.2 Building the Atmospheric Gravity Wave Shell model

. If the starting elevation point (observer's elevation) is not coincident with a point in temperature profile, an extra point in the temperature profile is created using linear interpolation. The reason for this extra point is that we want to start the ray path at the

intersection of a gravity wave shell so that the initial ray angle ϕ_0 can be calculated. This will become more clear in the later sections.

. After all of the necessary points in the atmospheric temperature profile are available, we need to search through this profile and divide the atmosphere into 3 sections according to its temperature gradient, as follows:

$$\text{temperature gradient} \equiv \frac{T_{\text{upper shell}} - T_{\text{lower shell}}}{Z_{\text{upper shell}} - Z_{\text{lower shell}}}$$

Threshold = maximum temperature gradient / 2

Section #1: temperature gradient < Threshold for $Z > H+\Delta H$

Section #2: temperature gradient > Threshold for $H < Z =< H+\Delta H$

Section #3: temperature gradient < Threshold for $Z =< H$

. The vertical wave number of each section is calculated as follows:

$$n_i^2 = k^2\left[\left(\frac{\omega_{bi}}{\omega}\right)^2 - 1\right]$$

where

$$\omega_{bi} = \frac{g}{T_{avg}}\left(\text{Temperature gradient} + \beta\, g\right) - \frac{g^2}{c_s^2}$$

T_{avg} = Sectional average temperature

. Each point in the temperature profile represents a concentric gravity wave shell as described in the previous chapter whose amplitude is found by its sectional amplitude wave equation (eq. 2.2.1).

. Once all the gravity wave amplitudes are available, we have a complete model of concentric gravity wave shell of the form:

$$W_i = A_i \cos\left(\frac{2\pi}{\lambda} x + \alpha\right)$$

where

A_i : gravity wave amplitude of shell number i

λ = horizontal wave length of gravity wave

α = arbitrary phase shift.

4.1.3 Ray path

To complete a ray path, a series of sequential parabolic arcs (eq.3.3.2) are projected. At the starting point of each ray arc, the ray angles ϕ, θ and ray curvature must be known in order to calculate its path. These ray angles and ray arcs are found as follows.

. Angle θ is calculated (eq.3.4.3) where intersection with gravity wave shell occurs (see Figure 3.5) and its value is kept constant during the ray transition within a layer (between upper and lower shells).

. Angle ϕ is calculated (eq. 3.4.2) at every starting point of parabolic ray arc and intersection point with gravity wave shell.

. Ray curvature (eq.3.1.1) is found based on the current data of temperature gradient, density and ray angles.

. If an intersection with a gravity wave shell is detected, the coordinates of the intersection are found using "Golden section Search" technique; then a ray arc of reduced step size is projected based on the current ray data at the intersection point so that the coordinates of ray paths are equally spaced in the x-direction.

. Termination of a light ray occurs when it exceeds the range limits.

4.1.4 Transfer Characteristic curve

The transfer characteristic (TC) curve characterizes the relation between the apparent height and real height of an object at a given distance away from the observer.

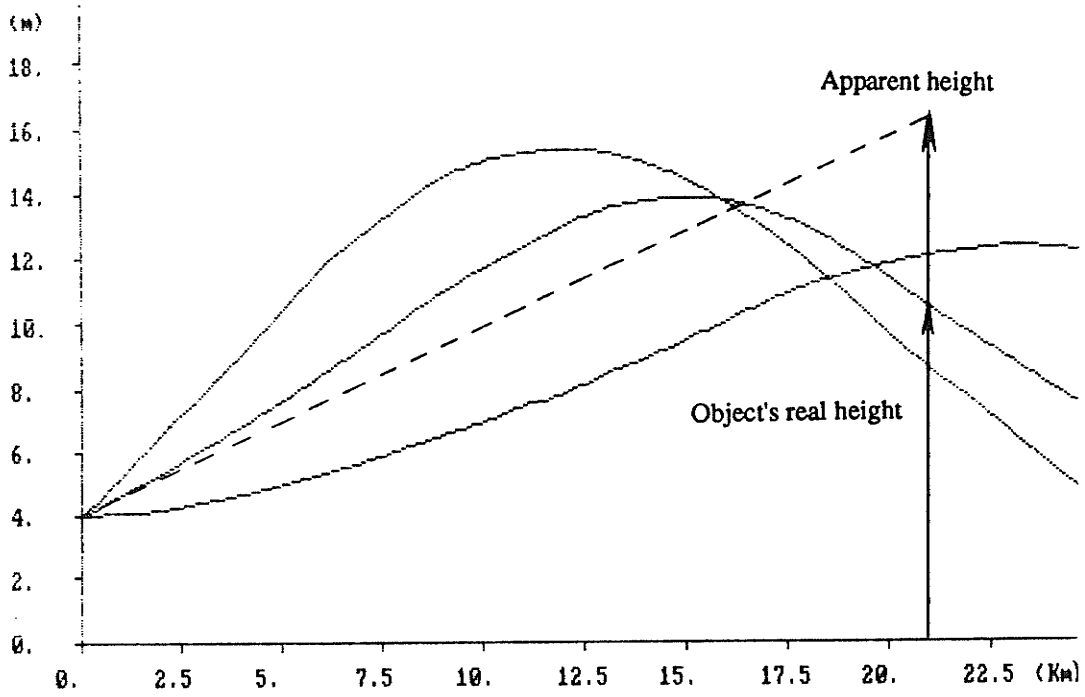


Figure 4.1 Object's Apparent Height Versus Real Height

The apparent height is defined as the height of an object as it appears to the observer's eyes. From the observer's eye, the tangent of the corresponding initial ray angle is extended in a straight line; the apparent height is the elevation of this extension line at the object's coordinates. The object's real height is the elevation of the actual light ray at the object's coordinates. Figure 4.1 shows the relation between the apparent height and object's real height.

4.1.5 Displaying

The gravity wave model, light ray paths and TC curves are displayed using the Halo 88 software package. These Halo 88 graphic subroutines can be called by the Microsoft C program for display purposes without exiting from the current C program. To compile a Microsoft C program and link to Halo 88 the following procedure is required:

- **msc filename;** (compile)
- **link filename+halodvxx,,,halous** (link)

4.2 Flow chart

The following flow chart shows the main operation of the ray tracing program. Details of each routine have been discussed in the previous section.

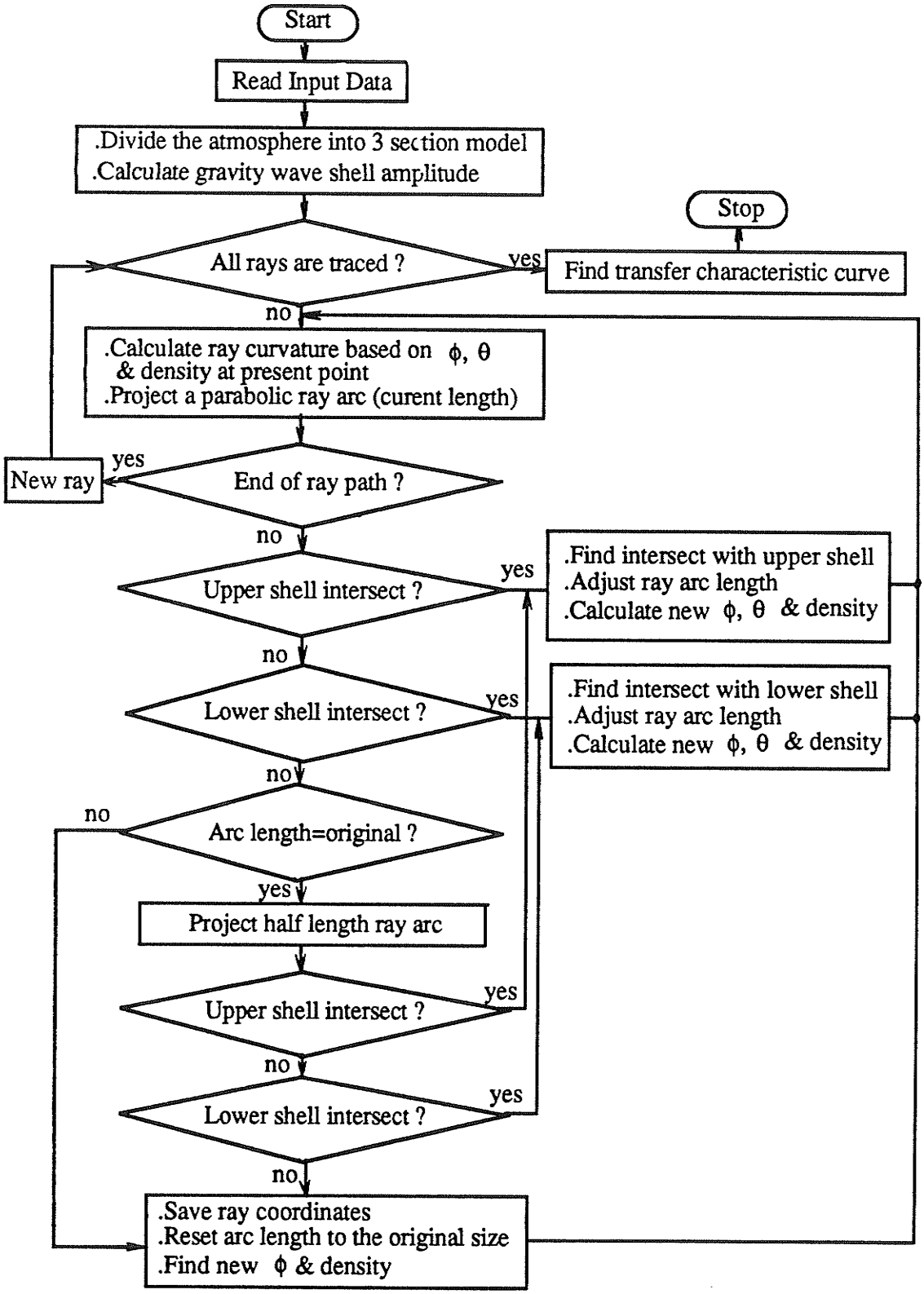


Figure 4.2 Ray Tracing Flowchart

4.3 Results

A temperature profile named "Jm3" is used as the input atmospheric temperature profile for ray tracing program to demonstrate the effect of varying gravity wave parameters. Figure 4.3 shows "Jm3" temperature profile.

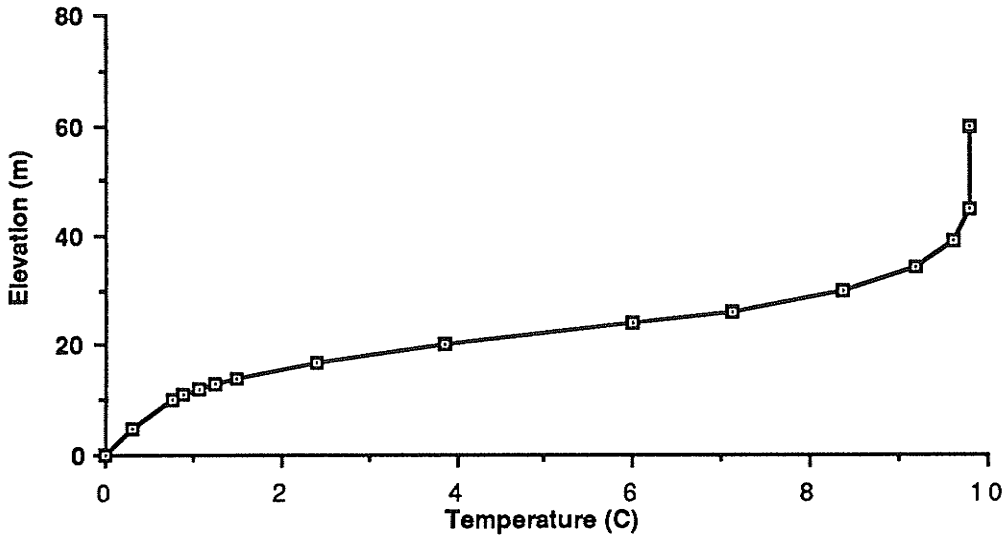


Figure 4.3 "Jm3" Temperature Profile

Having "Jm3" as the input temperature profile, the atmospheric gravity wave shell model is created as shown in Figure 4.4. In this figure, the left portion gives the boundaries between the sections and indicates the amplitude of gravity wave as a function of height for "three section model"; the horizontal scale is in meters. The right portion of this figure shows the atmospheric gravity wave shells with their corresponding amplitude as shown in the left portion and the wave length λ of 10 km. (Note that the horizontal unit in this right portion is in km and its scale can be read as: (indicated scale - 10 km)). As mentioned earlier, the atmospheric model plays an important role in the ray tracing process. The following results will show the sensitivity to the gravity wave's amplitude, wave length and phase shift.

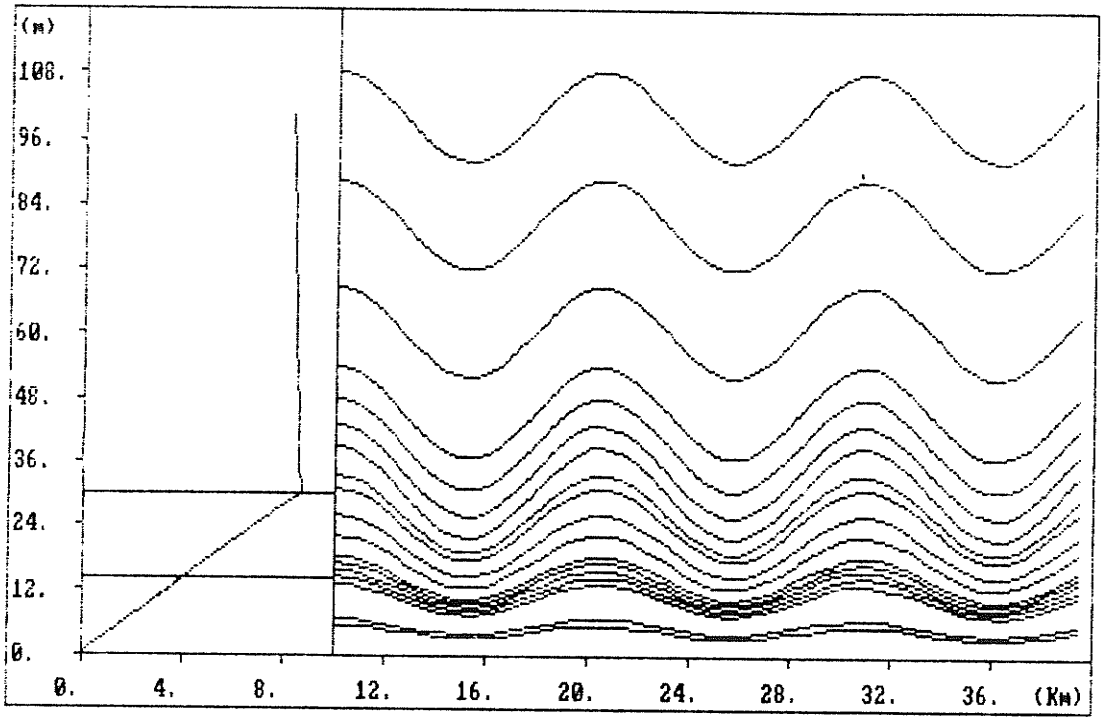


Figure 4.4 Gravity Wave Shell Model (W_H=4 m phase shift α=0)

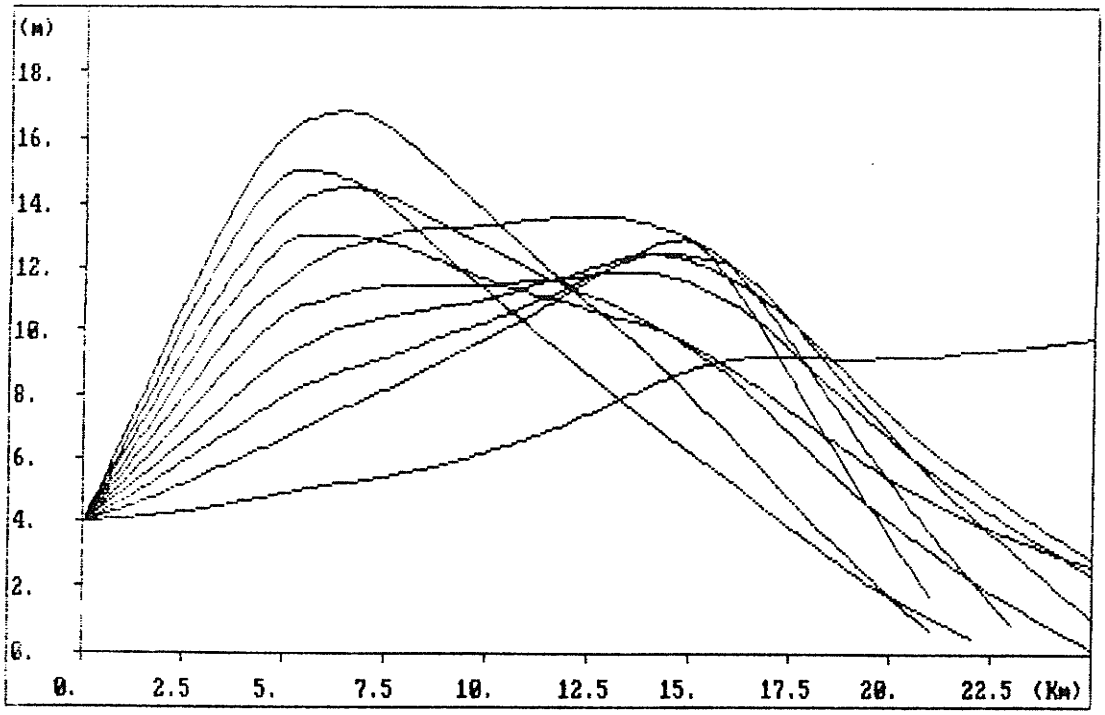


Figure 4.5 Ray Tracing (W_H=4 and Phase Shift $\alpha=0^0$)

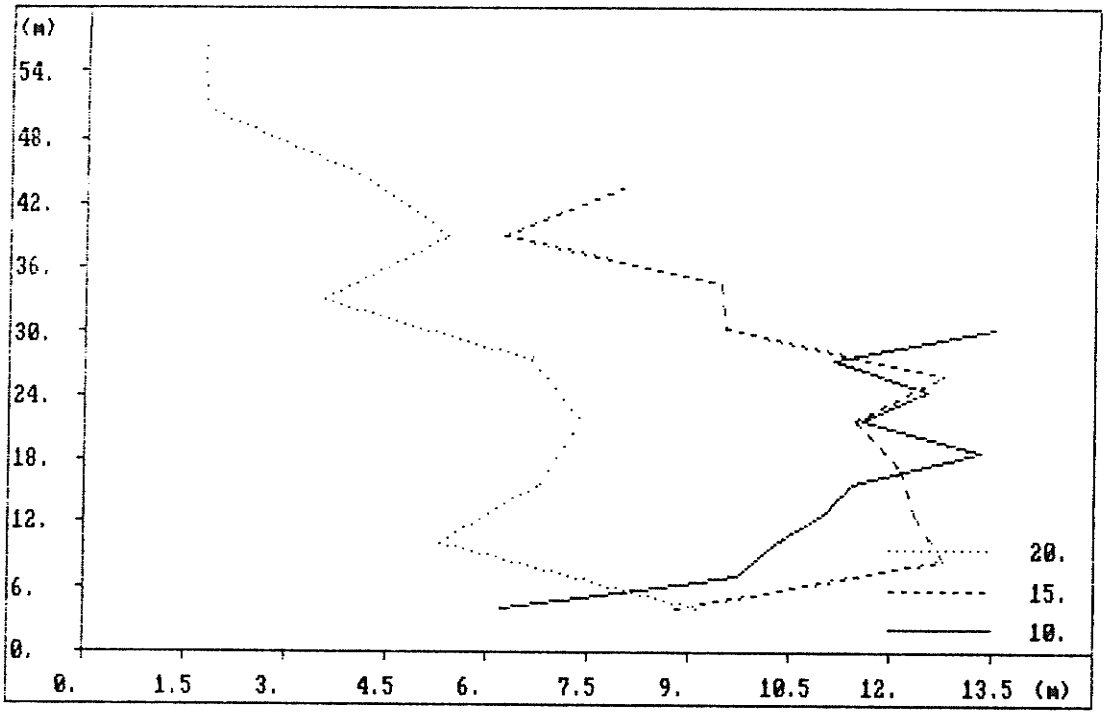


Figure 4.6 Transfer Characteristic curves (W_H=4 and Phase Shift $\alpha=0^0$)

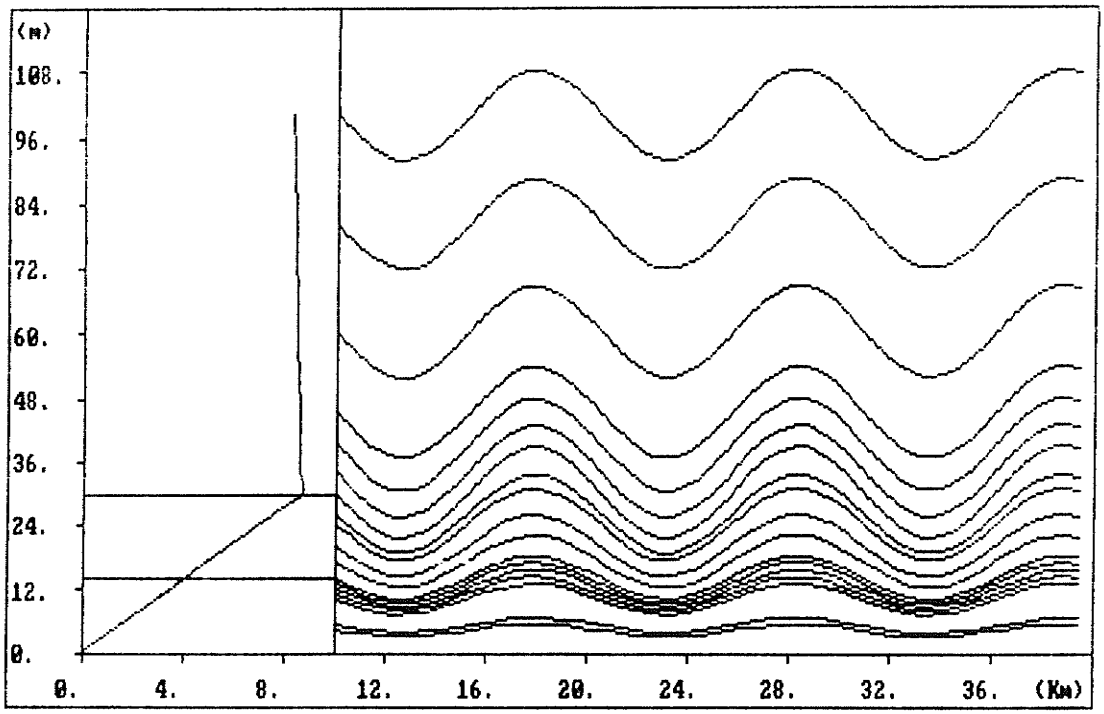


Figure 4.7 Gravity Wave Shell Model (W_H=4 m phase shift $\alpha=90^0$)

