MODELLING OF SLOPED ATMOSPHERE MIRAGES

BY

THOMAS L. LEGAL

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba

ISBN 0-612-13289-7

Name_____

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

ELECTRONICS AND ELECTRICAL ENGINEERING

SUBJECT TERM

0	5	4	4

SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

- Architecture 0729
- Art History 0377
- Cinema 0900
- Dance 0378
- Fine Arts 0357
- Information Science 0723
- Journalism 0391
- Library Science 0399
- Mass Communications 0708
- Music 0413
- Speech Communication 0459
- Theater 0465

EDUCATION

- General 0515
- Administration 0514
- Adult and Continuing 0516
- Agricultural 0517
- Art 0273
- Bilingual and Multicultural 0282
- Business 0688
- Community College 0275
- Curriculum and Instruction 0727
- Early Childhood 0518
- Elementary 0524
- Finance 0277
- Guidance and Counseling 0519
- Health 0680
- Higher 0745
- History of 0520
- Home Economics 0278
- Industrial 0521
- Language and Literature 0279
- Mathematics 0280
- Music 0522
- Philosophy of 0998
- Physical 0523
- Psychology 0525
- Reading 0535
- Religious 0527
- Sciences 0714
- Secondary 0533
- Social Sciences 0534
- Sociology of 0340
- Special 0529
- Teacher Training 0530
- Technology 0710
- Tests and Measurements 0288
- Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

- Language
  - General 0679
  - Ancient 0289
  - Linguistics 0290
  - Modern 0291
- Literature
  - General 0401
  - Classical 0294
  - Comparative 0295
  - Medieval 0297
  - Modern 0298
  - African 0316
  - American 0591
  - Asian 0305
  - Canadian (English) 0352
  - Canadian (French) 0355
  - English 0593
  - Germanic 0311
  - Latin American 0312
  - Middle Eastern 0315
  - Romance 0313
  - Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY

- Philosophy 0422
- Religion
  - General 0318
  - Biblical Studies 0321
  - Clergy 0319
  - History of 0320
  - Philosophy of 0322
- Theology 0469

SOCIAL SCIENCES

- American Studies 0323
- Anthropology
  - Archaeology 0324
  - Cultural 0326
  - Physical 0327
- Business Administration
  - General 0310
  - Accounting 0272
  - Banking 0770
  - Management 0454
  - Marketing 0338
- Canadian Studies 0385
- Economics
  - General 0501
  - Agricultural 0503
  - Commerce-Business 0505
  - Finance 0508
  - History 0509
  - Labor 0510
  - Theory 0511
- Folklore 0358
- Geography 0366
- Gerontology 0351
- History
  - General 0578
  - Ancient 0579
  - Medieval 0581
  - Modern 0582
  - Black 0328
  - African 0331
  - Asia, Australia and Oceania 0332
  - Canadian 0334
  - European 0335
  - Latin American 0336
  - Middle Eastern 0333
  - United States 0337
- History of Science 0585
- Law 0398
- Political Science
  - General 0615
  - International Law and Relations 0616
  - Public Administration 0617
- Recreation 0814
- Social Work 0452
- Sociology
  - General 0626
  - Criminology and Penology 0627
  - Demography 0938
  - Ethnic and Racial Studies 0631
  - Individual and Family Studies 0628
  - Industrial and Labor Relations 0629
  - Public and Social Welfare 0630
  - Social Structure and Development 0700
  - Theory and Methods 0344
- Transportation 0709
- Urban and Regional Planning 0999
- Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

- Agriculture
  - General 0473
  - Agronomy 0285
  - Animal Culture and Nutrition 0475
  - Animal Pathology 0476
  - Food Science and Technology 0359
  - Forestry and Wildlife 0478
  - Plant Culture 0479
  - Plant Pathology 0480
  - Plant Physiology 0817
  - Range Management 0777
  - Wood Technology 0746
- Biology
  - General 0306
  - Anatomy 0287
  - Biostatistics 0308
  - Botany 0309
  - Cell 0379
  - Ecology 0329
  - Entomology 0353
  - Genetics 0369
  - Limnology 0793
  - Microbiology 0410
  - Molecular 0307
  - Neuroscience 0317
  - Oceanography 0416
  - Physiology 0433
  - Radiation 0821
  - Veterinary Science 0778
  - Zoology 0472
- Biophysics
  - General 0786
  - Medical 0760

EARTH SCIENCES

- Biogeochemistry 0425
- Geochemistry 0996
- Geodesy 0370
- Geology 0372
- Geophysics 0373
- Hydrology 0388
- Mineralogy 0411
- Paleobotany 0345
- Paleoecology 0426
- Paleontology 0418
- Paleozoology 0985
- Palynology 0427
- Physical Geography 0368
- Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

- Environmental Sciences 0768
- Health Sciences
  - General 0566
  - Audiology 0300
  - Chemotherapy 0992
  - Dentistry 0567
  - Education 0350
  - Hospital Management 0769
  - Human Development 0758
  - Immunology 0982
  - Medicine and Surgery 0564
  - Mental Health 0347
  - Nursing 0569
  - Nutrition 0570
  - Obstetrics and Gynecology 0380
  - Occupational Health and Therapy 0354
  - Ophthalmology 0381
  - Pathology 0571
  - Pharmacology 0419
  - Pharmacy 0572
  - Physical Therapy 0382
  - Public Health 0573
  - Radiology 0574
  - Recreation 0575
  - Speech Pathology 0460
  - Toxicology 0383
- Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences