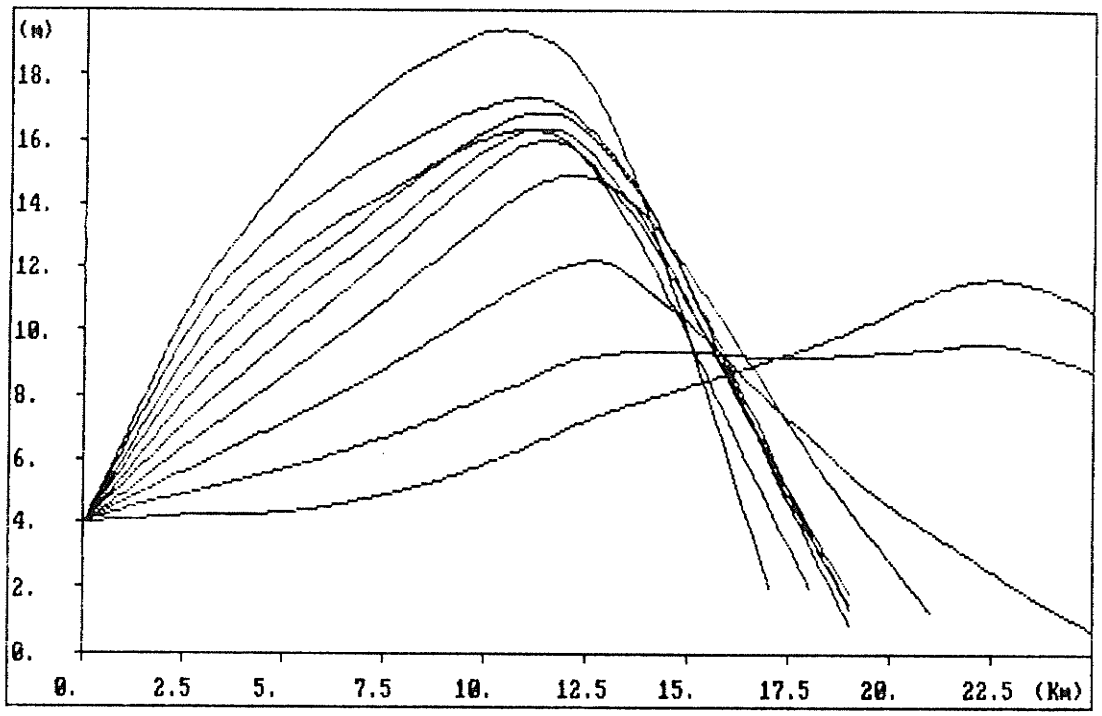


Figure 4.8 Ray Tracing (W_H=4 and Phase Shift α=90^0)

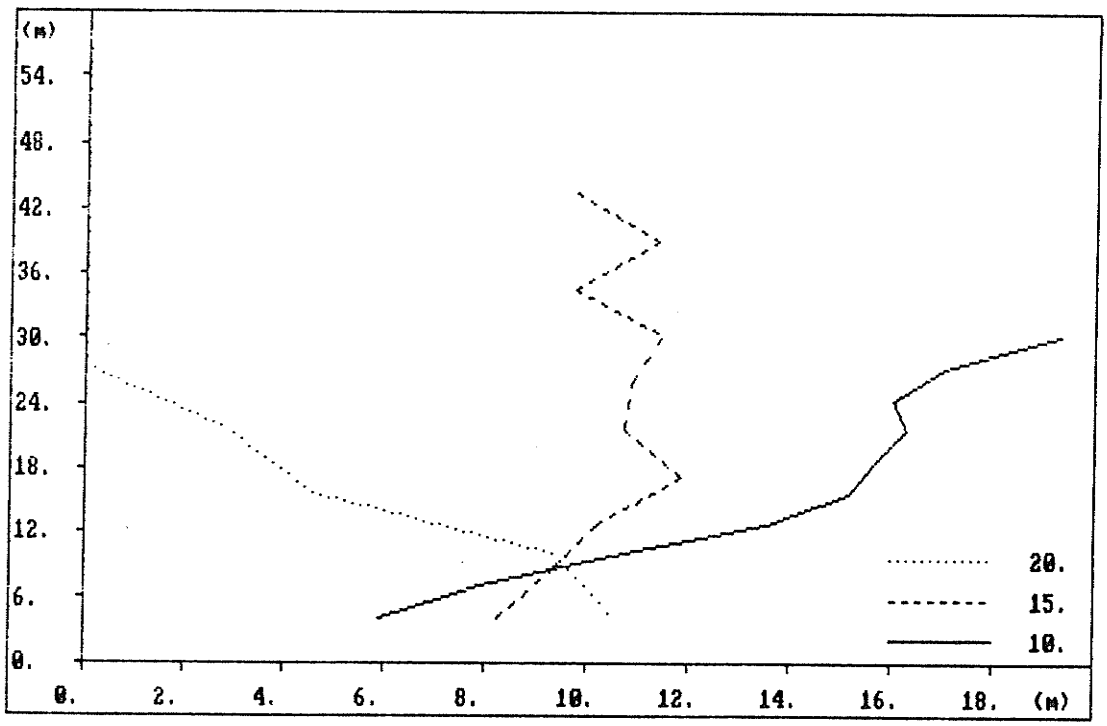


Figure 4.9 Transfer Characteristic curves (W_H=4 and Phase Shift α=90^0)

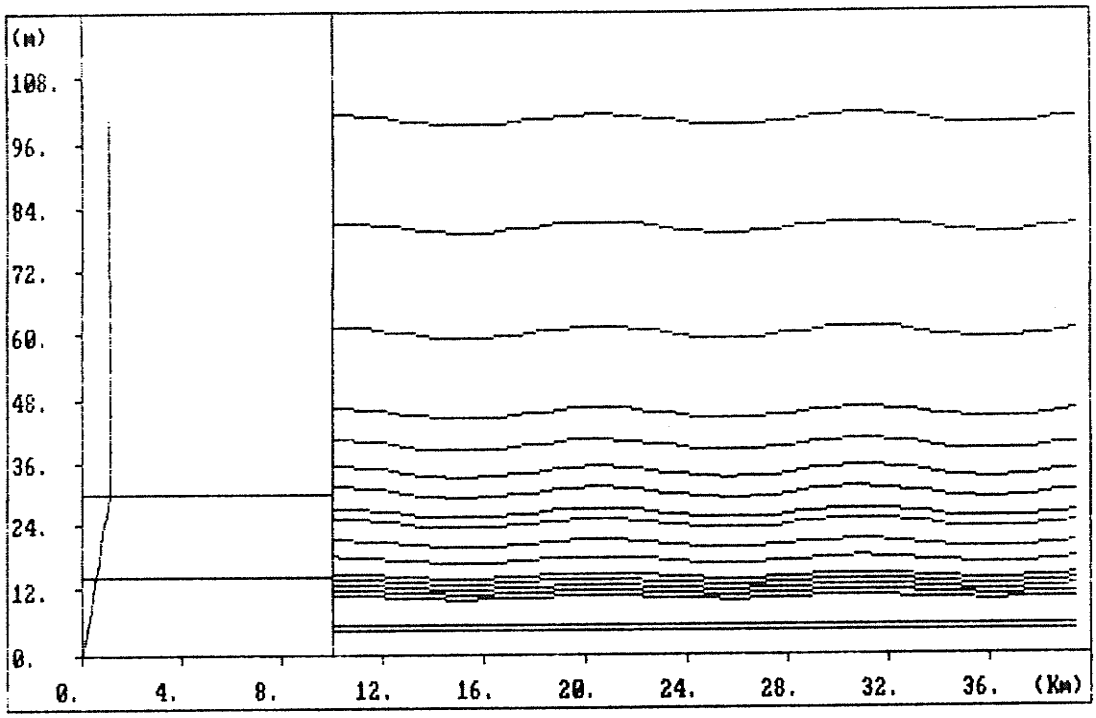


Figure 4.10 Gravity Wave Shell Model (W_H=0.5 m phase shift $\alpha=0^0$)

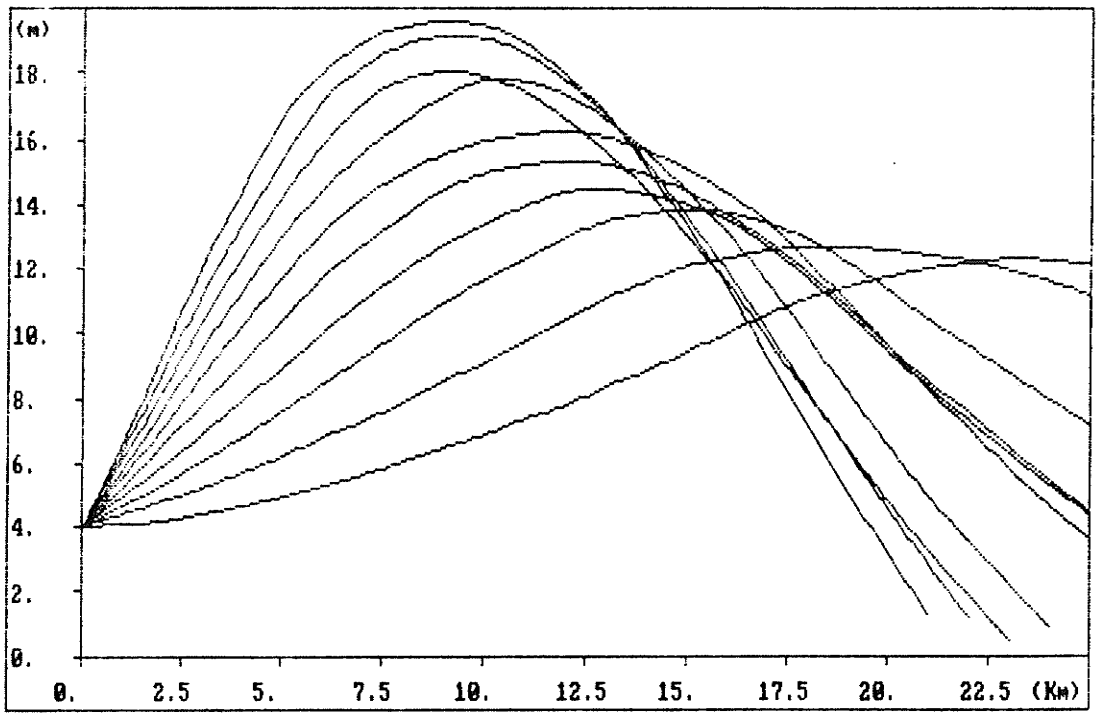


Figure 4.11 Ray Tracing (W_H=0.5 and Phase Shift $\alpha=0^0$)

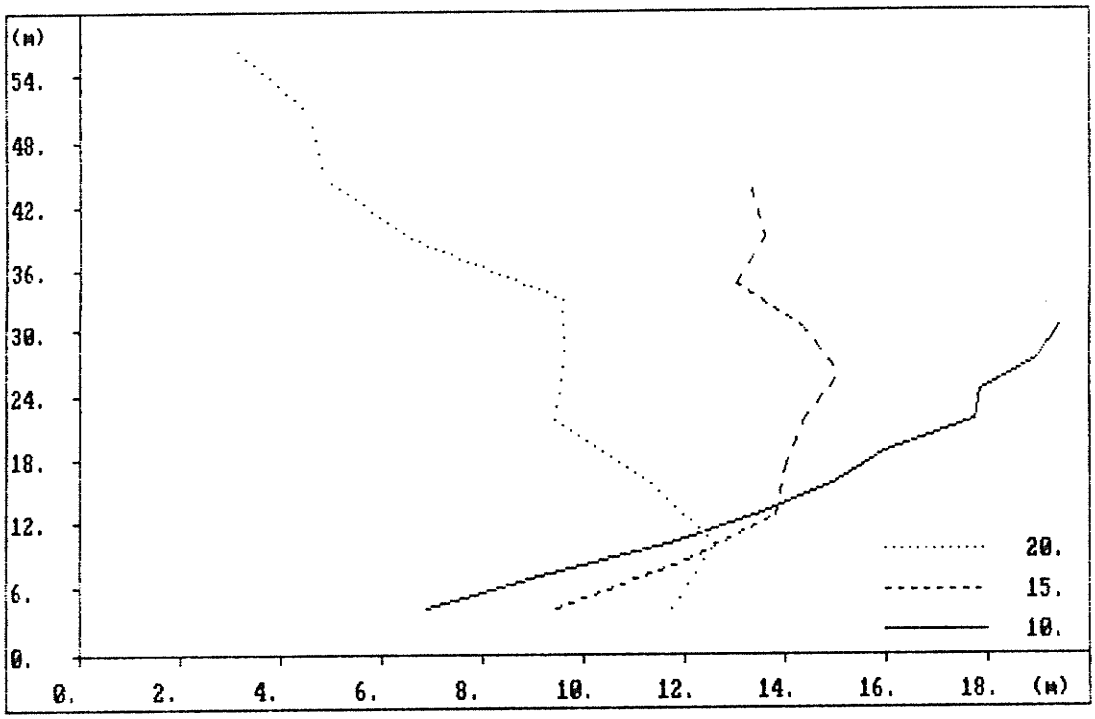


Figure 4.12 Transfer Characteristic curves (W_H=0.5 and Phase Shift $\alpha=0^0$)

From these results, we can see that ray tracing is very sensitive to gravity wave amplitude and phase shift. As predicted, when gravity wave amplitude W_H at z=H is small, the temperature gradient between the two shell stays almost constant, thus ray paths are evenly bent in the x direction. However when W_H is large, the temperature gradient fluctuates a bit, thus giving uneven ray bending effect. The effect of gravity phase shift is hard to predict since a phase shift will generate a different atmospheric gravity wave model; thus formation of ray paths is affected differently. Similarly, increasing gravity wave amplitude will increase the complexity of ray tracing process since it produces more variation in temperature gradient and intersections between a light ray and gravity wave shells. In order to have a better observation on the effect of changing gravity wave amplitude and wave length on TC curve formation, we will vary only the parameter of interest and keep all the rest of the parameters fixed and plot them on the same graph for easy comparison. A temperature profile named JO4, which is a smoother version of JOSA83 temperature profile obtained from a paper[8] named "Inversion of Superior Mirage Data" by Professor Lehn, will be used as the atmospheric temperature profile for ray tracing process in order to get smoother TC curves. Figure 4.13 shows the JO4 atmospheric temperature profile. Having the JO4 profile as the input, TC curves for different gravity wave length and

amplitude are shown in Figure 4.14 and 4.15 respectively. The following gravity wave parameters are used to produce these TC curves:

. Figure 4.14: Variation in gravity wave length.

- Gravity wave amplitude at z=H; $W_H = 2$ m
- Parabolic ray arc step size of 0.25 km
- Phase shift $\alpha = 0^0$.

. Figure 4.15: Variation in gravity amplitude.

- Gravity wave length $\lambda = 15000$ m
- Parabolic ray arc step size of 0.25 km
- Phase shift $\alpha = 0^0$.

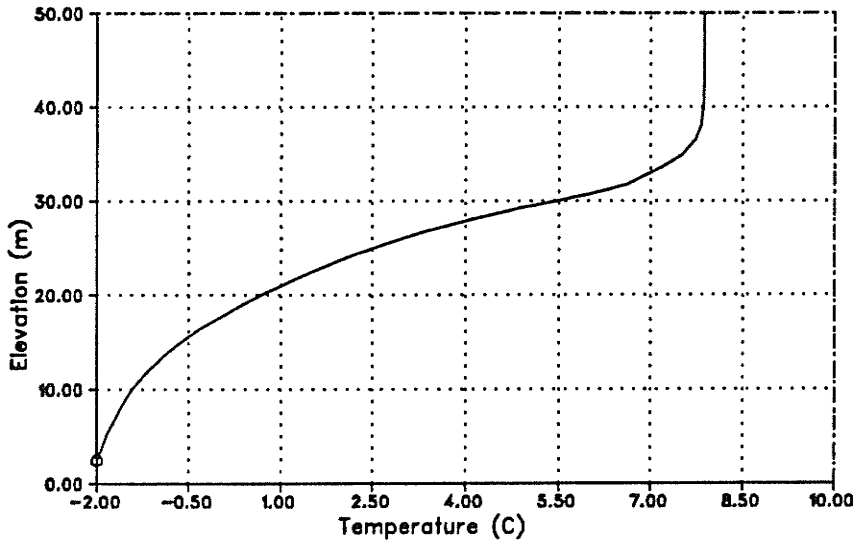


Figure 4.13 J04 Temperature Profile

From these two Figures, we can conclude that increasing gravity wave amplitude or decreasing wave length will increase the complexity in ray tracing process thus increasing the jaggedness of the TC curve. These jagged TC curves will produce jagged edges in mirages which will be seen in both actual and simulated mirages in the later section. As mentioned above, variations in gravity wave phase shift produce major changes in ray paths and the effect of phase change supports the theory of "waves travelling in the atmosphere" therefore it is more interesting to observe the effect of phase change associated with mirage simulation in the following section.

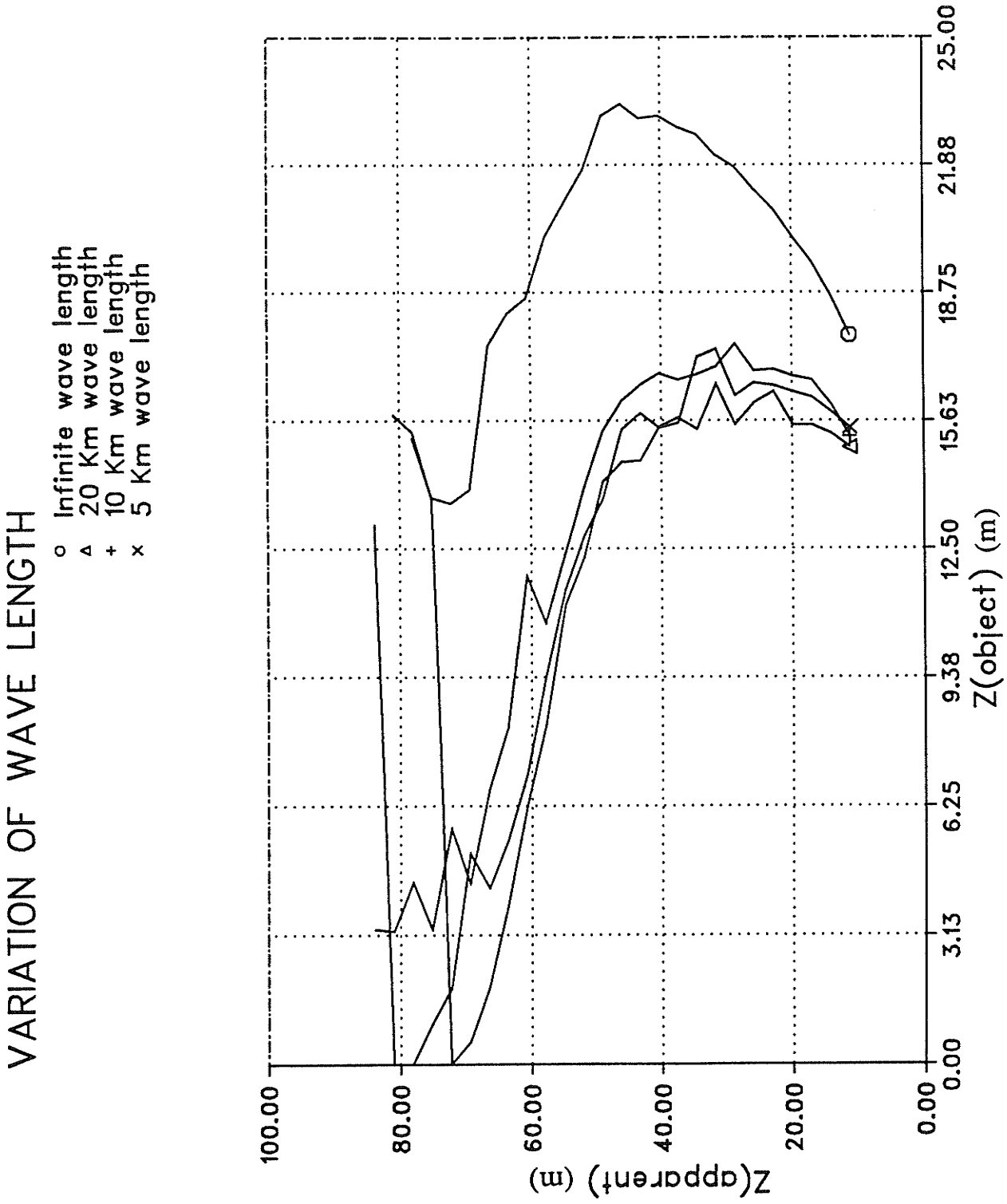


Figure 4.14 TC Curves of Different Gravity Wave Length λ

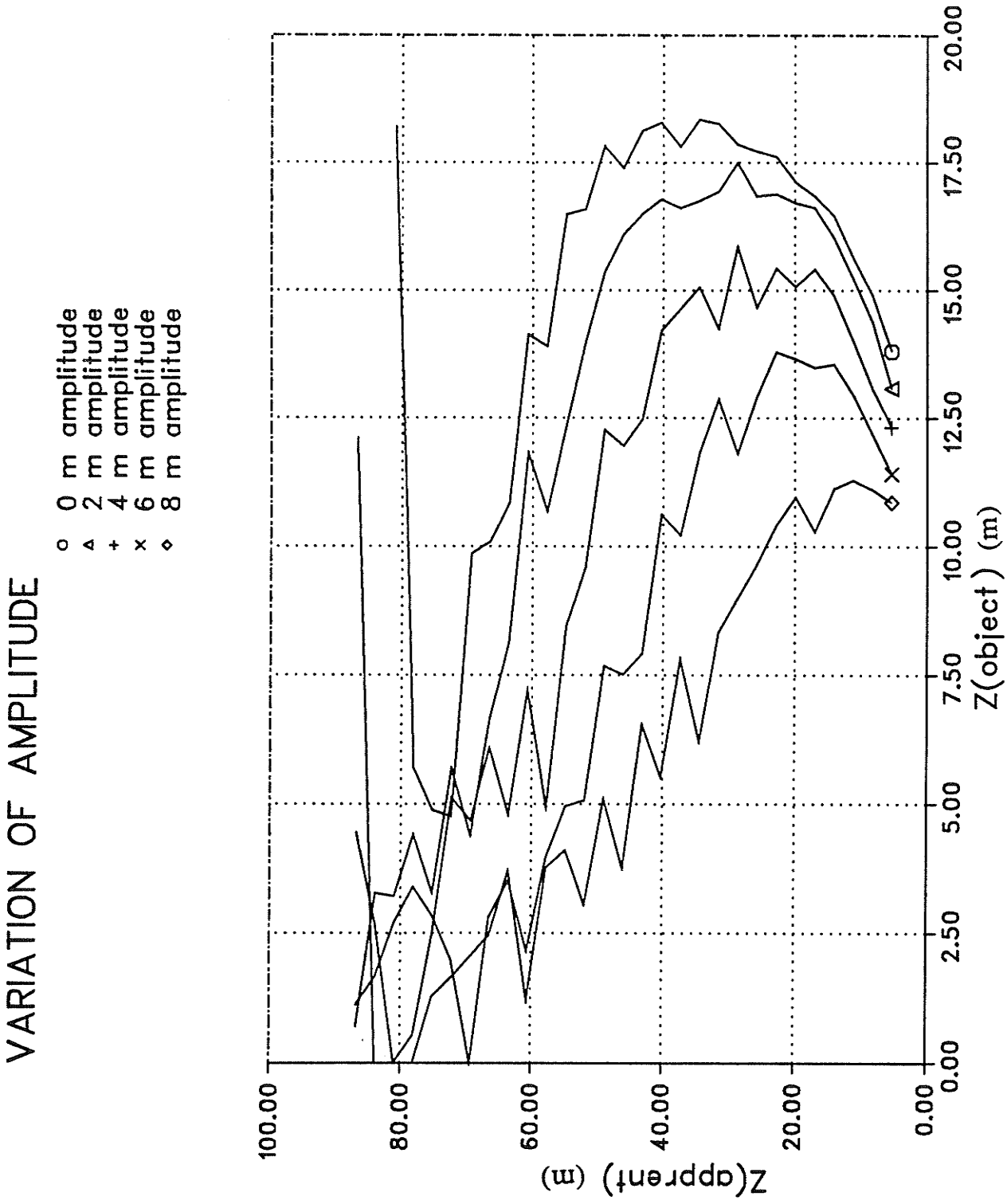


Figure 4.15 TC Curves of Different Gravity Wave Amplitude W_H

4.4 Simulation Of A Mirage Observation

4.4.1 Observed mirages

As an example, Figure 4.16 shows an normal view of 18.7 m hill, named Whitefish Summit, photographed, over the sea ice from a distance of 20 km at Tuktoyaktuk, Northwest Territories, Canada in May 20, 1979, 19:36 h .

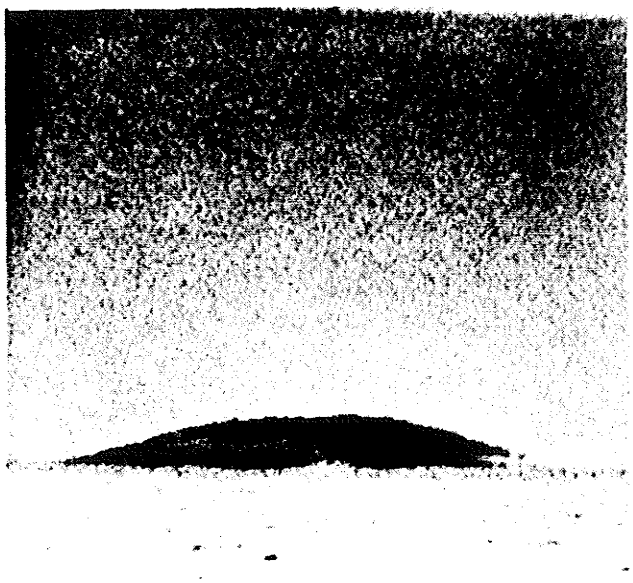


Figure 4.16 Normal View Of Whitefish Summit

To support the theory of "waves in the atmosphere", Figure 4.17 shows the superior mirages of Whitefish Summit which were photographed from the same distance of 20 km with a camera elevation of 2.5 m above the sea ice. The horizontal time scale shows the exact time at which each photo was taken thus giving readers a good idea of how a mirage changes with respect to time.

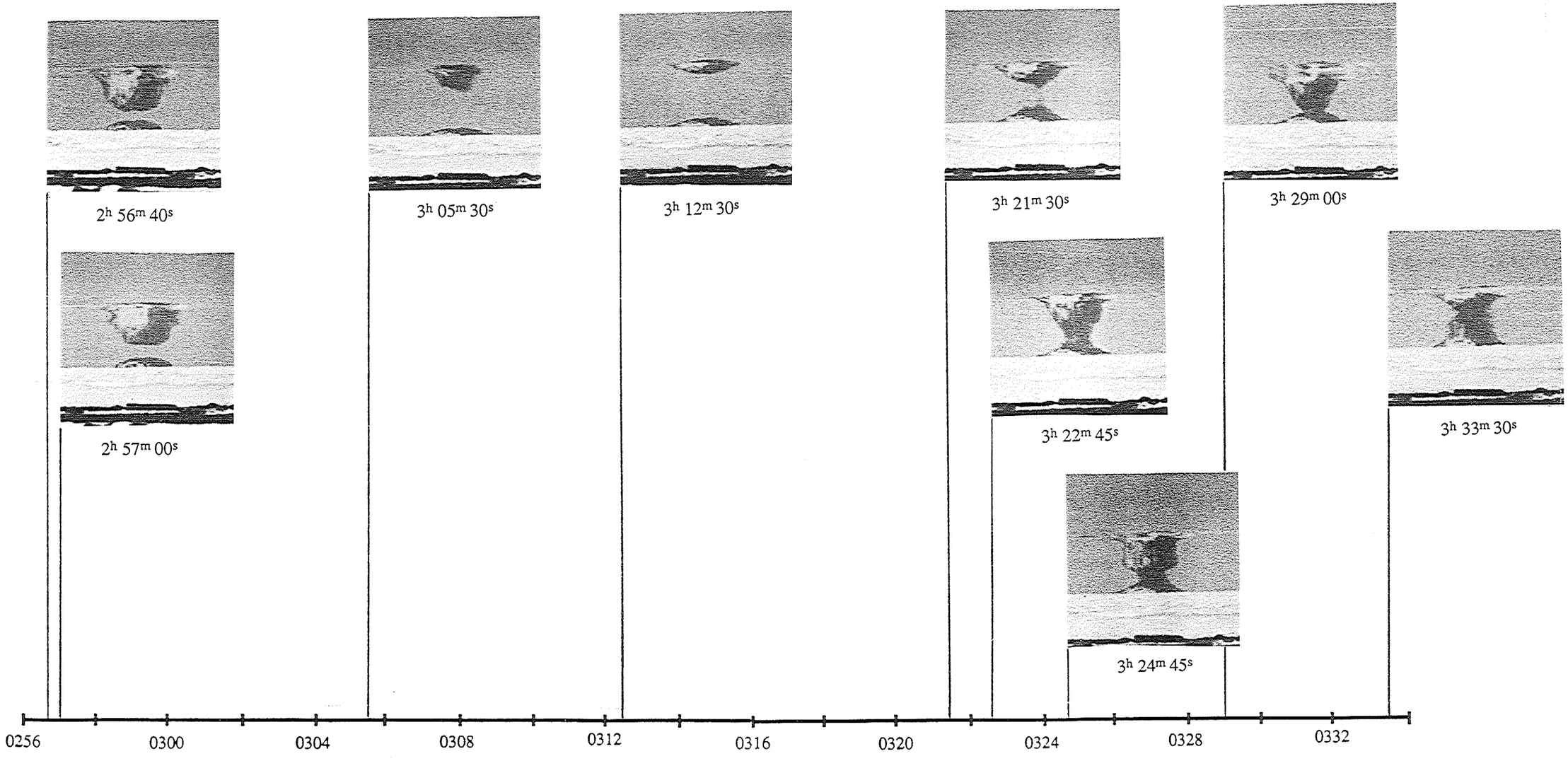


Figure 4.17 Superior Mirages of Whitefish Summit

4.4.2 Mirage Simulation

To create a mirage corresponding to a specific TC curve, a mirage simulation program[9] which maps rows of pixels of a digitized image of an object in real space into mirage space, was developed by Wesley Friesen in 1988. This simulation procedure requires a digitized image of the object of interest and information on the object's real height (m), location and the height of the object in the digitized image (row number) and the desired TC curve. Using this information, the rows in the area of interest (which is defined as the number of rows that contain the object) are mapped into mirage space as shown in Figure 4.18.

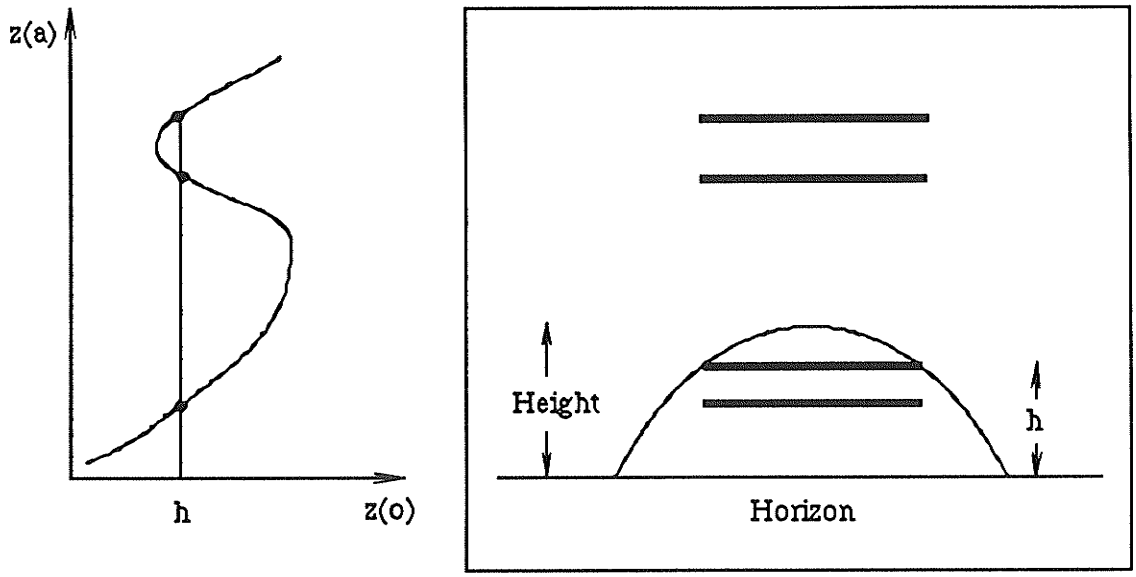


Figure 4.18 Mapping of an Object into Mirage Space

As an example, all the pixels in row "h" of the area of interest in Figure 4.18 are mapped into 3 rows of mirage space indicated by black lines. All the rows in the area of interest are similarly mapped into mirage space to complete an image of mirage.

This mirage simulation program will help to understand further the effect of variation of gravity wave parameters, the accuracy of atmospheric gravity wave model and the relation between TC curves and mirages. In order to justify the accuracy of our atmospheric gravity wave model, the photograph of Whitefish Summit is chosen as the

input to the mirage simulation program so that these simulation results can be compared with the actual mirages shown earlier in Figure 4.17. The simulated mirage images of 640X480 pixels and their corresponding TC curves are shown in Figures 4.19 to Figure 4.26 for a complete phase cycle of 360^0 by an incremental step of 45^0. The following gravity wave and ray tracing parameters are used to produce these TC curves:

. Gravity wave amplitude $W_H = 2$ m (at z=h)

. Gravity wave length $\lambda = 15$ km

. Initial ray angle from 0 to 15 minutes with a step of 0.5 minute

. Parabolic ray arc step of 0.25 km

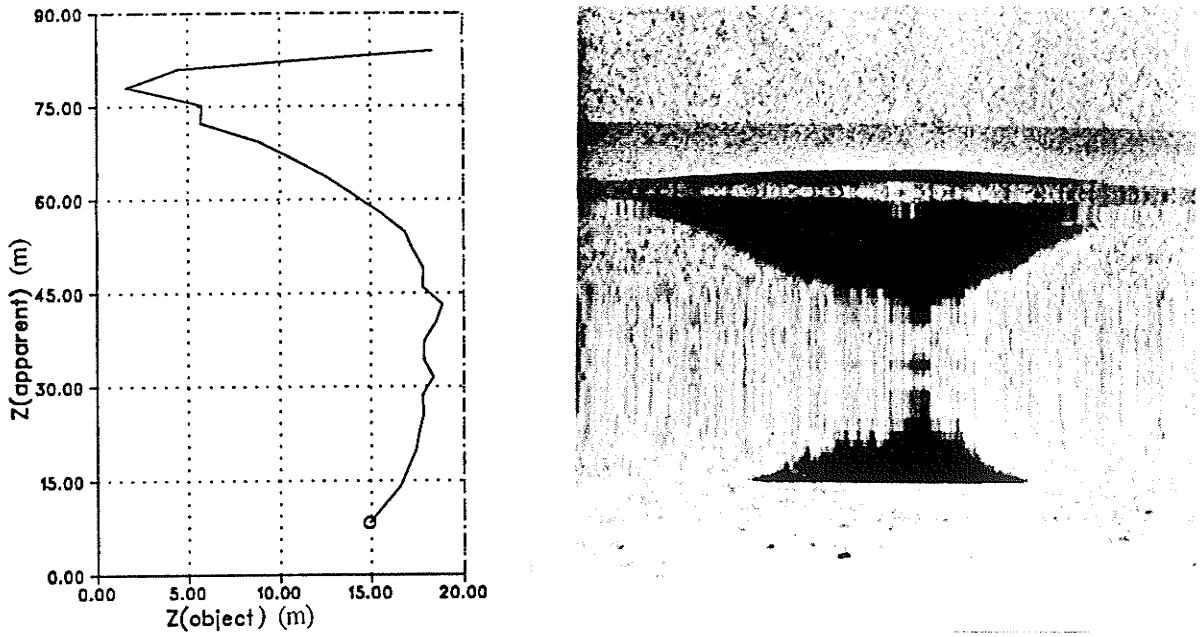


Figure 4.19 TC Curve and Mirage Simulation Image for $\alpha = 0^0$

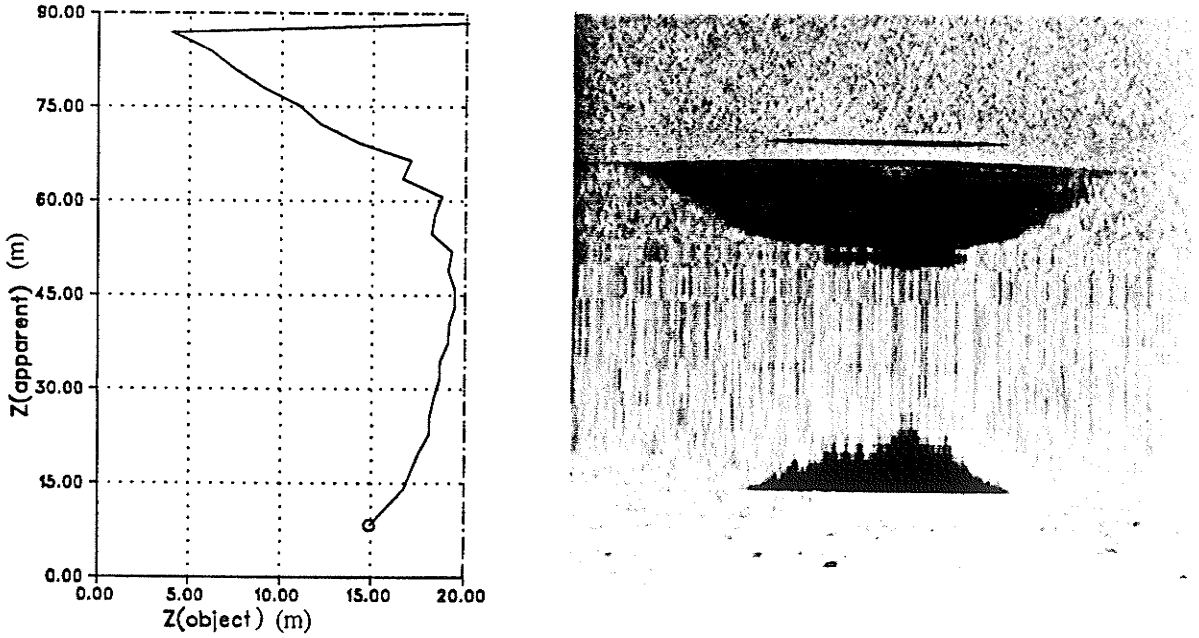
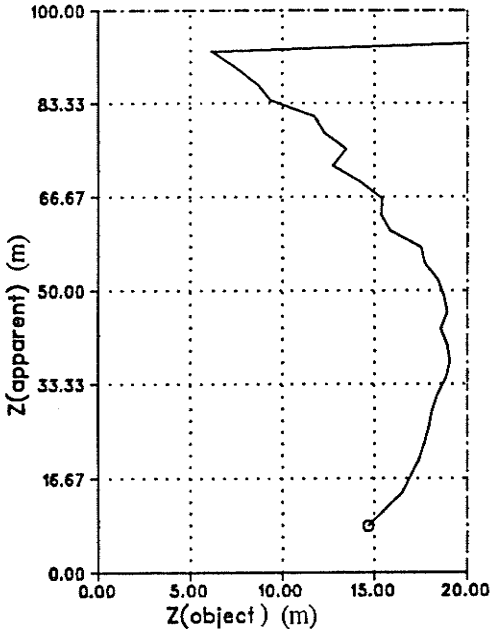


Figure 4.20 TC Curve and Mirage Simulation Image for $\alpha = 45^0$



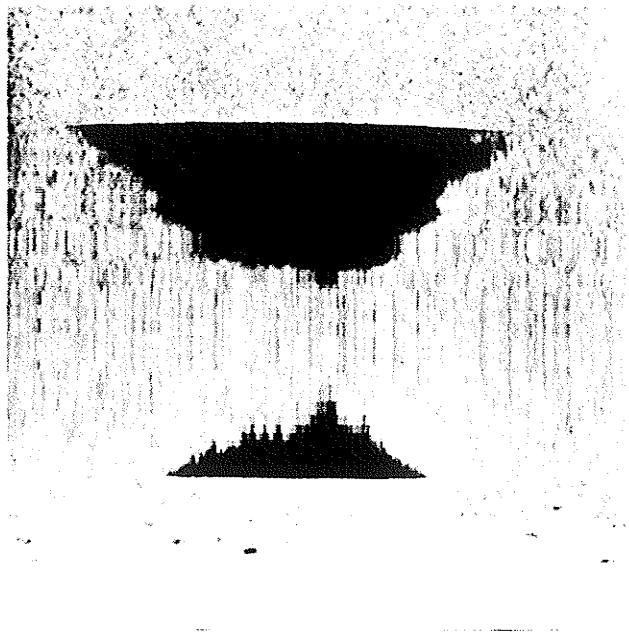
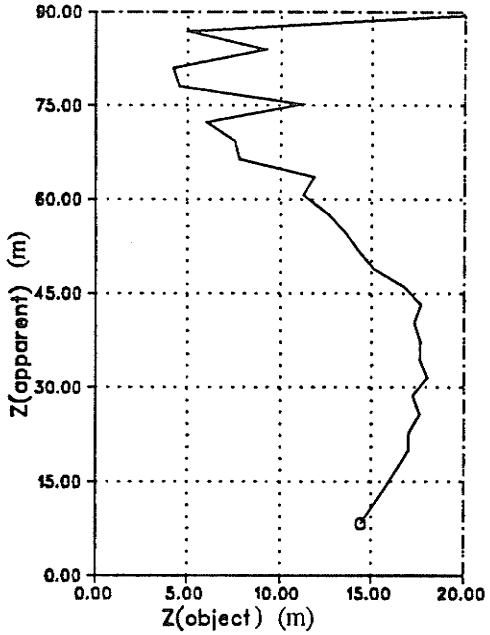


Figure 4.21 TC Curve and Mirage Simulation Image for $\alpha = 90^0$



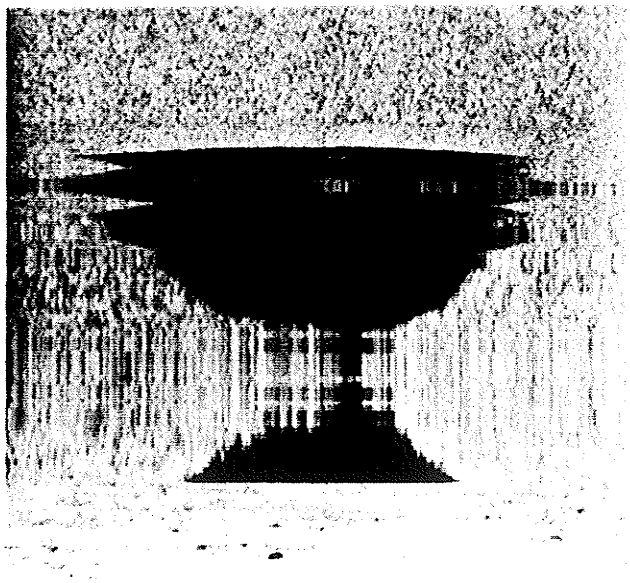


Figure 4.22 TC Curve and Mirage Simulation Image for $\alpha = 135^0$

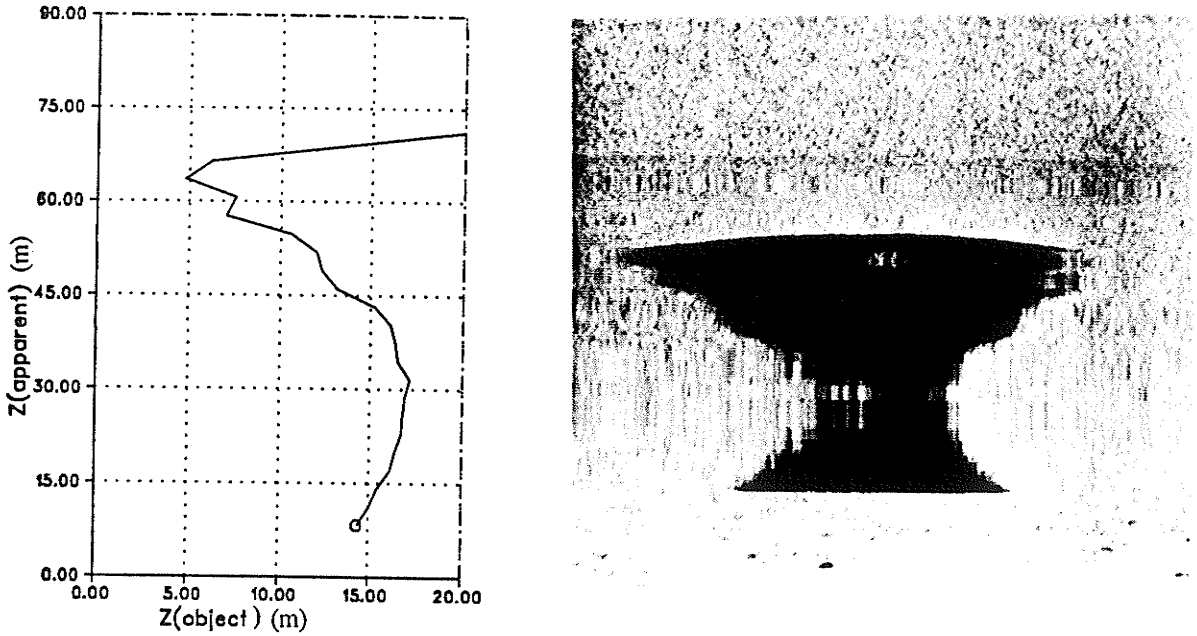


Figure 4.23 TC Curve and Mirage Simulation Image for $\alpha = 180^0$

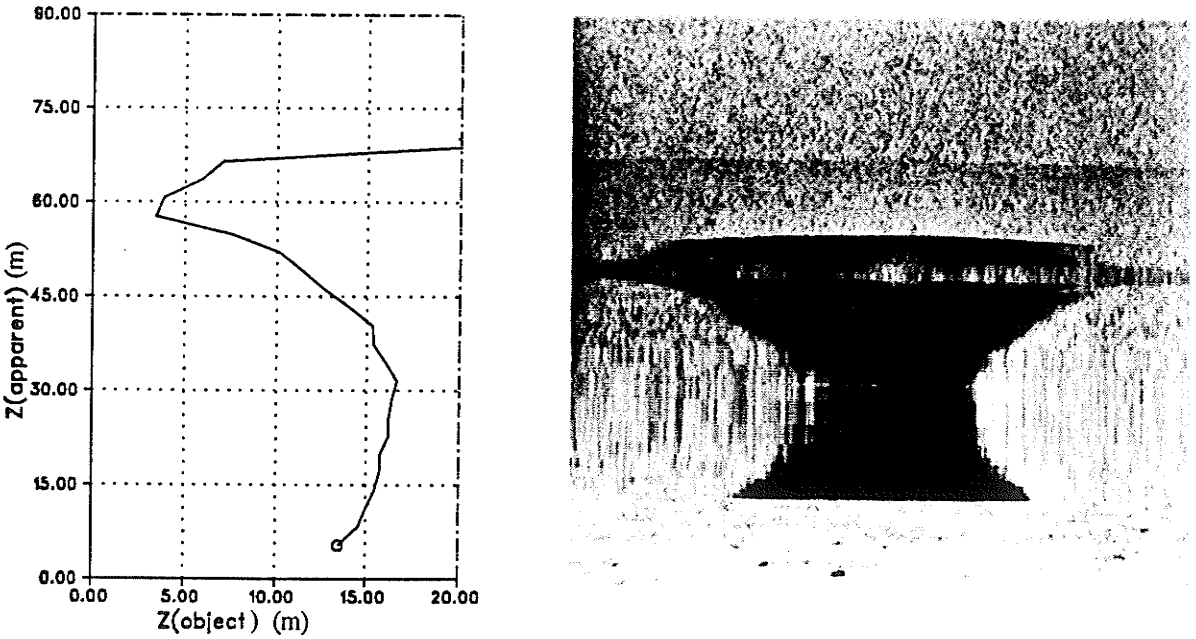
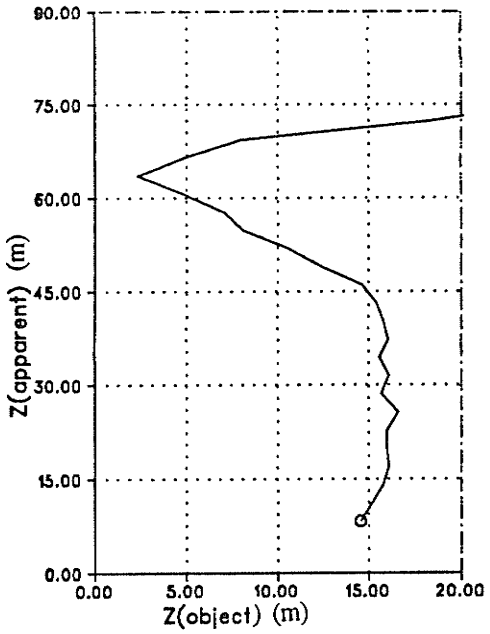


Figure 4.24 TC Curve and Mirage Simulation Image for $\alpha = 225^0$



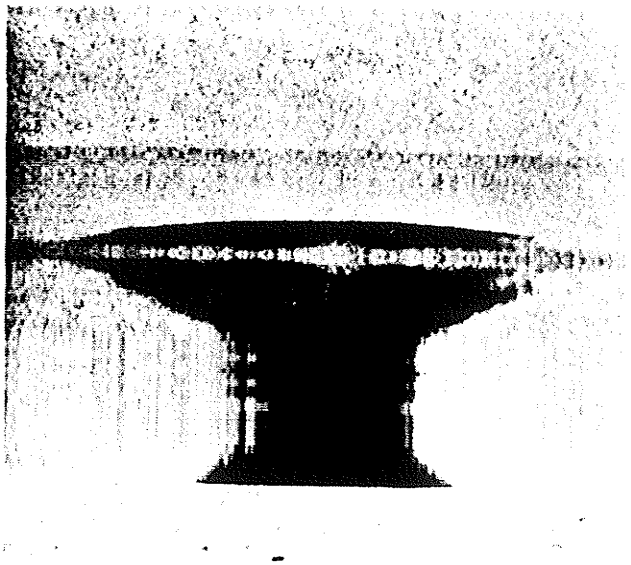
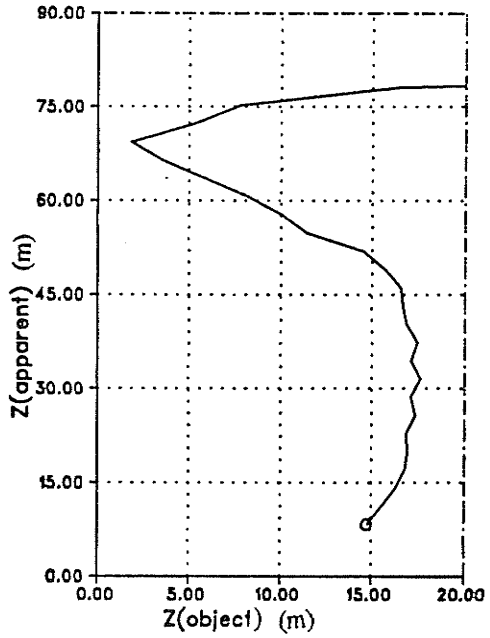


Figure 4.25 TC Curve and Mirage Simulation Image for $\alpha = 270^0$



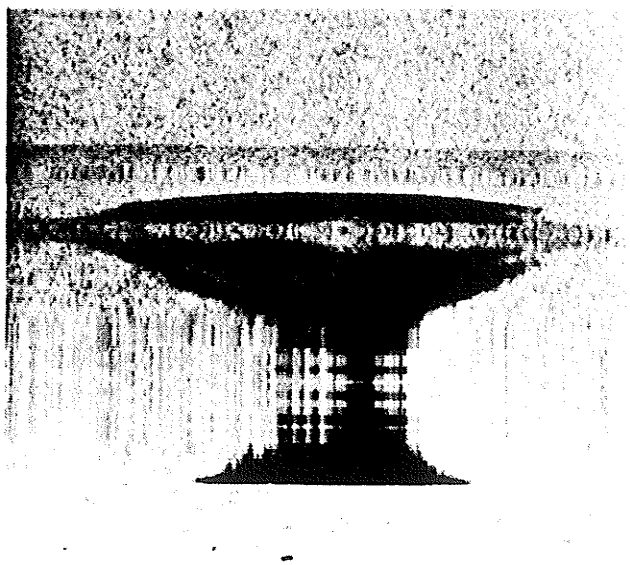


Figure 4.26 TC Curve and Mirage Simulation Image for $\alpha = 315^0$

Comparing these simulation results to the actual mirages of Figure 4.17, we can see that some of the simulated mirages show a very close fit to the observed ones. In order to get a perfect fit one might try to adjust the gravity wave parameters. However, sensitivities of these parameters make the adjustment quite difficult and hard to predict a good step size. But most importantly, these results have proved the validity of our atmospheric gravity wave model.

Chapter 5
Inversion of Mirage Data

5.1 Introduction

Previous chapters concern the forward problem of ray tracing, ie. given a temperature profile, we can trace light rays to obtain images of the surroundings which would appear under the given atmospheric temperature through a "transfer characteristic" curve. In this chapter, the inverse problem of ray tracing is investigated. Given a TC curve, we would like to know the condition of the atmospheric temperature that produces such a mirage. Due to the complication of gravity wave shell model which involves selection of gravity wave amplitude at z=H, we will limit the inverse problem of ray tracing to Lehn's model of "concentric spherical shell".

To solve the inverse problem of ray tracing, first we need to find a mathematical equation which can represent the atmospheric temperature profile. As mentioned earlier, we are only interested in atmospheric temperature profile which has an inflection point. In this type of curve, the temperature gradient is almost symmetrical about its inflection point. By judging this nature of temperature profile, we can suggest the use of a Fourier Series to represent the derivative of the temperature curve. Thus the temperature curve can be represented by:

$$T(z) = \int \left[\frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos \frac{2\pi z}{L} + b_k \sin \frac{2\pi z}{L} \right) \right] dz$$

where

a_k : are Fourier series coefficients (k=0,1,2...)

L : period

or

$$T(z) = \frac{a_0}{2} z + \frac{L}{2\pi} \sum_{k=1}^{\infty} \left(a_k \sin \frac{2\pi z}{L} - b_k \cos \frac{2\pi z}{L} \right) + C .$$

For every temperature profile T(z), a unique TC curve can be found using the ray tracing program. Now the problem is to find the coefficients of T(z) so that it represents the atmospheric temperature profile which produces (as closely as possible) the given TC curve. In other words, we try to find a set of coefficients of T(z) which minimize the sum of square of the difference (or sum of square errors) between the given TC curve and the TC curve produced by T(z). Figure 5.1 shows the difference between given TC and T(z) produced TC.

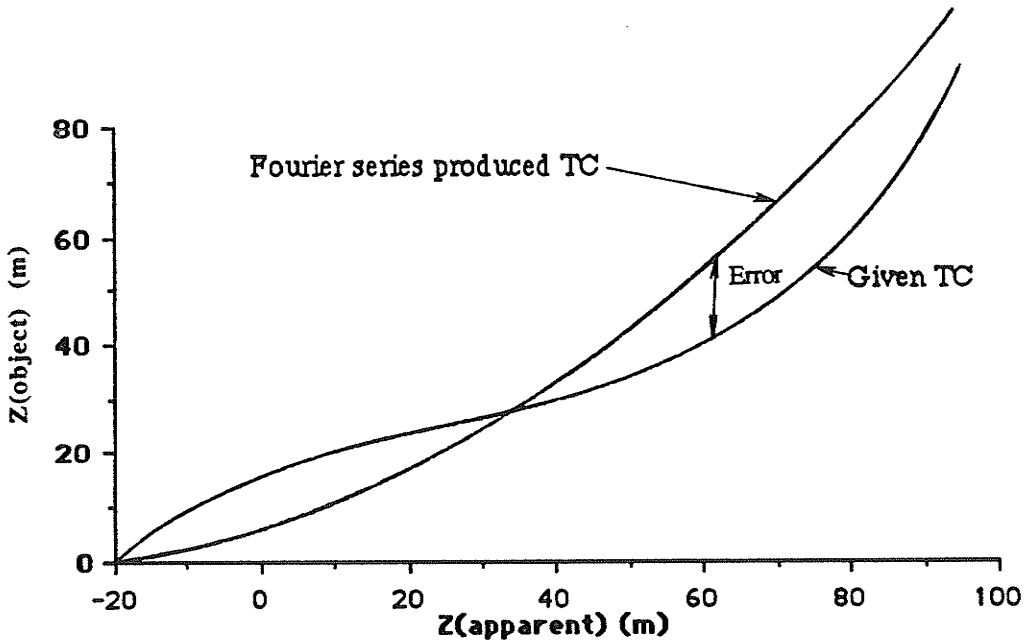


Figure 5.1 Errors between given TC and produced TC

An optimization method namely "Rotating Coordinates" by Rosenbrock[5] (1960) is used to search for the set of coefficients of T(z) which minimizes the sum of TC curve square errors.

5.2 Optimization Method of Rotating Coordinates

The reason for this optimization method having such name is because it incorporates the idea of rotating the coordinate system so that one of the search directions is aligned with

the expected direction which leads to the minimum. This minimizes the number of iterations in the search process. Each iteration in this process consists of two parts:

. A pattern search is used to determine the most suitable search direction. This is done by exploring every search direction to find the current minimum point.

. The vector specified by the starting point and current minimum point within an iteration defines the new search direction vector. The coordinate system is then rotated to align one of the axes with this new search direction.

This method can be demonstrated by the following brief example. Consider a two dimensional search with mutually orthogonal unit vectors at k^{th} iteration denoted by $U_{1,k}$ and $U_{2,k}$ and search step lengths $\delta_{1,k}$ and $\delta_{2,k}$ respectively. Figure 5.2 shows the steps in k^{th} iteration with the starting point at x_k. To help understand the steps in this figure, the following symbols are used:

1(s) : means test point #1 is a sucessful point (ie. function value at point #1 is less than function value at the current search base (point #0))

4(f) : means test point #4 is a failure (ie. function value at point #4 is more than function value at the current search base (point #3)).

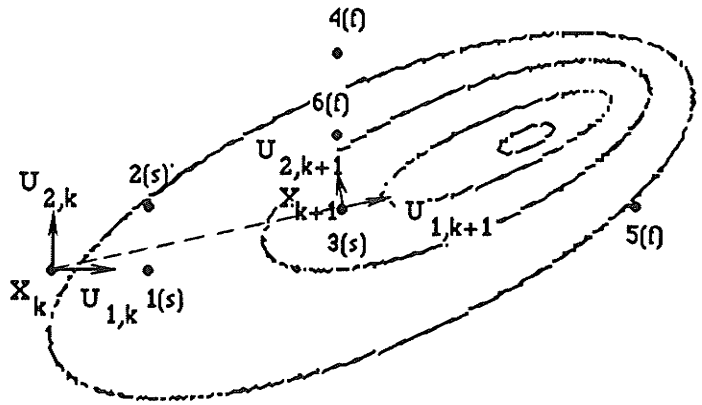


Figure 5.2 Exploratory Search in K^{th} Iteration

The following two parts will explain details of Rosenbrock search steps in this example.

Part 1 : Exploratory search.

a. Test point #1: Following the search direction $U_{1,k}$, the test point #1 at $(x_k + \delta_{1,k}U_{1,k})$ is a succesful point, thus increasing the current search step length to $2\delta_{1,k}$ and moving the new search base to point #1.

b. Test point #2: Following the search direction $U_{2,k}$, the test point #2 at (point #1 + $\delta_{2,k}U_{2,k}$) is a successful point, thus increasing the current search step length to $2\delta_{2,k}$ and moving the new search base to point #2.

c. Test point #3: Following the search direction $U_{1,k}$, the test point #3 at (point #2 + $2\delta_{1,k}U_{1,k}$) is a successful point, thus increasing the current search step length to $4\delta_{1,k}$ and moving the new search base to point #3.

d. Test point #4: Following the search direction $U_{2,k}$, the test point #4 at (point #3 + $2\delta_{2,k}U_{2,k}$) is a failure, thus reducing the current search step length to $\delta_{2,k}$ and maintaining the current search base at point #3.

e. Test point #5: Following the search direction $U_{1,k}$, the test point #5 at (point #3 + $4\delta_{1,k}U_{1,k}$) is a failure, thus reducing the current search step length to $2\delta_{1,k}$ and maintaining the current search base at point #3.

f. Test point #6: Following the search direction $U_{2,k}$, the test point #6 at (point #3 + $\delta_{2,k}U_{2,k}$) is a also failure, thus reducing the current search step length to $0.5\delta_{2,k}$ and maintaining the current search base at point #3.

Now, exploratory searches in both directions at base #3 are all failures. It is time to move on to part two which rotates the coordinate system to find a better search direction and begin the next iteration.

Part 2: Coordinates rotation.

In the next iteration, the coordinate system is rotated such that one of the search directions aligns with the vector defined by $(x_{k+1} - x_k)$. This is done by the Gram-Schmidt orthogonalization process[7] as follows.

Consider n-dimensional search, with total movement $d_1, d_2, \ldots d_n$ in each search direction within one iteration. Define a new set of vectors $V_1, V_2, \ldots V_n$ as follows:

$$V_n = d_n U_n$$

and

$$V_i = d_i U_i + V_{i+1} \quad \text{for } i = n-1, n-2, \ldots 1$$

Then the new set of orthonormal direction vectors is given by:

$$U_1^{k+1} = \frac{V_1}{|V_1|}$$

$$U_i^{k+1} = \frac{W_i}{|W_i|} \qquad i = 2,3 \ldots n$$

where

$$W_i = V_i - \sum_{j=1}^{i-1} \left(V_i^T U_j^{k+1} \right) U_j^{k+1}$$

5.3 Implementation

The optimization and circular shell model for ray tracing are both written in Fortran H. The ray tracing program was implemented by Professor Lehn, and the Rosenbrock method of rotating coordinates program was given in the "Optimization Methods Using Computer aided Design" course offered by Professor Menzies. Both of these programs are modified by the author to suit the specific use of TC curve fitting. The following explanatory comments and flowcharts are used to explain the TC curve fitting operation.

In this optimization process, every coefficient in the temperature equation T(z) will represent a search direction. Increasing the number of coefficients should produce better results in curve fitting. However, the time required to search will also increase. Figure 5.3 shows a brief flowchart of the TC curve fitting operation. In this process, the ray tracing program combined with a square errors summation routine is treated as the objective function evaluation equation for the Rosenbrock optimization method. For a current set of Fourier series coefficients which represent a temperature profile T(z), a TC curve at a fixed distance can be found; then it is compared with the given TC curve to find the summation of square errors.

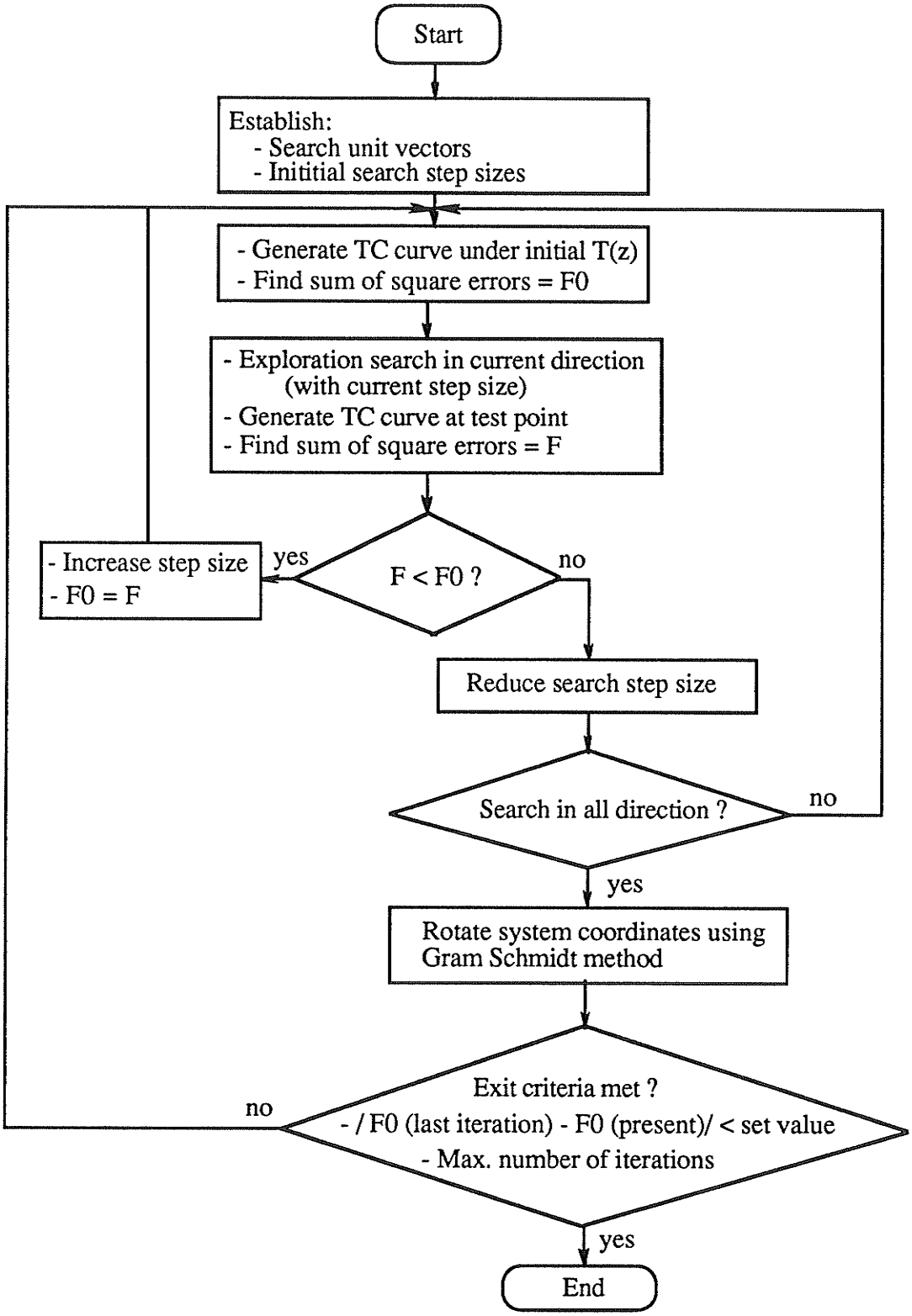


Figure 5.3 Flowchart of Curve Fitting Operation

The Rosenbrock optimization method will search through all the possible combinations of Fourier series coefficients that produce the minimum of summation of square errors.

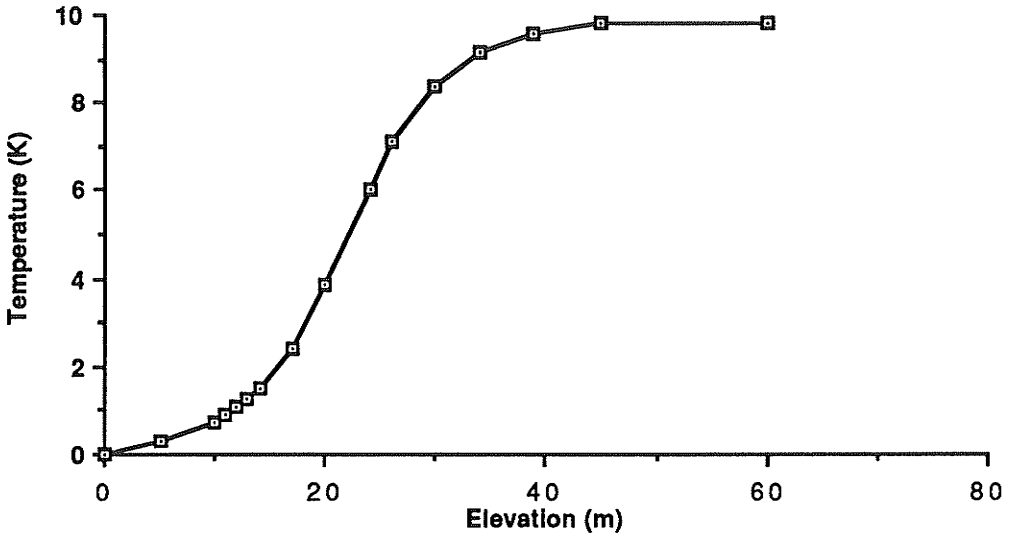


Figure 5.4 "Jm3" Temperature Profile

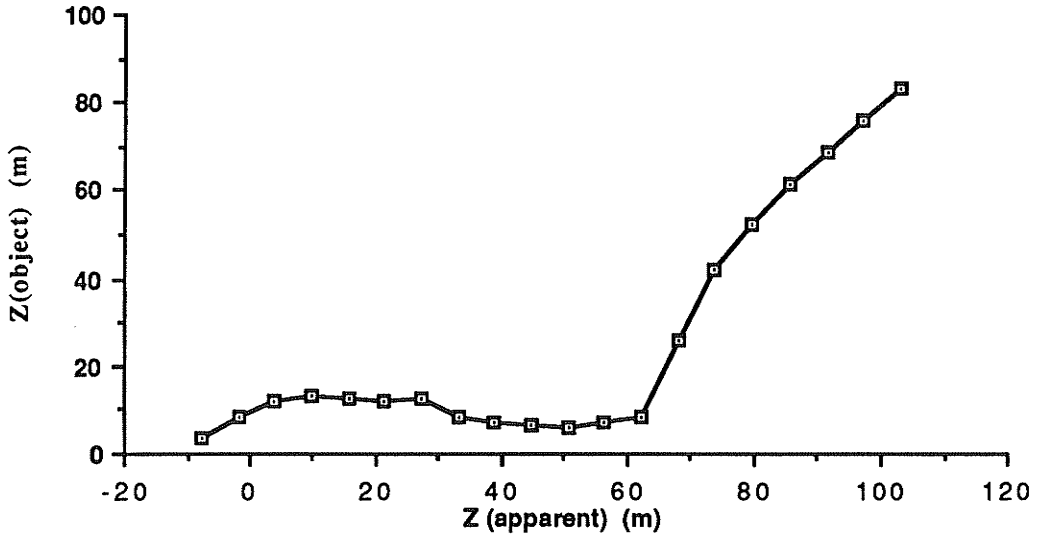


Figure 5.5 Transfer Characteristic Curve at 20 Km (by Jm3)

5.4 Results

In this section, some of the results from the curve fitting process are shown and commented. To generate a testing TC curve, a temperature profile namely "Jm3" is used as the input atmospheric temperature profile for the ray tracing program. Figure 5.4 shows the "Jm3" temperature profile. The TC curve at a distance of 20 Km is generated as shown in figure 5.5. This TC curve is then used as the given TC curve for "Inversion of mirage data" program described in previous sections. After 125 iterations, the Rosenbrock optimization method produces the following results:

. For Fourier series with 2 harmonic terms:

$a_0 = 0.3838$

$a_1 = -0.2113$

$b_1 = 0.0840$

$a_2 = 0.1074$

$b_2 = -0.0516$

$C = 0.0000$

thus the Fourier series temperature profile T(z) with a period L = 45 m can be expressed as follows:

$$T(z) = \frac{0.38}{2} + \frac{45}{2\pi}\left(-0.21 \sin\frac{2\pi z}{45} + 0.10 \sin\frac{2\pi z}{45} + 0.08 \cos\frac{2\pi z}{45} - 0.05 \cos\frac{2\pi z}{45}\right)$$

This particular temperature profile generates a TC curve which produces a sum of square errors of 35.6.

. For Fourier series with 4 harmonics terms:

$a_0 = 0.3872$

$a_1 = -0.2218$

$b_1 = 0.0620$

$a_2 = -0.0085$

$b_2 = 0.0179$

$a_3 = 0.0919$

$b_3 = -0.0484$

$a_4 = 0.0509$

$b_4 = -0.0139$

$C = 0.0000$

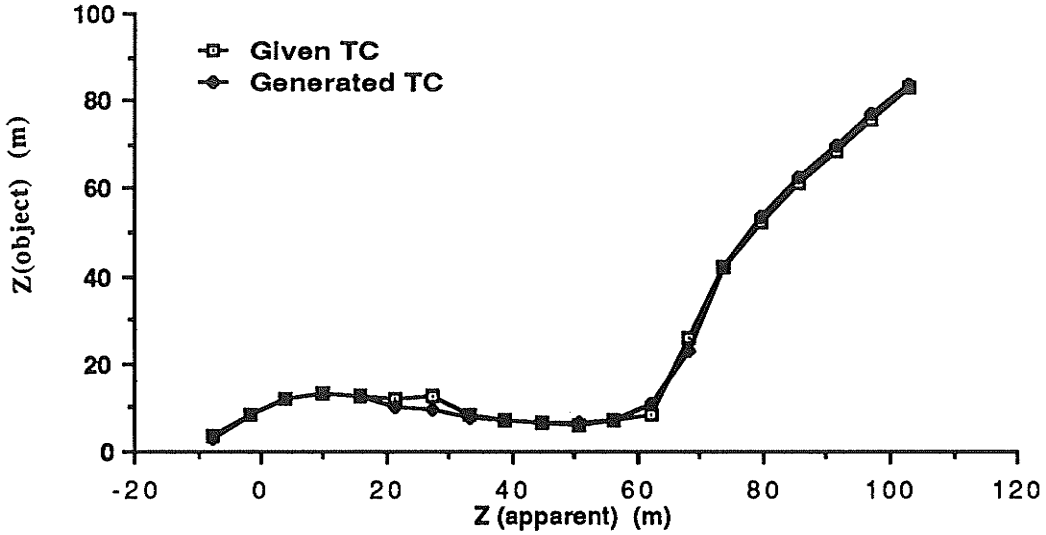


Figure 5.6 Original and 2 Harmonics Fourier Series Generated TC curves

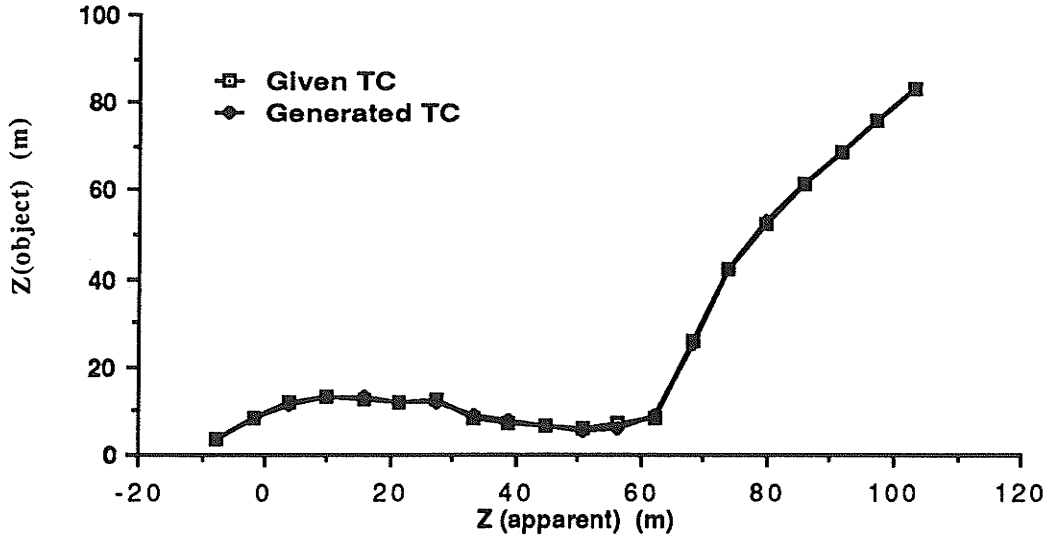


Figure 5.7 Original and 4 Harmonics Fourier Series Generated TC curves

thus the Fourier series temperature profile T(z) can be expressed as follows:

$$T(z) = \frac{0.38}{2} + \frac{45}{2\pi}\left(-0.22 \sin\frac{2\pi z}{45} - 0.01 \sin\frac{2\pi z}{45} + 0.05 \sin\frac{2\pi z}{45} + 0.09 \sin\frac{2\pi z}{45}\right)$$

$$- \frac{45}{2\pi}\left(0.06 \cos\frac{2\pi z}{45} + 0.02 \cos\frac{2\pi z}{45} - 0.05 \cos\frac{2\pi z}{45} - 0.01 \cos\frac{2\pi z}{45}\right)$$

This Fourier series temperature profile generates a TC curve which produces a sum of square errors of 4.2. Figures 5.6 and 5.7 show the given TC curve and TC curves generated by 2-harmonic and 4-harmonic Fourier series temperature profiles respectively. By comparing the two figures of 5.6 and 5.7, we can see clearly that with more harmonic terms in the Fourier expansion of temperature profile a better curve fitting results. At this point, we should compare the Fourier series temperature profile with the original temperature profile of "Jm3" to see how well the Fourier series can be used to represented atmospheric temperature profile. Figure 5.8 shows the original temperature profile of "Jm3" and 4-harmonics Fourier series temperature profile.

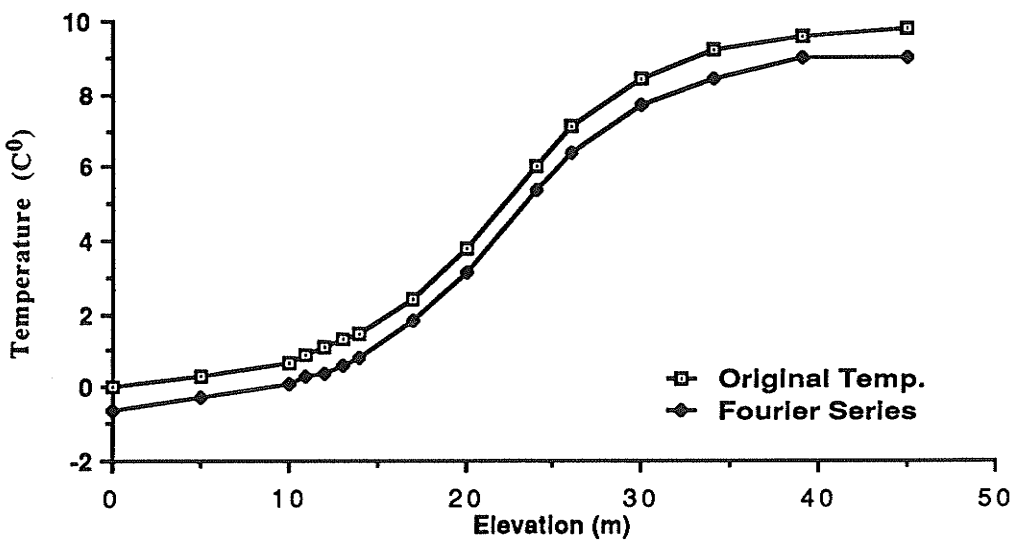


Figure 5.8 Jm3 and 4-Harmonics Fourier Series Temperature Profile

There is a small constant vertical-difference between these two curves, the reason for this small error is due to the nature of ray tracing program. As we have seen in the previous chapters, the path of light rays is largely dependent on the temperature gradient. Therefore

a small shift in temperature profile will not have much effect on the formation of the TC curve. Thus, when the Rosenbrock method reaches this point, it will not try to search any further since its objective of curve fitting has been successfully achieved.

Similarly, another temperature profile named "Bg1" is used to test the Rosenbrock curve fitting method. Figure 5.9 shows Bg1 temperature profile.

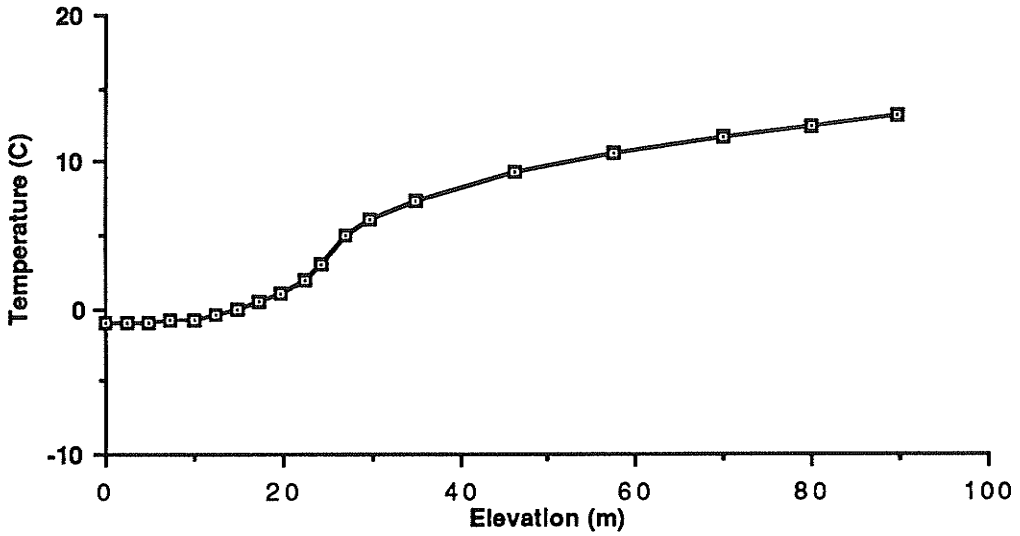


Figure 5.9 Bg1 temperature profile

After 235 iterations, Rosenbrock optimization search produces a 4 harmonic Fourier series temperature profile which generates a TC curve with a sum of square error of 15.87. The coefficients of this 4-harmonic Fourier series temperature profile are as follows:

$a_0 = 0.3998$
$a_1 = -0.2459$
$a_2 = 0.0784$
$a_3 = -0.1047$
$a_4 = 0.0339$
$b_1 = -0.0542$
$b_2 = 0.0256$
$b_3 = 0.0282$
$b_4 = 0.0106$
$C = 0.0000$

Thus the Fourier series temperature profile T(z) with 4 harmonic terms and a period of L=50 m can be expressed as follows:

$$T(z) = \frac{0.40}{2} + \frac{50}{2\pi}\left(-0.24 \sin \frac{2\pi z}{50} + 0.08 \sin \frac{2\pi z}{50} - 0.10 \sin \frac{2\pi z}{50} + 0.04 \sin \frac{2\pi z}{50}\right)$$

$$- \frac{50}{2\pi}\left(-0.05 \cos \frac{2\pi z}{50} + 0.02 \cos \frac{2\pi z}{50} + 0.03 \cos \frac{2\pi z}{50} + 0.01 \cos \frac{2\pi z}{50}\right)$$

Figure 5.10 shows the given TC and 4-hamonic Fourier series temperature profile generated TC curves.

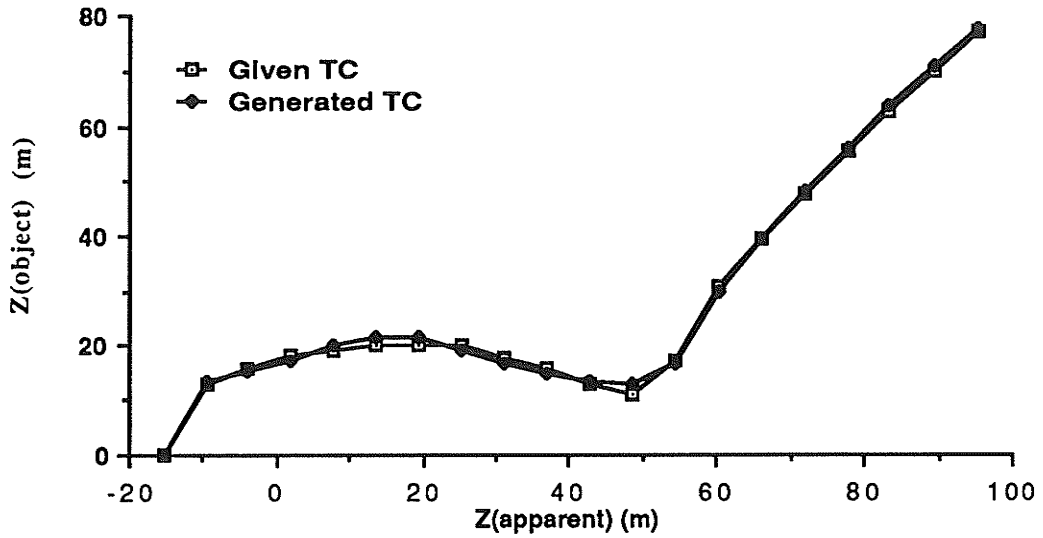


Figure 5.10 Given TC curve and 4-harmonic Fourier series generated TC

Figure 5.11 shows the original Bg1and Fourier series temperature profiles.

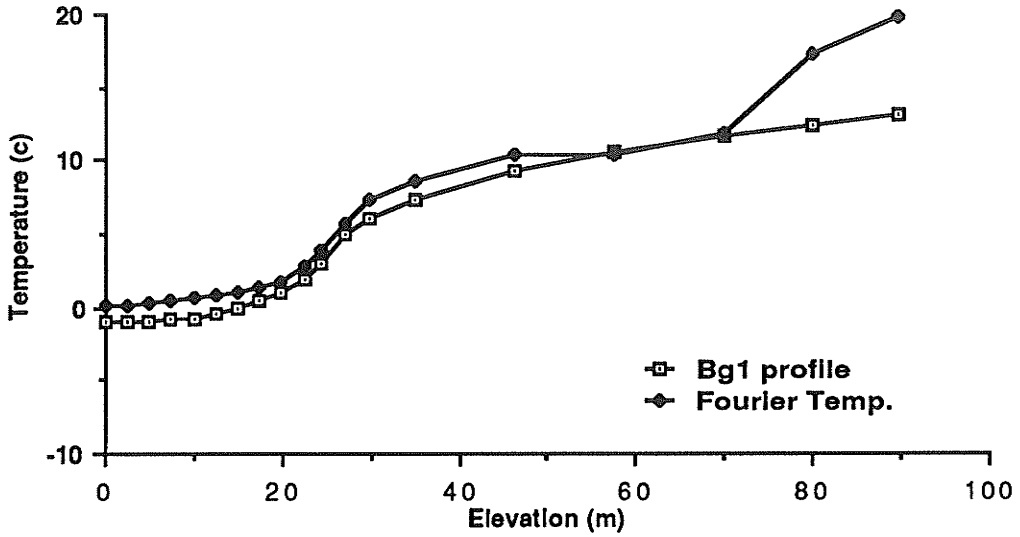


Figure 5.11 Bg1 and 4-harmonics Fourier series temperature profile

As we can see from this figure, for elevations less than 70 m, the two temperature curves have a very close fit, however the Fourier series temperature curve starts to diverge from the original temperature curve for elevations of more than 70 m. This error is again due to the nature of the ray tracing program. All of the light rays contributing to the above TC will not pass throught the region which is higher than 70 m. Therefore all data points in the Bg1 temperature profile that are higher than 70 m will not have any contribution to TC curve formation. These data points can be removed form the temperature profile without making any changes on the ray paths or TC curves. As far as the optimization result is concerned, the Rosenbrock optimization method has done a perfect job of curve fitting.

Chapter 6
Conclusions and Recommendations

6.1 Ray Tracing Process

Among several types of atmospheric wave models, a simple model of gravity wave is selected to build the model for the earth's atmosphere. Based on the atmospheric temperature conditions that cause multiple image mirages, a three section model of gravity wave is adapted to the "parabolic model for the optics of the atmospheric surface layers" developed by Professor Lehn. Several simplifying assumptions have been made in this model: no shear, constancy of vertical wave number in each section and ray angle θ within a layer, and many expression approximations. These simplifications allow implementation of the ray tracing process on a personal computer.

In this ray tracing process, the source of error that is most likely to produce inconsistency of ray paths is the size of parabolic ray arc length (in the x-direction). If this arc length is too large, some of the intersections between ray path and gravity wave shell will fail to be detected, thus introducing errors in calculation of the ray path. To reduce this source of error, a half-length ray arc check is used. The reason to choose this half-length check is because the only time that a ray arc most likely fails to detect an intersection with a gravity wave shell is when it is strongly concave. In this case, the elevation of the peak of the ray arc is either higher (case of concave down) or lower (case of concave up) than the elevation of ray arc's end point. For a parabolic arc, the peak or the minimum is most likely to happen at the middle of its length; therefore a half-length check will take care of a large percentage of these intersection missing cases. However, it is recommended to keep the arc length less than or equal to 1 km and to keep in mind that the smaller the arc length the more accurate the ray tracing. Similarly, the number of data points in the temperature profile also affects the accuracy of ray path calculation. Each data point in the temperature profile represents a gravity wave shell, therefore increasing the number of data points will

increase the number of ray-shell intersections. Thus the more data points in the temperature profile, the more accurate the ray path calculation. However, as the number of gravity wave shells increases, the size of ray arc length should be reduced to avoid missing of intersections.

The mirage simulaltion results have demonstrated the accuracy of the atmospheric gravity wave model. However, a set of gravity wave parameters must be carefully selected and fine adjusted in order to produce a simulated mirage which fits observed data. This process is a very time consuming task due to the high sensitivities of the ray paths to these parameters. More research must be done in this area to find the full relationship between gravity wave parameters and the measurable condition of the atmosphere.

6.2 Mirage Data Inversion Process

The method of mirage data inversion has proven to be a very useful tool in finding the atmospheric temperature profile which can cause an observed mirage condition. This process involves modeling the atmospheric temperature profile in a mathematical form and applying an optimization technique. By judging the nature of atmospheric conditions which cause multiple image mirages, the Fourier series is selected to represent the derivative of temperature curve. Integration is then carried out to find the mathematical expression for the atmospheric temperature profile. Once the number of harmonic terms in the Fourier series is set, a curve fitting method namely "Rosenbrock method of rotating coordinates" can be used to find a set of Fourier series coefficients which minimizes the sum of square errors between given TC and Fourier temperature profile generated TC.

In this process, the only uncertain parameter that we have to select carefully in order to get good results is the period L in the Fourier series expression for the temperature profile. The value of this period must be chosen or adjusted such that the inflection point of the temperature curve is as close as possible to the middle of the period. Some experience with atmospheric temperature profiles would help in selecting a good starting point for period value; otherwise, trial and error methods may be required to get a good curve fitting result.

Since all of the Fourier series coefficients are quite small, the starting value of coefficient C in the Fourier series expression of temperature profile should be chosen to be the atmospheric temperature at ground (ie. z=0). In the first trial of the curve fitting

process, the minimal number of harmonic terms in the temperature expression should be used to minimize searching time. Then the number of harmonic terms can be increased in the following trials to increase the accuracy of curve fitting.

Finally, the number of points in the given TC must be large enough in order to get a decent curve fitting result and keeping in mind that the large the number, the better the curve fit.

APPENDIX

Computer Program Listings

```
/*_____________________________________________________________________

                RAY TRACING IN ATMOSPHERIC GRAVITY WAVE PROGRAM
                                    by
                              Hung D. Pham
                          University of Manitoba 1989
                                (Microsoft C)
_____________________________________________________________________*/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#define G 9.81
#define BETA 3.48e-03
#define PNULL 1.0137931e05
#define ESP 0.000226
#define RE 6.37e06
static int NumLayer, PointCount, UpperIndex, LowerIndex, NumberOfRay,
    ObserverIndex, NumPoint[25], PrintInterval, num_TC,
    IndexH, IndexAH;
static float LayerElev[50], LayerTemp[50], density[50], RadianAngle[25],
    angle[25], ApparentElevation[8][25], ObjectElevation[8][25],
    ray_z[500], ray_x[500], X_RayPlot[100][25], Z_RayPlot[100][25],
    x, z, PhiZero, TanPhiZero, SinTheta, UpperLayer, LowerLayer,
    ObserverElevation, KmStep, MaxElevation, MaxHorizon,
    MeterStep, VertexElev, Z_RayTestPoint, X_RayTestPoint, TcDist[8],
    n1, n2, n3, Amplitude[50],GravityWave[50][60], WavyZp[50], RayArcX,
    EffectiveCurvature, AmpAtH, WaveLength, Phase;
static int gridRay, gridTc;
static float xorginRay, xmaxRay, yorginRay, ymaxRay,
    xorginTc, xmaxTc, yorginTc, ymaxTc;
static double rad;
static char title[80];

main()
{
 int herc, k, i, j, interval, ti_ans, ray_ans, wait,
    selection, RaySelector, TcSelector;
 float Density, HalfStep, Z_HalfStep;
 float FindRayElevation(),FindSinTheta(),FindEffectiveCurvature(),
     FindWavyDensity(),findEffectiveCurvature();
 void DrawAxes(), RayMenu(), TcMenu(), WriteTitle(),
    UpperLayerIntersect(), LowerLayerIntersect(),
    TC_curve(), ReadData(),
    PrintRayFile(), PrintTioutFile(), SearchLayer(),
    PlotWave(), WavyElevationAtTestPoint();
 FILE *out;

 printf("RAY TRACING WITH HALO\n");
```

```
 printf("HAS HERC. GRAPHICS MODE BEEN SET ? 0=NO\n");
 scanf("%d",&herc);
 if(herc == 0) goto end;

/* ____________________________________________________________________

   READ INPUT DATA AND CREATE A NEW LAYER AT OBSERVER ELEVATION
 ____________________________________________________________________*/

 ReadData();

/* ____________________________________________________________________

              DIVIDE THE ATMOSPHERE INTO 3-SECTION MODEL
 ____________________________________________________________________*/

NewPhase:
 SearchLayer();

 printf("DO YOU WISH TO PLOT WAVY ATMOSPHERE ? 0=NO\n");
 scanf("%d",&herc);
 if(herc != 0)
  PlotWave();


/*____________________________________________________________________

                START OF RAY PATHS CALCULATION
 ____________________________________________________________________*/

 for(k=0;k<NumberOfRay;k++)  /* k=ray number */
 {
   printf("RAY NO. %d PHIZERO=%f ARCMIN\n",k,angle[k]);
   PhiZero=RadianAngle[k];
   ray_x[0]=0.0;
   ray_z[0]=ObserverElevation;
   if(PhiZero >0.0)
   {
     LowerIndex=ObserverIndex;
     UpperIndex=ObserverIndex+1;
   }
   else
   {
     LowerIndex=ObserverIndex-1;
     UpperIndex=ObserverIndex;
   }
   RayArcX=MeterStep;
   HalfStep=MeterStep/2.0;
   PointCount=0;
   TanPhiZero=tan(PhiZero);
   x=0.0;
   z=ObserverElevation;
   SinTheta=FindSinTheta(ObserverIndex);
   Density=FindWavyDensity(x,z);
```

```
  EffectiveCurvature=FindEffectiveCurvature(Density);
  Z_RayTestPoint=FindRayElevation(RayArcX);

  X_RayTestPoint=MeterStep;
  WavyElevationAtTestPoint(X_RayTestPoint);

/* ____________________________________________________________________

  CALCULATE INDIVIDUAL RAY PATH CORRESPONDING TO ITS INITIAL RAY
                              ANGLE
 ____________________________________________________________________*/

  while( Z_RayTestPoint<=MaxElevation && Z_RayTestPoint>=0.0 &&
      X_RayTestPoint<=MaxHorizon)
  {
    if(Z_RayTestPoint >= UpperLayer)
    {
      UpperLayerIntersect(RayArcX);
      Density=FindWavyDensity(x,z);
      EffectiveCurvature=FindEffectiveCurvature(Density);
      Z_RayTestPoint=FindRayElevation(RayArcX);
    }
    else if(Z_RayTestPoint <= LowerLayer)
    {
      LowerLayerIntersect(RayArcX);
      Density=FindWavyDensity(x,z);
      EffectiveCurvature=FindEffectiveCurvature(Density);
      Z_RayTestPoint=FindRayElevation(RayArcX);
    }
    else
    {
      if(RayArcX == MeterStep)
      {
        Z_HalfStep=FindRayElevation(HalfStep);
        if( Z_HalfStep >= UpperLayer)
        {
          UpperLayerIntersect(HalfStep);
          Density=FindWavyDensity(x,z);
          EffectiveCurvature=FindEffectiveCurvature(Density);
          Z_RayTestPoint=FindRayElevation(RayArcX);
        }
        else if( Z_HalfStep <= LowerLayer )
        {
          LowerLayerIntersect(HalfStep);
          Density=FindWavyDensity(x,z);
          EffectiveCurvature=FindEffectiveCurvature(Density);
          Z_RayTestPoint=FindRayElevation(RayArcX);
        }
        else
        {
          PointCount=PointCount+1;
            ray_x[PointCount]=X_RayTestPoint;
            ray_z[PointCount]=Z_RayTestPoint;
          z=ray_z[PointCount];
```

```
                x=ray_x[PointCount];
                RayArcX=MeterStep;
              TanPhiZero=TanPhiZero-EffectiveCurvature*RayArcX;
              Density=FindWavyDensity(x,z);
              EffectiveCurvature=FindEffectiveCurvature(Density);
              Z_RayTestPoint=FindRayElevation(RayArcX);
            }
          }
          else
          {
            PointCount=PointCount+1;
              ray_x[PointCount]=X_RayTestPoint;
              ray_z[PointCount]=Z_RayTestPoint;
            z=ray_z[PointCount];
              x=ray_x[PointCount];
              RayArcX=MeterStep;
            TanPhiZero=TanPhiZero-EffectiveCurvature*RayArcX;
            Density=FindWavyDensity(x,z);
            EffectiveCurvature=FindEffectiveCurvature(Density);
            Z_RayTestPoint=FindRayElevation(RayArcX);
          }
          }

        X_RayTestPoint=ray_x[PointCount]+MeterStep;  /* test points(x,z) */
        WavyElevationAtTestPoint(X_RayTestPoint);

    }     /*---- end while loop----*/

    NumPoint[k]=PointCount/PrintInterval;
    for(i=0;i<=NumPoint[k];i++)
    {
      interval=i*PrintInterval;
      X_RayPlot[i][k]=ray_x[interval];
      Z_RayPlot[i][k]=ray_z[interval];
    }
  }       /*---- end of k loop ----*/


/* ____________________________________________________________________

    WRITE RAY DATA TO DISK IF REQUIRED
  ____________________________________________________________________*/

  printf("DO YOU WISH TO WRITE RAY DATA TO TIOUT ? 0=NO\n");
  scanf("%d",&ti_ans);
  if(ti_ans != 0)
   PrintTioutFile();

  printf("DO YOU WISH TO WRITE DATA TO RAYFILE ? 0=NO\n");
  scanf("%d",&ray_ans);
  if(ray_ans !=0)
   PrintRayFile();

/*____________________________________________________________________
```

```
                    PLOT RAY PATHS
_____________________________________________________________*/

 RaySelector=0;     /* RaySelector=0 ==> new ray plot else old one */
ReturnRay:
 selection=PlotRayTracing(RaySelector);
 if(selection == 0)
 {
   closegraphics();
   goto decision;
 }
 else if(selection == 1)
 {
   closegraphics();
   goto NewTc;
 }
 else if(selection == 2)
 {
   closegraphics();
   TcSelector=1;
   goto PreviousTc;
 }

/* _____________________________________________________________

        CALCULATE AND PLOT TRANSFER CHARACTERISTIC CURVES
_____________________________________________________________*/

NewTc:
 TC_curve();
 TcSelector=0;          /*  TcSelector = 0 ==> new TC plot */

PreviousTc:
 selection=PlotTcCurves(TcSelector);
 if(selection == 0)
 {
   closegraphics();
   goto decision;
 }
 if(selection == 1)
 {
   closegraphics();
   goto NewTc;
 }
 if(selection == 2)
 {
   closegraphics();
   RaySelector=1;         /* return to most recent ray plot */
   goto ReturnRay;
 }
 if(selection == 3)
 {
   closegraphics();
```

```
    RaySelector=0;        /* return to new ray plot */
    goto ReturnRay;
  }

decision:
  printf("DO YOU WISH TO TRACE WITH A NEW <GRAVITY WAVE PHASE> ? 0=NO\n");
  scanf("%d",&herc);
  if(herc != 0)
  {
    printf("PLEASE ENTER NEW <GRAVITY WAVE PHASE> IN DEGREE...\n");
    scanf("%f",Phase);
    Phase=Phase*3.1416/180.0;
    goto NewPhase;
  }

end: ; }

/*=================== END OF MAIN PROGRAM ===================*/


void WavyElevationAtTestPoint(coordinate)
float coordinate;
{
  UpperLayer= Amplitude[UpperIndex]*cos((6.28/WaveLength)*coordinate+Phase)+
          LayerElev[UpperIndex];
  LowerLayer= Amplitude[LowerIndex]*cos((6.28/WaveLength)*coordinate+Phase)+
          LayerElev[LowerIndex];
}

/*_____________________________________________________________________*/


void UpperLayerIntersect(UpperLimit)
float UpperLimit;
{
  float Xintersect,FindIntersect();

  Xintersect=FindIntersect(UpperIndex,UpperLimit);
  z=FindRayElevation(Xintersect);
  x=x+Xintersect;
  RayArcX=RayArcX-Xintersect;
  TanPhiZero=TanPhiZero-EffectiveCurvature*Xintersect;
  SinTheta=FindSinTheta(UpperIndex);
  UpperIndex=UpperIndex+1;
  LowerIndex=LowerIndex+1;
}

/*_____________________________________________________________________*/


void LowerLayerIntersect(UpperLimit)
float UpperLimit;
{
```

```
  float Xintersect,FindIntersect();

  Xintersect=FindIntersect(LowerIndex,UpperLimit);
  z=FindRayElevation(Xintersect);
  x=x+Xintersect;
  RayArcX=RayArcX-Xintersect;
  TanPhiZero=TanPhiZero-EffectiveCurvature*Xintersect;
  SinTheta=FindSinTheta(LowerIndex);
  UpperIndex=UpperIndex-1;
  LowerIndex=LowerIndex-1;
}

/*_____________________________________________________________________*/


float FindIntersect(index,UpperLimit)
int index;
float UpperLimit;
{
  double Ea,Eb,FunctValue();
   float K,Xupper,Xlower,Xa,Xb,inter;

  K=0.618;
  Xupper=UpperLimit;
  Xlower=0.0;
  Xa=Xupper-K*(Xupper-Xlower);
  Xb=Xlower+K*(Xupper-Xlower);
  Ea=FunctValue(index,Xa);
  Eb=FunctValue(index,Xb);
  while((Xupper-Xlower) > 1.0)
  {
   if(fabs(Ea) <= fabs(Eb))
    {
      Xupper=Xb;
      Xb=Xa;
      Xa=Xupper-K*(Xupper-Xlower);
      Eb=Ea;
      Ea=FunctValue(index,Xa);
    }
   else
    {
      Xlower=Xa;
      Xa=Xb;
      Xb=Xlower+K*(Xupper-Xlower);
      Ea=Eb;
      Eb=FunctValue(index,Xb);
    }
  }
  inter=Xlower+(Xupper-Xlower)/2.0;
  return inter;
}

/*_____________________________________________________________________*/
```

```
double FunctValue(index,Xvalue)
int index;
float Xvalue;
{
  double evaluation;

  evaluation=Amplitude[index]*cos((6.28/WaveLength)*(Xvalue+x)+Phase)+
        LayerElev[index]+0.5*Xvalue*Xvalue*EffectiveCurvature-
        Xvalue*TanPhiZero-z;
  return evaluation;
}

/*_____________________________________________________________________*/


float FindRayElevation(Xarc)
float Xarc;
{
  float Zray;

  Zray=-Xarc*Xarc*EffectiveCurvature+TanPhiZero*Xarc+z;
  return Zray;
}

/*_____________________________________________________________________*/


float FindWavyDensity(xcoord,zcoord)
float xcoord,zcoord;
{
  int i;
  float tint,power,alpha,dtint,PointDensity,slope;

  WavyZp[0]=0.0;
  for(i=1;i<NumLayer;i++) /*--- Wavy Elevation at index point ---*/
   WavyZp[i]= Amplitude[i]*cos((6.28/WaveLength)*xcoord+Phase)+LayerElev[i];

  tint=0.0;
  i=0;
  while(WavyZp[i] <= zcoord)
  {
  power=-G*BETA*tint;
  density[i]=(BETA*PNULL/(LayerTemp[i]+273))*exp(power);
  if(i >=(NumLayer-1)) goto skip;
  alpha=(LayerTemp[i+1]-LayerTemp[i])/(WavyZp[i+1]-WavyZp[i]);
  if(alpha == 0.0)
   dtint=(WavyZp[i+1]-WavyZp[i])/(LayerTemp[i]+273.0);
  else
   dtint=log(1.0+alpha*(WavyZp[i+1]-WavyZp[i])/(LayerTemp[i]+273.0))
       /alpha;
  tint=tint+dtint;
  i=i+1;
  }
```

```
   power=-G*BETA*tint;
   density[i]=(BETA*PNULL/(LayerTemp[i]+273))*exp(power);
skip:
   slope=(density[i]-density[i-1])/(WavyZp[i]-WavyZp[i-1]);
   PointDensity=(zcoord-WavyZp[i-1])*slope+density[i-1];
   return PointDensity;
}

/*_____________________________________________________________________*/


float FindEffectiveCurvature(dens)
float dens;
{
  float rise,run,dtdz,temp,RayCurve,EarthCurve,EffectCurve;

  rise=LayerTemp[UpperIndex]-LayerTemp[LowerIndex];
  run=(WavyZp[UpperIndex]-WavyZp[LowerIndex]);
  dtdz=rise/run;
  temp=(z-WavyZp[LowerIndex])*rise/run+
     LayerTemp[LowerIndex]+273.0;
  RayCurve=(dtdz+G*BETA)*ESP*dens*SinTheta/(temp*(1.0+ESP*dens));
  EarthCurve=1.0/RE;
  EffectCurve=RayCurve-EarthCurve;
  return EffectCurve;
}

/*_____________________________________________________________________*/

float FindSinTheta(IntersectIndex)
int IntersectIndex;
{
  float dzdx,dummy;

  dzdx=-(6.28/WaveLength)*Amplitude[IntersectIndex]*
      sin((6.28/WaveLength)*x+Phase);
  dummy=1.0-0.5*(TanPhiZero-dzdx)*(TanPhiZero-dzdx);
  return dummy;
}

/*_____________________________________________________________________*/


void SearchLayer()
{
  int i;
  float slope[40],MaxSlope,SlopeThreshold,AverageTemp,DTDZ,DeltaH;
  double WWb1,WWb2,WWb3,WW,kk,nn1,nn2,nn3,dummy,dummy1;

  MaxSlope=-20.0;
  for(i=1;i<NumLayer;i++)
  {
   slope[i]=(LayerTemp[i]-LayerTemp[i-1])/
         (LayerElev[i]-LayerElev[i-1]);
```

```
 if(slope[i] > MaxSlope)
   MaxSlope=slope[i];
}
SlopeThreshold=MaxSlope/2.0;
i=1;
while(slope[i] <= SlopeThreshold)
{
  i=i+1;
}
IndexH=i-1;
while(slope[i] > SlopeThreshold)
{
  i=i+1;
}
IndexAH=i-1;
AverageTemp=((LayerTemp[0]+273)+(LayerTemp[IndexH]+273))/2.0;
DTDZ=((LayerTemp[IndexH]+273)-(LayerTemp[0]+273))/
    (LayerElev[IndexH]-LayerElev[0]);
WWb1=(9.81/AverageTemp)*(DTDZ+9.81*BETA);
AverageTemp=((LayerTemp[IndexH]+273)+(LayerTemp[IndexAH]+273))/
          2.0;
DTDZ=((LayerTemp[IndexAH]+273)-(LayerTemp[IndexH]+273))/
    ((LayerElev[IndexAH]+273)-(LayerElev[IndexH]+273));
WWb2=(9.81/AverageTemp)*(DTDZ+9.81*BETA);
AverageTemp=((LayerTemp[NumLayer-1]+273)+(LayerTemp[IndexAH]+273))
         /2.0;
DTDZ=((LayerTemp[NumLayer-1]+273)-(LayerTemp[IndexAH]+273))/
    ((LayerElev[NumLayer-1]+273)-(LayerElev[IndexAH]+273));
WWb3=(9.81/AverageTemp)*(DTDZ+9.81*BETA);
WW=WWb2/2.0;
kk=(6.28/WaveLength)*(6.28/WaveLength);
nn1=kk*((WWb1/WW)*(WWb1/WW)-1);
nn2=kk*((WWb2/WW)*(WWb2/WW)-1);
nn3=kk*((WWb3/WW)*(WWb3/WW)-1);
n1=sqrt(fabs(nn1));
n2=sqrt(fabs(nn2));
n3=sqrt(fabs(nn3));

dummy=sinh(n1*LayerElev[IndexH]);
for(i=0;i<=IndexH;i++)
{
  Amplitude[i]=AmpAtH*sinh(n1*LayerElev[i])/dummy;
}
dummy=n1/(n2*tanh(n1*LayerElev[IndexH]));
for(i=IndexH+1;i<=IndexAH;i++)
{
  Amplitude[i]=AmpAtH*(cos(n2*(LayerElev[i]-LayerElev[IndexH]))+
           dummy*sin(n2*(LayerElev[i]-LayerElev[IndexH])));
}
DeltaH=LayerElev[IndexAH]-LayerElev[IndexH];
dummy1=AmpAtH*(cos(n2*DeltaH)+dummy*sin(n2*DeltaH));
for(i=IndexAH+1;i<NumLayer;i++)
{
  Amplitude[i]=dummy1*exp(-n3*(LayerElev[i]-LayerElev[IndexAH]));
```

```
  }
}

*/_____*/


void PlotWave()
{
  int i,j,PlotPoints,result;
   static int pattr[]={-1,-1,0,0,0,0,0,0,-1,0,0,-1,0,-1,-1,-1,0};
  float Xarray[60],Yarray[60],Ymax,Ymin,Xmarker,Ymarker,dx,dy;
  static char menu[1],XaxisLabel[]="DISTANCE",YaxisLabel[]="ELEVATION",
       filename[]="HALOEPSN.PRN";
  void WriteTitle(),DrawAxes(),RayMenu();

  for(i=1;i<NumLayer;i++)
  {
   for(j=0;j<50;j++)
     GravityWave[i][j]=Amplitude[i]*cos((6.28/WaveLength)*j*600.0+Phase)+
                LayerElev[i];
  }

  Ymax=LayerElev[NumLayer-1]+20.0;
  Ymin=0.0;
WaveRescale:
  WriteTitle(title,XaxisLabel,YaxisLabel);
  DrawAxes(0.0,Ymin,40.0,Ymax,0,0);

  for(i=0;i<NumLayer;i++)
  {
    Xarray[i]=Amplitude[i];
    Yarray[i]=LayerElev[i];
  }
  movabs(&Xarray[0],&Yarray[0]);
  polycabs(Xarray,Yarray,&NumLayer);

  Xmarker=0.0;
  Ymarker=LayerElev[IndexH];
  dx=10.0;
 dy=0.0;
  movabs(&Xmarker,&Ymarker);
  lnrel(&dx,&dy);
  Ymarker=LayerElev[IndexAH];
  movabs(&Xmarker,&Ymarker);
  lnrel(&dx,&dy);

  Xmarker=10.0;
  Ymarker=0.0;
  dx=0.0;
  dy=Ymax;
  movabs(&Xmarker,&Ymarker);
  lnrel(&dx,&dy);
  for(i=1;i<NumLayer;i++)
  {
```

```
    for(j=0;j<50;j++)
    {
      Xarray[j]=(float)j*0.6+10.0;
      Yarray[j]=GravityWave[i][j];
    }
    movabs(&Xarray[0],&Yarray[0]);
    PlotPoints=50;
    polycabs(Xarray,Yarray,&PlotPoints);
  }
  RayMenu();
NextWaveMenu:
  scanf("%s",menu);
  if(result=strcmpi(menu,"s") == 0)
  {
     closegraphics();
     printf("ENTER NEW Y-AXIS LOWER LIMIT (in m)\n");
     scanf("%f",&Ymin);
     printf("ENTER NEW Y-AXIS UPPER LIMIT (in m)\n");
     scanf("%f",&Ymax);
     goto WaveRescale;
  }
  else if(result=strcmpi(menu,"p") == 0)
  {
     setprn(filename);
     setpattr(pattr);
     gprint();
     goto NextWaveMenu;
  }
  else
     closegraphics();
}

/*_____________________________________________________________________*/


void ReadData()
/*-------------------------------------------------------------------------------------------------
  - READ INPUT DATA FROM SPECIFIED FILE
  - A NEW LAYER IS CREATED IF OBSERVER ELEVATION IS NOT AT THE GIVEN
    LAYERS ELEVATION
 -------------------------------------------------------------------------------------------------*/
{
  float dum_ele[50],dum_temp[50],rise,run;
  char indata[10];
  int i,j;
  FILE *in;

  printf("NAME OF INPUT DATA FILE:\n");
  scanf("%s",indata);
  in=fopen(indata,"r");
   fscanf(in,"%f\n",&ObserverElevation);
                  fscanf(in,"%f\n",&KmStep);
                  MeterStep=1000.0*KmStep;
   fscanf(in,"%d\n",&PrintInterval);
```

```
  fscanf(in,"%f\n",&MaxHorizon);
  fscanf(in,"%f\n",&MaxElevation);
  fscanf(in,"%d\n",&NumberOfRay);
  for(i=0;i<NumberOfRay;i++)
   fscanf(in,"%f\n",&angle[i]);
  for(i=0;i<NumberOfRay;i++)          /* convert angle to rad */
   RadianAngle[i]=angle[i]/3437.747;
  fgets(title,80,in);
  printf("%s\n",title);
  fscanf(in,"%d\n",&NumLayer);
  for(i=0;i<NumLayer;i++)
   fscanf(in,"%f %f\n",&dum_ele[i],&dum_temp[i]);
  fscanf(in,"%f\n",&AmpAtH);
  fscanf(in,"%f\n",&Phase);
   Phase=Phase*3.1416/180.0; /* Convert to Radian */
  fscanf(in,"%f\n",&WaveLength);

/*======== create a new layer at observer elevation if required =========*/

  for(i=0;i<NumLayer;i++)
  {
   if(ObserverElevation == dum_ele[i])
   {
     ObserverIndex=i;
     for(j=0;j < NumLayer;j++)
     {
       LayerElev[j]=dum_ele[j];
       LayerTemp[j]=dum_temp[j];
     }
   }
   else if(ObserverElevation<dum_ele[i])
   {
     ObserverIndex=i;
     NumLayer=NumLayer+1;
     for(j=0;j<NumLayer;j++)
     {
       if(j<ObserverIndex)
       {
         LayerElev[j]=dum_ele[j];
         LayerTemp[j]=dum_temp[j];
       }
       else if(j == ObserverIndex)
       {
         LayerElev[j]=ObserverElevation;
         rise=dum_ele[j]-dum_ele[j-1];
         run=dum_temp[j]-dum_temp[j-1];
         LayerTemp[j]=LayerTemp[j-1]+(ObserverElevation-
                 dum_ele[j-1])*run/rise;
       }
       else
       {
          LayerElev[j]=dum_ele[j-1];
          LayerTemp[j]=dum_temp[j-1];
       }
```

```
        }
        goto finish;
      }
    }       /*--------- end of i loop ---------*/
finish:
  if(MaxElevation > LayerElev[NumLayer-1])
    MaxElevation=LayerElev[NumLayer-1];
  fclose(in);
}

/*_____________________________________________________________________*/


void PrintRayFile()
{
  int k, i;
  FILE *ray;

  ray=fopen("rayfile","w");
   fprintf(ray,"%s\n",title);
   fprintf(ray,"%d\n",NumberOfRay);
   for(k=0;k<NumberOfRay;k++)
   {
    fprintf(ray,"%d\n",NumPoint[k]);
    for(i=0;i<=NumPoint[k];i++)
      fprintf(ray,"%10.2f   %10.4f   %10.2f\n",X_RayPlot[i][k],
          Z_RayPlot[i][k],angle[k]);
   }
  fclose(ray);
}

/*_____________________________________________________________________*/


void PrintTioutFile()
{
  int k, i;
  FILE *out;

  out=fopen("tiout","a");
   fprintf(out,"HORIZONTAL STEP SIZE = %f Km\n",KmStep);
   fprintf(out,"OBSERVER ELEVATION = %f m\n",ObserverElevation);
   for(k=0;k<=NumPoint[k];k++)
   {
    fprintf(out,"INITIAL RAY ANGLE = %f ARCMIN\n",angle[k]);
    for(i=0;i<NumberOfRay;i++)
     fprintf(out,"pts=%2d   x=%10.2f   z=%10.4f\n",i,X_RayPlot[i][k],
         Z_RayPlot[i][k]);
   }
  fclose(out);
 }

/*_____________________________________________________________________*/
```

```
void TC_curve()
{
  int k,object_interval,tc_ans;
  float distance,TC_distance,rise,slope,ext_dist;
  FILE *tc;

  num_TC=0;
  tc=fopen("tiout.tc","w");
    fprintf(tc,"*** TRANSFER CHARACTERSITIC ***\n\n");
                        fprintf(tc,"%s\n",title);
    fprintf(tc,"OBSERVER ELEVATION = %f\n",ObserverElevation);
    fprintf(tc,"HORIZONTAL STEP SIZE = %f Km\n",KmStep);
next:  num_TC=num_TC+1;
    printf("DISTANCE FOR WHICH TC IS REQUIRED (in Km) ?\n");
    scanf("%f",&distance);
    object_interval=distance/(KmStep*PrintInterval)+0.5;
    TC_distance=object_interval*(PrintInterval*KmStep);
    TcDist[num_TC]=TC_distance;
    printf("TC WILL BE FOUND FOR DISTANCE OF %f Km\n",TC_distance);
    fprintf(tc,"TRANSFER CHARACTERISTIC FOUND AT %f Km\n\n",TC_distance);
  for(k=0;k<NumberOfRay;k++)
  {
    ApparentElevation[num_TC][k]=ObserverElevation+1000.0*TC_distance*
                        tan(RadianAngle[k]);
    if(NumPoint[k]<object_interval)
    {
     rise=Z_RayPlot[NumPoint[k]][k]-Z_RayPlot[NumPoint[k]-1][k];
     if(rise < 0.0)
      ObjectElevation[num_TC][k]=0.0;
     else
     {
      slope=rise/(MeterStep*PrintInterval);
      ext_dist=(1000.0*TC_distance-X_RayPlot[NumPoint[k]][k])*slope;
      ObjectElevation[num_TC][k]=Z_RayPlot[NumPoint[k]][k]+ext_dist;
     }
    }
    else
      ObjectElevation[num_TC][k]=Z_RayPlot[object_interval][k];
    fprintf(tc,"%10.2f    %10.2f\n",ObjectElevation[num_TC][k],
          ApparentElevation[num_TC][k]);
  }
  printf("IS ANOTHER TC REQUIRED ? (0=NO)\n");
  scanf("%d",&tc_ans);
  if(tc_ans != 0)
   goto next;
  fclose(tc);
}

/*_____________________________________________________________________*/


void WriteTitle(title,XaxisLabel,YaxisLabel)
char title[], XaxisLabel[], YaxisLabel[];
```

```
{
  int mode, path;
  float x1, x2, y1, y2, tx, ty, height, width, aspect, iheight, offset;
  static char device[]="HALOHERC.DEV", fontname[]="HALO104.FNT";

  setdev(device);
  mode=0;
  closegraphics();
  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=192.0;
  movtcurabs(&tx,&ty);
  btext(title);

  setfont(fontname);
  height=15.0;
  aspect=1.3;
  path=1;
  setstext(&height,&aspect,&path);
  inqstsize(YaxisLabel,&iheight,&width,&offset);
  ty=((y2-width)/2.0)+45.0;
  tx=30.0;
  movtcurabs(&tx,&ty);
  stext(YaxisLabel);

  height=10.0;
  path=0;
  aspect=3.0;
  setstext(&height,&aspect,&path);
  inqstsize(XaxisLabel,&iheight,&width,&offset);  /* write x axis lable */
  tx=(x2-width)/2.0;
  ty=25.0;
  movtcurabs(&tx,&ty);
  stext(XaxisLabel);

  deltcur();
}

/*_____________________________________________________________________*/


void DrawAxes(xorgin,yorgin,xmax,ymax,grid,type)
float xorgin, yorgin, xmax, ymax;
int grid, type;
{
  int i, border, black, index;
  float x1, x2, y1, y2, newx1, newy1, newx2, newy2, dx, dy,
        Xunit, Yunit, Xtext, Ytext, Xmark, Ymark,
        Xorgin, Yorgin, Xmax, Ymax, value;
```

```
static char label[5], unit1[]="(Km)", unit2[]="(m)";


x1=0.05; y1=0.05; x2=0.99; y2=0.80, border=1; black=-1;
setviewport(&x1,&y1,&x2,&y2,&border,&black);

Xorgin=xorgin;
Yorgin=yorgin;
Xmax=xmax;
Ymax=ymax;
Xunit=(Xmax-Xorgin)/10.0;
Yunit=(Ymax-Yorgin)/10.0;

x1=0.0; y1=0.0; x2=215; y2=108;
setworld(&x1,&y1,&x2,&y2);

dx=0.0;
if(grid == 0)  /* grid=0 ==> no grid */
   dy=-1.5;
else
{
   dy=108;
   index=3;
   setlnstyle(&index);
}
Xmark=15.0;
Ymark=8.0;
Xtext=-10.0;
Ytext=1.0;
for(i=0;i<=10;i++)
{
  movabs(&Xmark,&Ymark);
  lnrel(&dx,&dy);
  Xtext=Xtext+20.0;
  movtcurabs(&Xtext,&Ytext);   /* along X axis */
  value=Xorgin+(float)i*Xunit;
  gcvt(value,4,label);

  if(i == 10)
  {
       Xtext=Xtext-6.0;
       movtcurabs(&Xtext,&Ytext);
       if(type == 0)  /* type=0 ray plot else tc plot */
         btext(unit1);
       else
         btext(unit2);
   }
   else
    btext(label);
   Xmark=Xmark+20.0;

  }

dy=0.0;
```

```
  if(grid == 0)
    dx=-1.5;
  else
  {
    dx=215;
    index=3;
    setlnstyle(&index);
  }
  Xmark=15.0;
  Ymark=8.0;
  Xtext=1.0;
  Ytext=-3.8;
  for(i=0;i<=10;i++)
  {
    movabs(&Xmark,&Ymark);
    lnrel(&dx,&dy);
    Ytext=Ytext+10.0;
    movtcurabs(&Xtext,&Ytext);
    value=Yorgin+(float)i*Yunit;
    gcvt(value,4,label);

    if(i == 10)
    {
        Ytext=Ytext-3.0;
        movtcurabs(&Xtext,&Ytext);
        btext(unit2);
    }
    else
        btext(label);
    Ymark=Ymark+10.0;
  }
  deltcur();
  index=1;
  setlnstyle(&index);

  x1=0.117; y1=0.05; x2=0.99; y2=0.743, border=1; black=-1;
  setviewport(&x1,&y1,&x2,&y2,&border,&black);

  setworld(&Xorgin,&Yorgin,&Xmax,&Ymax);

}

/*_____________________________________________________________________*/


void RayMenu()
{
  int i, border, black;
  float x1, x2, y1, y2, tx, ty;
  static char list1[]="Q=QUIT    P=PRINT SCREEN   G=GRID ON/OFF";
  static char list2[]="N=NEW TC  T=PREVIOUS TC    S=SCALE";

  x1=0.05; y1=0.90; x2=0.99; y2=0.99; 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=23.0; tx=5.0;
  movtcurabs(&tx,&ty);
  btext(list1);
  ty=6.0;
  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 TcMenu()
{
  int i, border, black;
  float x1, x2, y1, y2, tx, ty;
  static char list1[]="P=PRINT SCREEN    G=GRID ON/OFF    Q=QUIT   ";
  static char list2[]="R=LAST RAY PLOT   O=ORG. RAY PLOT   N=NEW TC   S=SCALE";

  x1=0.05; y1=0.90; x2=0.99; y2=0.99; 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=23.0; tx=5.0;
  movtcurabs(&tx,&ty);
  btext(list1);
  ty=6.0;
  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);
}

/*_____________________________________________________________________*/


PlotRayTracing(indicator)
int indicator;
{
  int type, result, i, k, PlotPoints;
   static int pattr[]={-1,-1,0,0,0,0,0,0,0,-1,0,0,-1,0,-1,-1,-1,0};
```

```
  float  Xarray[50], Yarray[50], UpperRoundOff();
  static char menu[1], filename[]="HALOEPSN.PRN",
          XaxisLabel[]="OBJECT DISTANCE", YaxisLabel[]="ELEVATION";
       void WriteTitle(), DrawAxes(), RayMenu();

  type=0;       /* type=0 : ray plot */
  if(indicator != 0)
    goto RayRescale;    /* indicator=0 ==> old ray plot */
  gridRay=0;
  xorginRay=0.0;
  yorginRay=0.0;
  xmaxRay=MaxHorizon/1000.0;
  ymaxRay=0.0;
  for(k=0;k<NumberOfRay;k++)
  {
    for(i=0;i<=NumPoint[k];i++)
    {
      if(Z_RayPlot[i][k] > ymaxRay)
        ymaxRay=Z_RayPlot[i][k];
    }
  }
  ymaxRay=UpperRoundOff(ymaxRay);

RayRescale:
  WriteTitle(title,XaxisLabel,YaxisLabel);

  DrawAxes(xorginRay,yorginRay,xmaxRay,ymaxRay,gridRay,type);

  for(k=0;k<NumberOfRay;k++)
  {
    for(i=0;i<=NumPoint[k];i++)
    {
      Xarray[i]=X_RayPlot[i][k]/1000.0;
        Yarray[i]=Z_RayPlot[i][k];
    }
    movabs(&Xarray[0],&Yarray[0]);
    PlotPoints=NumPoint[k]+1;
    polycabs(Xarray,Yarray,&PlotPoints);
  }

  RayMenu();
RayNextMenu:
  scanf("%s",menu);
  if(result=strcmpi(menu,"g") == 0)
  {
    closegraphics();
    if(gridRay == 0)
      gridRay=1;
    else
        gridRay=0;
    goto RayRescale;
  }
  else if(result=strcmpi(menu,"s") == 0)
  {
```

```
    closegraphics();
    printf("ENTER NEW X-AXIS LOWER LIMIT (in Km)\n");
    scanf("%f",&xorginRay);
    printf("ENTER NEW X-AXIS UPPER LIMIT (in Km)\n");
    scanf("%f",&xmaxRay);
    printf("ENTER NEW Y-AXIS LOWER LIMIT (in m)\n");
    scanf("%f",&yorginRay);
    printf("ENTER NEW Y-AXIS UPPER LIMIT (in m)\n");
    scanf("%f",&ymaxRay);
        goto RayRescale;
  }
  if(result=strcmpi(menu,"p") == 0)
  {
    setprn(filename);
    setpattr(pattr);
    gprint();
    goto RayNextMenu;
  }
  else if(result=strcmpi(menu,"q") == 0)
  {
    return(0);
  }
  else if(result=strcmpi(menu,"n") == 0)
  {
    return(1);
  }
  else if(result=strcmpi(menu,"t") == 0)
  {
    return(2);
  }
  else
    return(1);
}

/*_____________________________________________________________________*/


float UpperRoundOff(InputNumber)
float InputNumber;
{
  float OutputNumber, count, remainder;

  if(remainder=fmod(InputNumber,10.0) == 0.0)
    OutputNumber= InputNumber;
  else
  {
    count=0.0;
    if(InputNumber > 0.0)
    {
      while(InputNumber >= 10.0)
        {
          InputNumber=InputNumber-10.0;
          count=count+1.0;
        }
```

```
      if(InputNumber >5.0)
          OutputNumber=(count+1.0)*10.0;
        else
          OutputNumber=count*10.0+5.0;
    }
    else
    {
      while(InputNumber <= -10)
        {
          InputNumber=InputNumber+10.0;
          count=count+1.0;
        }
        if(InputNumber < -5.0)
          OutputNumber=-10.0*(count+1.0);
        else
          OutputNumber=-10.0*count+5.0;
    }
  }
  return OutputNumber;
}

/*_____________________________________________________________________*/


float LowerRoundOff(InputNumber)
float InputNumber;
{
  float OutputNumber, count, remainder;

  if(remainder=fmod(InputNumber,10.0) == 0.0)
    OutputNumber= InputNumber;
  else
  {
    count=0.0;
    if(InputNumber > 0.0)
    {
      while(InputNumber >= 10.0)
        {
          InputNumber=InputNumber-10.0;
          count=count+1.0;
        }
        if(InputNumber < 5.0)
          OutputNumber=count*10.0;
        else
          OutputNumber=count*10.0+5.0;
    }
    else
    {
      while(InputNumber <= -10)
        {
          InputNumber=InputNumber+10.0;
          count=count+1.0;
        }
        if(InputNumber < -5.0)
```

```
            OutputNumber=-10.0*(count+1.0);
          else
            OutputNumber=-10.0*count+5.0;
      }
    }
  return OutputNumber;
}

/*_____________________________________________________________________*/


PlotTcCurves(TcSelector)
int TcSelector;
{
  int i, k, LineIndex, type, result, width;
   static int pattr[]={-1,-1,0,0,0,0,0,0,-1,0,0,-1,0,-1,-1,-1,0};
  float Xarray[50], Yarray[50],
  XlineStart, YlineStart, XlineEnd, XnoteStart, YnoteStart,
  Ymove, UpperRoundOff(), LowerRoundOff();
  static char menu[1], note[4], filename[]="HALOEPSN.PRN",
        XaxisLabel[]="Z(object)", YaxisLabel[]="Z(apparent)";
  void WriteTitle(), DrawAxes(), RayMenu();

  type=1;
 if(TcSelector != 0)
    goto TcRescale;
  gridTc=0;
  xorginTc=ObjectElevation[0][0];
  yorginTc=ApparentElevation[0][0];
  xmaxTc=xorginTc;
  ymaxTc=yorginTc;

  for(i=1;i<=num_TC;i++)
  {
    for(k=0;k<NumberOfRay;k++)
    {
     if(ObjectElevation[i][k] < xorginTc)
          xorginTc=ObjectElevation[i][k];
        if(ApparentElevation[num_TC][k] < yorginTc)
          yorginTc=ApparentElevation[i][k];
        if(ObjectElevation[i][k] > xmaxTc)
          xmaxTc=ObjectElevation[i][k];
        if(ApparentElevation[i][k] > ymaxTc)
          ymaxTc=ApparentElevation[i][k];
    }
  }
  xorginTc=LowerRoundOff(xorginTc);
  yorginTc=LowerRoundOff(yorginTc);
  xmaxTc=UpperRoundOff(xmaxTc);
  ymaxTc=UpperRoundOff(ymaxTc);

TcRescale:
  WriteTitle(title,XaxisLabel,YaxisLabel);
```

```
  DrawAxes(xorginTc,yorginTc,xmaxTc,ymaxTc,gridTc,type);
  XlineStart=xmaxTc-(xmaxTc-xorginTc)/5.0;
  XlineEnd=XlineStart+(xmaxTc-xorginTc)/10.0;
  XnoteStart=XlineEnd+(xmaxTc-xorginTc)/25.0;
  Ymove=(ymaxTc-yorginTc)/16.0;
  YlineStart=yorginTc+Ymove/2.0;
  YnoteStart=yorginTc+Ymove/4.0;

  LineIndex=1;
  for(i=1;i<=num_TC;i++)
  {
    setlnstyle(&LineIndex);
    for(k=0;k<NumberOfRay;k++)
    {
      Xarray[k]=ObjectElevation[i][k];
        Yarray[k]=ApparentElevation[i][k];
    }
    movabs(&Xarray[0],&Yarray[0]);
    polylnabs(Xarray,Yarray,&NumberOfRay);

    movabs(&XlineStart,&YlineStart);
    lnabs(&XlineEnd,&YlineStart);
    movtcurabs(&XnoteStart,&YnoteStart);
    gcvt(TcDist[i],4,note);
    btext(note);
    YnoteStart=YnoteStart+Ymove;
    YlineStart=YlineStart+Ymove;

    LineIndex=LineIndex+1;
    if(LineIndex > 3)
    {
        LineIndex=1;
        width=3;
        setlnwidth(&width);
    }
  }
  LineIndex=1;
  setlnstyle(LineIndex);
  width=1;
  setlnwidth(&width);

  TcMenu();
TcNextMenu:
  scanf("%s",menu);
  if(result=strcmpi(menu,"g") == 0)
  {
    closegraphics();
    if(gridTc == 0)
        gridTc=1;
    else
        gridTc=0;
    goto TcRescale;
  }
```

```
  else if(result=strcmpi(menu,"s") == 0)
  {
    closegraphics();
    printf("ENTER NEW X-AXIS LOWER LIMIT (in m)\n");
    scanf("%f",&xorginTc);
    printf("ENTER NEW X-AXIS UPPER LIMIT (in m)\n");
    scanf("%f",&xmaxTc);
    printf("ENTER NEW Y-AXIS LOWER LIMIT (in m)\n");
    scanf("%f",&yorginTc);
    printf("ENTER NEW Y-AXIS UPPER LIMIT (in m)\n");
    scanf("%f",&ymaxTc);
    goto TcRescale;
  }
  else if(result=strcmpi(menu,"p") == 0)
  {
    setprn(filename);
    setpattr(pattr);
    gprint();
    goto TcNextMenu;
  }
  else if(result=strcmpi(menu,"q") == 0)
  {
    return(0);
  }
  else if(result=strcmpi(menu,"n") == 0)
  {
    return(1);
  }
  else if(result=strcmpi(menu,"r") == 0)
  {
    return(2);
  }
  else if(result=strcmpi(menu,"o") == 0)
  {
    return(3);
  }
  else
    return (0);
}
```

Input Data to "Ray Tracing" Program (jo4)

```
 2.5                 : Observer Elevation (m)
 0.25                : Parabolic ray arc step size in x-direction (Km)
  1                  : Interval between data points of light ray to be printed (Km)
 20000.0             : Maximum limit of ray in x-direction (m)
 100.0               : Maximum limit of ray in z-direction (m)
 30                  : Number of rays to be traced (maximum 30)
0.                   : Initial ray angle (minute)
0.5                          "
1.                           "
1.5                          "
2.                           "
2.5                          "
3.                           "
3.5                          "
4.                           "
4.5                          "
5.                           "
5.5                          "
6.                           "
6.5                          "
7.                           "
7.5                          "
8.                           "
8.5                          "
9.                           "
9.5                          "
10.                          "
10.5                         "
11.                          "
11.5                         "
12.                          "
13.5                         "
14.                          "
14.5                         "
'PROFILE JO4  (SMOOTHED PROFILE OF JOSA83)'     : Title of the graph
  36                 : Number of data point in the temperature profile (maximum 50)
0.0    -2.39        : Elevation (m) --- Atmospheric temperature (C)
2.48   -2.0                  "
5.5    -1.83                 "
8.0    -1.62                 "
10.0   -1.43                 "
11.75  -1.20                 "
13.5   -0.92                 "
15.0   -0.63                 "
16.5   -0.30                 "
18.0    0.09                 "
19.0    0.37                 "
20.0    0.67                 "
21.0    0.99                 "
```

22.0	1.33	"
23.0	1.69	"
24.0	2.08	"
25.0	2.50	"
25.9	2.90	"
26.7	3.30	"
27.4	3.70	"
28.07	4.10	"
28.7	4.50	"
29.3	4.90	"
29.8	5.30	"
30.3	5.70	"
30.75	6.00	"
31.25	6.30	"
31.8	6.60	"
32.7	6.90	"
33.7	6.90	"
34.9	7.50	"
36.5	7.72	"
38.0	7.81	"
40.2	7.85	"
44.5	7.87	"
50.0	7.87	"

2.0 : Gravity wave amplitude W_H (at z=H) (m)

45.0 : Gravity wave phase (degree)

15000.0 : Gravity wave length λ (m)

```
//PHAM JOB ',,,L=4,T=3M','JM3'
// EXEC FORT7CLG,P=D
//FORT.SYSIN DD *
C
C       INVERSION OF MIRAGE DATA PROGRAM:
C         THIS PROGRAM WILL FIND A SET OF FOURIER SERIES COEFFICIENTS
C         REPRESENTING THE ATMOSPHERIC TEMPERATURE PROFILE
C         WHICH MINIMIZE THE SUM OF SQUARE ERRORS BETWEEN GIVEN
C         TRANSFER CHARACTERISTIC AND FOURIER SERIES GENERATED
C         TRANSFER CHARACTERISTIC USING  OPTIMIZATION METHOD OF
C          ROSEN BROCK METHOD OF ROTATING COORDINATES
C
C
C
     COMMON /IDATA/ ZONE,DU,IPRT,XMAX,RVMAX,NPHI,PHINIT(25),N,
    *ZP(50),DIST,PERIOD,TP(50),MM,DR(50),IFLAG
     COMMON /IOUT/ IN, IO, ID
      DIMENSION U(10,10), V(10,10), W(10,10), S(10), D(10)
      DIMENSION IS(10), IF(10), EXIT(2), X(10), XO(10),DZOBJ(25)
      INTEGER M,NFEMAX,KEXIT,NMX,IN,IO,NUM,HARMO
     REAL ALPHA,BETA,STEP
     IN=1
     IO=6
     NMX=10
     READ(IN,*)ID
     READ(IN,*)M
      READ(IN,*)(XO(I),I=1,M)
     READ(IN,*)NUM
      READ(IN,*)(DZOBJ(I),I=1,NUM)
C
     READ(3,*)ZONE
     READ(3,*)DU
     READ(3,104)IPRT
     READ(3,*)XMAX
     READ(3,*)RVMAX
     READ(3,104)NPHI
104 FORMAT(I4)
      READ(3,*)(PHINIT(I),I=1,NPHI)
     READ(3,104)N
      READ(3,*)(ZP(J),J=1,N)
     READ(3,*)DIST
     READ(3,*)PERIOD
C
     NFEMAX=100
     KEXIT=1
     ALPHA=2.00
     BETA=0.50
     STEP=0.1
     EXIT(1)=0.1
     EXIT(2)=0.1
     IFLAG=0
     CALL RSNBRK(XO,M,NMX,STEP,ALPHA,BETA,NFEMAX,EXIT,KEXIT,DZOBJ)
     WRITE(IO,100)
100   FORMAT(/,1X,'THE OPTIMUM FOURIER POLYNOMIAL COEFFICIENTS ARE:'
```

```
     * ,/)
      DO 8 I=1,M
8     WRITE(IO,120)I,XO(I)
120     FORMAT(20X,'X0(',I1,')=',F16.9,/)
      WRITE(IO,*)'TEMPERATURE PROFILE AND TC CURVE :'
      DO 22 J=1,N
22      WRITE(6,*)ZP(J),TP(J)
      IFLAG=1
      CALL FUNCT(F,XO,NFE,DZOBJ,FO,M)
      STOP
      END
C
C
      SUBROUTINE RSNBRK(XO,M,NMX,STEP,ALPHA,BETA,NFEMAX,EXIT,
     + KEXIT,DZOBJ)
C     ID  : DEBUG PARAMETER
C           0 NO PRINT-OUT WILL BE GIVEN,
C           1 LIMITED PRINT-OUT WILL BE GIVEN,
C           2 FULL PRINT-OUT WILL BE GIVEN.
C     M   : NUMBER OF VARIABLES.
C
C     ALPHA: EXPANSION FACTOR (3.0 IS RECOMENDED).
C     BETA : PENALTY FACTOR FOR FAILURE (0.5 IS RECOMENDED).
C     NFEMX: NUMBER OF MAXIMUM FUNCTION EVALUATION.
C     STEP : INITIAL STEP SIZE (0.1 IS RECOMENDED).
C     KEXIT: EXIT CRITERIA OPTION
C           1 TO EXIT WITH THE IMPROVEMENT OF THE FUNCTION VALUE.
C           2 TO EXIT ROOT MEAN SQUARE OF THE IMPROVEMENTS IN ALL
C             DIRECTIONS.
C     EXIT : EXIT CRITERIA VECTOR
C           1ST ELEMENT IS TO USE WITH K=1
C           2ND ELEMENT IS TO USE WITH K=2.
      COMMON /IOUT/IN,IO,ID
      DIMENSION U(10,10),V(10,10),W(10,10),S(10),D(10),IF(10),
     + IS(10),XO(10),X(10),EXIT(2),DZOBJ(15)
      INTEGER M,NMX,KEXIT,NFEMAX
      REAL ALPHA,BETA,STEP,FO
      IF (ID.GE.0) WRITE (IO,120) ID,M,ALPHA,BETA,NFEMAX,KEXIT,
     + STEP,(EXIT(I),I=1,2)
      IF (ID.GE.0) WRITE (IO,130) (XO(I),I=1,M)
      DO 20 J=1,M
      DO 10 I=1,M
      U(J,I)=0.0
10    S(I)=STEP
20    U(J,J)=1.0
      NFE=0
      FO=100000.0
      CALL FUNCT(F,XO,NFE,DZOBJ,FO,M)
      FO=F
30    IF (ID.LT.2) GO TO 50
      DO 40 I=1,M
40    IF (ID.GE.2) WRITE (IO,140) I,(U(I,J),J=1,M)
50    IF (ID.GE.1) THEN
      WRITE (IO,150) NFE,FO
```

```
      DO 52 I=1,M
52    WRITE (IO,151) XO(I)
      END IF
151   FORMAT(7X,F16.9)
      CALL RSRCH(FO,XO,X,U,S,D,IF,IS,M,NMX,ALPHA,BETA,NFE,DZOBJ)
      IF (NFE.GT.NFEMAX) ID=2
      IF (NFE.GT.(NFEMAX+20)) GO TO 90
      SUM=0.0
      DO 60 I=1,M
60    SUM=SUM+D(I)*D(I)
      ST=SQRT(SUM/M)
      IF (KEXIT.EQ.1) GO TO 70
      IF (ST.LT.EXIT(2)) GO TO 100
      GO TO 80
70    IF (ABS(FO-F).LT.EXIT(1)) GO TO 100
80    CALL ROTATE(U,V,W,D,M,NMX)
      IF (ID.GE.2) WRITE (IO,160) (D(J),J=1,M)
      IF (ID.GE.2) WRITE (IO,170) (S(J),J=1,M)
      GO TO 30
90    IF (ID.GE.0) WRITE (IO,180) FO,NFE
      GO TO 110
100   IF (ID.GE.0) WRITE (IO,190) FO,NFE
110   DO 112 I=1,M
112   WRITE (IO,200) XO(I)
      RETURN
120   FORMAT (1H1//1X,70('-')/2X,'ROSENBROCK METHOD OF ROTATING ','CO-OR
     +DINATES'/1X,70('-')///5X,'INPUT DATA:'/5X,11('-')//5X,'ID    =',I3/
     +5X,'M     =',I3/5X,'ALPHA=',F6.2/5X,'BETA =',F6.2/5X,'NFEMX=',I3/5X
     +,'KEXIT=',I3/5X,'STEP =',F5.2/5X,'EXIT =',F10.7/11X,F10.7/)
130   FORMAT (3F16.9,/,3F16.9,/4F16.9)
140   FORMAT (2X,'U(',I2,')=',10F10.6)
150   FORMAT (/2X,'NFE=',I4,2X,'FO=',F16.7)
160   FORMAT (2X,'D(J)=',10F10.5)
170   FORMAT (2X,'S(J)=',10F10.6)
180   FORMAT (//1X,'PROGRAM STOPPED AT F=',F16.7,/1X,'BECAUSE OF EXCESS
     +FUNCTION EVALUATIONS. NFE=',I3//)
190   FORMAT (//1X,'MINIMUM FUNCTION VALUE OF',F16.7/2X,'IS FOUND IN ',
     +I4,' FUNCTION EVALUATIONS AT '/)
200   FORMAT (2X,'X=',F16.9)
      END
C
C
      SUBROUTINE RSRCH(FO,XO,X,U,S,D,IF,IS,M,NMX,ALPHA,BETA,NFE,
     + DZOBJ)
      COMMON /IOUT/IN,IO,ID
      DIMENSION U(10,10),XO(10),X(10),S(10),D(10),IF(10),IS(10),
     + DZOBJ(15)
      INTEGER NFE,M,NMX
      REAL ALPHA,BETA,F,FO
      DO 10 J=1,M
      IS(J)=0
      IF(J)=0
10    D(J)=0.0
20    DO 60 J=1,M
```

```
      DO 30 I=1,M
30    X(I)=XO(I)+S(J)*U(J,I)
      CALL FUNCT(F,X,NFE,DZOBJ,FO,M)
      IF (ID.GE.2) WRITE (IO,80)J,NFE,F,(X(I),I=1,M)
      IF (F.LE.FO) GO TO 40
      IF (IS(J).EQ.1) IF(J)=1
      S(J)=-BETA*S(J)
      GO TO 60
40    IS(J)=1
      D(J)=D(J)+S(J)
      S(J)=ALPHA*S(J)
      FO=F
      DO 50 I=1,M
50    XO(I)=X(I)
60    CONTINUE
      DO 70 J=1,M
70    IF (IF(J)*IS(J).EQ.0) GO TO 20
      RETURN
80      FORMAT  (/2X,'J=',I2,2X,'NFE=',I3,2X,'F=',F13.7/2X,'X=',10F10.5)
      END
C
C
      SUBROUTINE ROTATE(U,V,W,D,M,NMX)
      DIMENSION U(10,10),W(10,10),V(10,10),D(10)
      INTEGER M,NMX
      DO 10 I=1,M
10    V(M,I)=D(M)*U(M,I)
      DO 30 J=2,M
      K=M-J+1
      K1=K+1
      DO 20 I=1,M
20    V(K,I)=D(K)*U(K,I)+V(K1,I)
30    CONTINUE
      SQ=0.0
      DO 40 I=1,M
      W(1,I)=V(1,I)
40    SQ=SQ+W(1,I)**2
      RSQ=1.0/SQRT(SQ)
      DO 50 I=1,M
50    U(1,I)=W(1,I)*RSQ
      DO 110 J=2,M
      SQ=0.0
      DO 60 I=1,M
60    W(J,I)=V(J,I)
      K=J-1
      DO 80 L=1,K
      SUM=0.0
      DO 70 I=1,M
70    SUM=SUM+V(J,I)*U(L,I)
      DO 80 I=1,M
80    W(J,I)=W(J,I)-SUM*U(L,I)
      DO 90 I=1,M
90    SQ=SQ+W(J,I)**2
      RSQ=1.0/SQRT(SQ)
```

```
      DO 100 I=1,M
  100 U(J,I)=W(J,I)*RSQ
  110 CONTINUE
      RETURN
      END

C
C
C
C
C       TC59    RAY TRACING BY PARABOLIC ARCS  -  INCL. TC
C
C               WRITTEN BY W. LEHN
C
C                  UPDATED  FEB 1986
C
C       SUBSCRIPT 1 REFERS TO DR AT Z = 0  ETC.
C       DR = DENSITY
C       ENTER INITIAL RAY ANGLES, PHINIT(I), IN ARCMINUTES
C       ENTER STEP SIZE  DU  IN KM
C       ENTER TEMP PROFILE  TP  IN CELSIUS
C       NEW ORIGIN IS TAKEN EACH TIME A NEW LAYER IS ENTERED
C       EVERY NTH POINT IS PRINTED, WHERE N = IPRT
C
C
C
      SUBROUTINE FUNCT(F,XO,NFE,DZOBJ,FO,M)
C
      DOUBLE PRECISION DSIN,DCOS,DATAN,DTAN,DSQRT,DSIGN
      DOUBLE PRECISION PHI,PHIR,THETA,RELEV,ZRAY,ZU,ZL
      DOUBLE PRECISION W,DW,WDOT,X,XX,XOR,XKM,Z,ZZ
      DOUBLE PRECISION RAD,RE,ROOT1,ROOT2
      DOUBLE PRECISION SG,SG2,SLX,VERT,XVERT,ZUV,ZLV
      INTEGER NFE,MN,HARMO
      REAL LBDRY, TEMP, F, FO,TWOPI
      CHARACTER*80 ATMTYP,VPLOT
      DIMENSION XDUM(1000),RDUM(1000)
      DIMENSION PHI(500),PHIR(25),Z(500)
      DIMENSION XV(101,25),RV(101,25),NPTS(25),IPTS(25)
      DIMENSION APELEV(25,25),MTC(25),XO(10),DZOBJ(25),A(4),B(4)
      COMMON /IDATA/ZONE,DU,IPRT,XMAX,RVMAX,NPHI,PHINIT(25),N,
     *ZP(50),DIST,PERIOD,TP(50),MM,DR(50),IFLAG
      Z(1)=ZONE
      TWOPI=6.2831
      DATA RE/6.37D06/
C
C
C CALCULATE TP(J) FROM INTERGRAL OF FOURIER SERIES
C
C
      HARMO=(M-1)/2
      AO=XO(1)
      DO 1 I=1,HARMO
       A(I)=XO(I+1)
```

```
 1   B(I)=XO(I+1+HARMO)
     DO2 J=1,N
      SUMA=0.0
      SUMB=0.0
      DO3 K=1,HARMO
       TERMA=A(K)*PERIOD*SIN(TWOPI*K*ZP(J)/PERIOD)/(TWOPI*K)
       SUMA=SUMA+TERMA
       TERMB=-B(K)*PERIOD*COS(TWOPI*K*ZP(J)/PERIOD)/(TWOPI*K)
 3     SUMB=SUMB+TERMB
 2    TP(J)=AO*ZP(J)*0.5+SUMA+SUMB
C
C
     IF(RVMAX.GT.ZP(N)) RVMAX=ZP(N)
  80 CONTINUE
     CALL DENS
C
C        CONVERSION OF INPUT ANGLES TO RADIANS
     DO40 L=1,NPHI
     PHIR(L)=PHINIT(L)/3437.747D0
  40 CONTINUE
     LIM=NPHI
     DX=1000.*DU
C
C
     DO300 K=1,LIM
C        K = RAY NUMBER
     NPTS(K)=1
     XOR=0.0
     XINIT=DX
      PHI(1)=PHIR(K)
      XV(1,K)=0.0
     RV(1,K)=Z(1)
C
     KK=1
     RELEV=Z(1)
     ZST=Z(1)
     XDUM(1)=0.0
     CALL LOC(RELEV,UBDRY,LBDRY,LU,LDEC)
C
C        * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C                                                *
     DO210 J=1,500
     CALL RADIUS(LU,LDEC,RAD)
     X=XINIT
     HU=ZP(LU)
     HL=ZP(LDEC)
C
C        - - - - - - - - - - - - - - - - - - - - - - - - -
C                                        |
     DO200 I=1,500
     ZRAY=-X*X/(2.*RAD)+DTAN(PHI(J))*X+ZST
      ZU=HU-X*X/(2.*RE)
     ZL=HL-X*X/(2.*RE)
    RELEV=HL+(ZRAY-ZL)
```

```
      IF(RELEV.GE.RVMAX) GO TO 10
C
C                                        |
C
C
C     VERTEX TEST
C
       SG=DSIGN(1.D0,PHI(J))

       ISG=IDINT(SG)
      SLX=DATAN(-X/RAD+DTAN(PHI(J)))-DATAN(-X/RE)
       SG2=DSIGN(1.D0,SLX)
      ISLX=IDINT(SG2)
       IF(ISG.NE.ISLX) THEN
            IF(ISG.EQ.1) THEN
             VERT=-0.5D0*RE*RAD*(DTAN(PHI(J)))**2/(RAD-RE)+HL
                 IF(J.EQ.1) THEN
             VERT=-0.5D0*RE*RAD*(DTAN(PHI(J)))**2/(RAD-RE)+Z(1)
                 END IF
              IF(VERT.GT.HU) GO TO 30
            END IF
C
             IF(ISG.EQ.-1) THEN
              VERT=-0.5D0*RE*RAD*(DTAN(PHI(J)))**2/(RAD-RE)+HU
                 IF(J.EQ.1) THEN
             VERT=-0.5D0*RE*RAD*(DTAN(PHI(J)))**2/(RAD-RE)+Z(1)
                 END IF
              IF(VERT.LE.HL) GO TO 50
          END IF
       END IF
C
C
C
C
      IF(RELEV.GE.HU)GO TO 30
      IF(RELEV.LE.HL)GO TO 50
C
C
C
       KK=KK+1
      XDUM(KK)=XDUM(KK-1)+DX
      RDUM(KK)=RELEV
  900 FORMAT(1X,'X = ',F12.2,'   RELEV =',F14.4)
      IF(XDUM(KK).GT.XMAX) GO TO 10
       NPTS(K)=NPTS(K)+1
      X=X+DX
  200 CONTINUE
C                                END OF I-LOOP  |
C       --------------------------
C
   30 CONTINUE
C        START-UP:
     IF(J.EQ.1) GO TO 31
      IF(PHI(J).GE.0.0) GO TO 32
C
```

```
C       REENTRY CASE - EQ.11:
      XOR=2.*RE*RAD*DTAN(PHI(J))/(RE-RAD)
      GO TO 33
C
   31 HL=Z(1)
C
C       NORMAL CASE, UPWARD RAY - EQ.8:
   32 ROOT1=DSQRT(DTAN(PHI(J))*DTAN(PHI(J))+2.*(HU-HL)*(RAD-RE)
     +/(RE*RAD))
      XOR=(RE*RAD/(RAD-RE))*(-DTAN(PHI(J))+ROOT1)
C
   33 PHI(J+1)=DATAN(-XOR/RAD+DTAN(PHI(J)))-DATAN(-XOR/RE)
      SLOPE=3437.75*PHI(J+1)
      LU=LU+1
      LDEC=LDEC+1
      XINIT=X-XOR
      ZST=ZP(LDEC)
      GO TO 209
C       END OF J-LOOP, THIS BRANCH
C
C
   50 CONTINUE
      IF(RELEV.LE.0.0) GO TO 10
C       START-UP:
      IF(J.EQ.1) GO TO 51
      IF(PHI(J).LT.0.0) GO TO 52
C
C       REENTRY CASE - EQ.9:
      XOR=2.*RE*RAD*DTAN(PHI(J))/(RE-RAD)
      GO TO 53
C
   51 HU=Z(1)
C
C       NORMAL CASE, DOWNWARD RAY - EQ.10:
   52 ROOT2=DSQRT(DTAN(PHI(J))*DTAN(PHI(J))-2.*(HU-HL)*(RAD-RE)
     +/(RE*RAD))
      XOR=(RE*RAD/(RAD-RE))*(-DTAN(PHI(J))-ROOT2)
   53 PHI(J+1)=DATAN(-XOR/RAD+DTAN(PHI(J)))-DATAN(-XOR/RE)
      SLOPE=3437.75*PHI(J+1)
      LU=LU-1
      LDEC=LDEC-1
      XINIT=X-XOR
      ZST=ZP(LU)
  209 CONTINUE
  210 CONTINUE
      WRITE(4,111)
  111 FORMAT(1X,'PROBLEM *** INTERSEC W. LAYER BDRY NOT ACHIEVED'/)
      STOP
C
C                                END OF J-LOOP   *
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
   10 CONTINUE
C
```

```
C       PRINT EVERY NTH POINT; N=IPRT
C
      KPRT=1+(NPTS(K)-1)/IPRT
      IPTS(K)=KPRT
C       THIS IS THE NUMBER OF DATA PTS TO BE PLOTTED FOR K'TH RAY
      IF(KPRT.LT.2)GO TO 604
         DO600 L=2,KPRT
         MPRT=1+(L-1)*IPRT
         XV(L,K)=XDUM(MPRT)
         RV(L,K)=RDUM(MPRT)
  600 CONTINUE
C
  604 CONTINUE
C
  300 CONTINUE
C
C       END OF K-LOOP: K'TH RAY NOW COMPLETE
C
  500 CONTINUE
C
C
C
C          *****    TRANSFER CHARACTERISTICS    *****
C
C
C
C
      NTC=0
  770   CONTINUE
      NTC=NTC+1
C
C
C DISTANCE FOR WHICH TC IS REQUIRED  (IN KM)
C       WRITE(*,*)'DISTANCE FOR WHICH TC IS REQUIRED (IN KM) :  '
C       READ(*,*)DIST
C     DIST=10
      XIPRT=IPRT
      MTC(NTC)=DIST/(DU*XIPRT)+1.5
      MN=MTC(NTC)
      TCDIST=DU*FLOAT((MN-1)*IPRT)
      DO 708 K=1,LIM
       APELEV(NTC,K)=Z(1)+1000.*TCDIST*TAN(PHIR(K))
       IF(IPTS(K) .LT. MTC(NTC)) THEN
        DRV=RV(IPTS(K),K)-RV(IPTS(K)-1,K)
        IF(DRV .LT. 0.0) THEN
         RV(MN,K)=0.0
        END IF
        IF(DRV .GT. 0.0) THEN
         XINT=1000.*DU*XIPRT
         DZO=(1000.*TCDIST-XV(IPTS(K),K))*DRV/XINT
         RV(MN,K)=RV(IPTS(K),K)+DZO
        END IF
       END IF
  708 CONTINUE
```

```
C
C
C          TC OUTPUT FOR MAPPLOT
C
C
      IF(IFLAG .EQ. 0) GO TO 705
       WRITE(6,*)'MAPPLOT OUTPUT FOR ',TCDIST,'  KM'
      WRITE(6,*)LIM
      WRITE(6,710)(RV(MN,K),APELEV(NTC,K),DZOBJ(K),K=1,LIM)
710    FORMAT(3F9.3)
C
C
C
 705 CONTINUE
 706 CONTINUE
 715   CONTINUE
C
C
C  CALCULATE THE SQUARE ERROR
C
C
     F=0.0
     DO950 I=1,LIM
      F=F+(DZOBJ(I)-RV(MN,I))**2
950   CONTINUE
1000  NFE=NFE+1
C     WRITE(*,*)'NUMBER OF FUNCTION EVALUATION =',NFE
C     WRITE(*,*)'SQUARE ERROR =',F
     RETURN
     END
C
C
C
     SUBROUTINE DENS
C
C       CALCULATION OF DENSITY AT ZP,TP DATA POINTS
C          ZP IN METERS, TP IN DEGREES C.
C          MODIFICATION OF APR.82
C
C     COMMON MM,N,RVMAX,ZP(50),TP(50),DR(50)
     COMMON /IDATA/ZONE,DU,IPRT,XMAX,RVMAX,NPHI,PHINIT(25),N,
    *ZP(50),DIST,PERIOD,TP(50),MM,DR(50),IFLAG
     DATA G,BETA,PNULL/9.81,3.48E-03,1.0137931E05/
     TINT=0.
     DO200 J=1,N
     PWR=-G*BETA*TINT
     DR(J)=(BETA*PNULL/(TP(J)+273.))*EXP(PWR)
     IF(J.GE.N) GO TO 30
     ALPHA=(TP(J+1)-TP(J))/(ZP(J+1)-ZP(J))
     IF(ALPHA.EQ.0.0) GO TO 10
     DTINT=ALOG(1.+ALPHA*(ZP(J+1)-ZP(J))/(TP(J)+273.))/ALPHA
     GO TO 20
  10 DTINT=(ZP(J+1)-ZP(J))/(TP(J)+273.)
  20 TINT=TINT+DTINT
```

```
 200 CONTINUE
  30 CONTINUE
     RETURN
     END
C
C
C
     SUBROUTINE RADIUS(LU,LDEC,RAD)
C        USES AVERAGE LAYER PROPERTIES (TEMP, DENSITY)
C        TO RETURN RAY RADIUS IN METERS
     DOUBLE PRECISION RAD
C     COMMON MM,N,RVMAX,ZP(50),TP(50),DR(50)
     COMMON /IDATA/ZONE,DU,IPRT,XMAX,RVMAX,NPHI,PHINIT(25),N,
    *ZP(50),DIST,PERIOD,TP(50),MM,DR(50),IFLAG
     DATA G,BETA,EPS/9.81,3.48E-03,0.000226/
     DRAV=(DR(LU)+DR(LDEC))/2.
     ER=EPS*DRAV
     TAVK=(TP(LU)+TP(LDEC))/2.+273.
     DTDZ=(TP(LU)-TP(LDEC))/(ZP(LU)-ZP(LDEC))
     CURV=(DTDZ+G*BETA)*ER/((1.+ER)*TAVK)
     RAD=1./CURV
     RETURN
     END
C
C
     SUBROUTINE LOC(Z,UBDRY,LBDRY,LU,LDEC)
C        THIS SR LOCATES  Z  WITHIN THE ZP,TP TABLE
C        GIVES LAYER BOUNDARIES ABOVE & BELOW  Z
     DOUBLE PRECISION Z
     REAL LBDRY
C     COMMON MM,N,RVMAX,ZP(50),TP(50),DR(50)
     COMMON /IDATA/ZONE,DU,IPRT,XMAX,RVMAX,NPHI,PHINIT(25),N,
    *ZP(50),DIST,PERIOD,TP(50),MM,DR(50),IFLAG
     DO10 L=2,N
     IF(Z.LE.ZP(L)) GO TO 20
  10 CONTINUE
  20 LU=L
     LDEC=L-1
     UBDRY=ZP(LU)
     LBDRY=ZP(LDEC)
     RETURN
     END
//GO.FT01F001 DD *
++EMBED JM3OPTI NOSEQ
//GO.FT03F001 DD *
++EMBED JM3 NOSEQ
//G0.SYSIN DD *
//
```

Input Data to "Inversion of Mirage Data" Program

Profile "JM3"

4.0	: Observer elevation (m)
1.0	: Parabolic ray arc step size in x-direction (Km)
1	: Interval between data points of light ray to be saved (Km)
25000.0	: Maximum limit of ray in x-direction (m)
50.0	: Maximun limit of ray in z-direction (m)
20	: Number of ray to be traced (maximum 25)
-2.	: Initial ray angle (minute)
-1.	"
0.	"
1.	"
2.	"
3.	"
4.	"
5.	"
6.	"
7.	"
8.	"
9.	"
10.	"
11.	"
12.	"
13.	"
14.	"
15.	"
16.	"
17.	"
15	: Number of elevation data point to be calculated for temperature
0.	: Elevation data point value
5.	"
10.	"
11.	"
12.	"
13.	"
14.	"
17.	"
20.	"
24.	"
26	"
30.	"
34.	"
39.	"
45.	"
20.0	: The distance between observer and object at which TC is found (Km)
45.0	: Period of Fourier series (m)

Profile "JM3OPTI"

1	: Print control parameter (0-no print out; 1-limited; 2-full)
10	: Number of Fourier series coefficients used to represent temperature
0.3844	: Starting Fourier series coefficient a_0
-0.2247	: a_1
0.0729	: a_2
-0.0087	: a_3
0.0001	: a_4
0.0941	: b_1
-0.0482	: b_2
0.0335	: b_3
0.0001	: b_4
0.0000	: Constant C in Fourier series temperature expression
20	: Number of data point in given TC
3.65	: Z(objective) (m) of given TC
8.70	(note: Z(apparent) of corresponding point is found by given initial
11.86	ray angles)
13.33	"
12.85	"
12.02	"
12.70	"
8.75	"
7.33	"
6.40	"
6.20	"
6.96	"
8.37	"
26.02	"
41.89	"
52.53	"
61.26	"
68.70	"
76.17	"
83.15	"

Profile "BG1"

2.0	: Observer elevation (m)
1.0	: Parabolic ray arc step size in x-direction (Km)
1	: Interval between data points of light ray to be saved (Km)
25000.0	: Maximum limit of ray in x-direction (m)
50.0	: Maximun limit of ray in z-direction (m)
20	: Number of ray to be traced (maximum 25)
-3.	: Initial ray angle (minute)
-2.	"
-1.	"
0.	"
1.	"
2.	"

3.	"
4.	"
5.	"
6.	"
7.	"
8.	"
9.	"
10.	"
11.	"
12.	"
13.	"
14.	"
15.	"
16.	"
19	: Number of elevation data point to be calculated for temperature
0.	: Elevation data point value
2.24	"
4.88	"
7.42	"
9.96	"
12.35	"
14.93	"
17.37	"
19.96	"
22.44	"
24.18	"
27.01	"
29.95	"
34.97	"
46.06	"
57.31	"
69.80	"
79.80	"
89.71	"
20.0	: The distance between observer and object at which TC is found (Km)
50.0	: Period of Fourier series (m)

Profile "BG1OPTI"

1	: Print control parameter (0-no print out; 1-limited; 2-full)
9	: Number of Fourier series coefficients used to represent temperature
0.3983	: Starting Fourier series coefficient a_0
-0.2474	: a_1
0.0601	: a_2
-0.0862	: a_3
0.0001	: a_4
-0.0546	: b_1
0.0288	: b_2
0.0303	: b_3
0.0001	: b_4
20	: Number of data point in given TC

0.00	: Z(objective) (m) of given TC
12.96	(note: Z(apparent) of corresponding point is found by given initial
16.10	ray angles)
18.15	"
19.53	"
20.30	"
20.28	"
20.43	"
17.69	"
15.92	"
12.77	"
11.07	"
17.51	"
30.78	"
39.68	"
47.73	"
55.45	"
62.82	"
69.95	"
76.98	"

BIBLIOGRAPHY

1. W. H. Lehn, "*A Simple Parabolic Model for the Optics of the Atmospheric Surface Layer* ", Applied Mathematical Modelling, **9**, 447-453 (1985).

2. E. E. Gossard and W. H. Hooke, *Waves in the Atmosphere,* (Elsevier Scientific Publishing Company, New York 1975).

3. J. S. Morrish, "*Inferior Mirages and Their Corresponding Temperature Structures*", Theis, Master of Science in Electrical Engineering, University of Manitoba, pp. 6, 1985.

4. F. Verniani, *Structure and Dynamics of the Upper Atmosphere* , (Elsevier Scientific Publishing Company, New York 1974).

5. P. Adby, *Introduction to Optimization Methods*, (Halsted Press, New York 1974).

6. Microsoft C, version 4.0, Microsoft Corporation, 1986.

7. Halo '88, Graphics Kernel System, Media Cybernetics, Silver Spring, Maryland, 1988.

8. W. H. Lehn, "Inversion of Superior Mirage Data to Compute Temperature Profiles", J. Opt. Soc. Am., **73**, 1622-1625 (1983).

9. W. Friesen, "Image Processing on the PC (Mirage Simulation)", Thesis, Bachelor of Science in Electrical Engineering, University of Manitoba, 1988.