- Chemistry
  - General 0485
  - Agricultural 0749
  - Analytical 0486
  - Biochemistry 0487
  - Inorganic 0488
  - Nuclear 0738
  - Organic 0490
  - Pharmaceutical 0491
  - Physical 0494
  - Polymer 0495
  - Radiation 0754
- Mathematics 0405
- Physics
  - General 0605
  - Acoustics 0986
  - Astronomy and Astrophysics 0606
  - Atmospheric Science 0608
  - Atomic 0748
  - Electronics and Electricity 0607
  - Elementary Particles and High Energy 0798
  - Fluid and Plasma 0759
  - Molecular 0609
  - Nuclear 0610
  - Optics 0752
  - Radiation 0756
  - Solid State 0611
- Statistics 0463

Applied Sciences

- Applied Mechanics 0346
- Computer Science 0984
- Engineering
  - General 0537
  - Aerospace 0538
  - Agricultural 0539
  - Automotive 0540
  - Biomedical 0541
  - Chemical 0542
  - Civil 0543
  - Electronics and Electrical 0544
  - Heat and Thermodynamics 0348
  - Hydraulic 0545
  - Industrial 0546
  - Marine 0547
  - Materials Science 0794
  - Mechanical 0548
  - Metallurgy 0743
  - Mining 0551
  - Nuclear 0552
  - Packaging 0549
  - Petroleum 0765
  - Sanitary and Municipal System Science 0554
- Geotechnology 0790
- Operations Research 0428
- Plastics Technology 0796
- Textile Technology 0795

PSYCHOLOGY

- General 0994
- Behavioral 0621
- Clinical 0384
- Developmental 0622
- Experimental 0620
- Industrial 0623
- Personality 0624
- Physiological 0625
- Psychobiology 0989
- Psychometrics 0349
- Social 0632

MODELLING OF SLOPED ATMOSPHERE MIRAGES

BY

THOMAS L. LEGAL

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

MASTER OF SCIENCE

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

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

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

Abstract

The purpose of this thesis is to develop a method to simulate a number of mirages observed and photographed at Resolute Bay, Northwest Territories. Since the standard flat model of the atmosphere was unable to simulate these mirages, a new or modified model was needed. The thesis reviews mirage formation and the meteorological conditions and processes that produce the thermal layers necessary for their creation. It also develops mechanisms that can explain how a thermally sloped atmosphere might evolve. Satisfactory simulations of the Resolute mirages were obtained with this new sloped model. These simulations show that the sloped atmosphere model is a solution to the problem of simulating long range mirages.

Acknowledgements

I would like to thank Professor W.H. Lehn for his guidance and support during the last year and eight months. I hope the work done here will prove useful in subsequent research.

I would like to thank my family for their support and assistance.

I would like to thank the Japanese Television Crew for their funding and support for a trip the Tuktoyuktuk and Resolute Bay, Northwest Territories.

Finally, I would like to thank NSERC for their grant, which allowed this research.

Table of Contents

List of Figures viii

Introduction 1
- **1.1 Introduction** 1
- **1.2 Problem** 4
- **1.3 Scope** 4

Characteristics of the Atmosphere 6
- **2.1 Introduction** 6
- **2.2 The Lower Atmosphere** 8
  - 2.2.1 General Characteristics of the Lower Atmosphere 8
  - 2.2.2 Atmospheric Stability 10
  - 2.2.3 Heating and Cooling 14
  - 2.2.4 Horizontal Convergence and Divergence 16
  - 2.2.5 Topographic Uplifting 17
  - 2.2.6 Fronts and Air Masses 17
- **2.3 Sloping Atmospheres** 18

Mirages 22
- **3.1 History of Mirages** 22
- **3.2 Types of Mirages** 24
  - 3.2.1 Normal Atmosphere 25
  - 3.2.2 Inferior Mirages 27
  - 3.2.3 Hillingar Mirages 29
  - 3.2.4 Hafgerdingar Mirages 31
  - 3.2.5 Novaya Zemlya 32
  - 3.2.6 Other Superior Mirages 33
- **3.3 Previous Mirage Research** 34

Simulation of Mirages 37
- **4.1 Introduction** 37
- **4.2 The Flat Atmosphere Model** 39
  - 4.2.1 The Basic Model 39
  - 4.2.2 Ray Tracing in the Flat Atmosphere Model 40
  - 4.2.3 Simulating a Mirage 43
  - 4.2.4 Multiple Atmospheres 45
- **4.3 The Wavy Atmosphere Model** 46
  - 4.3.1 Atmospheric Model 47

4.3.2 Ray Tracing in a Gravity Wave Atmosphere 50
4.3.3 Simulating a Mirage 50
4.4 Inadequacies in Current Simulation Models 51
4.5 The Sloped Atmosphere Model 52
4.5.1 Atmospheric Model 53
4.5.2 Ray Tracing in a Sloped Atmosphere 54
4.5.3 Simulating the Mirage 56

Experimental Results 57
5.1 Software Used in Research 57
5.2 General Procedure for Mirage Simulation 59
5.3 Simulating a Complex Long Range Mirage 60
5.3.1 Generating a Transfer Characteristic 60
5.3.2 Single Flat Atmosphere Model 63
5.3.3 Single Sloped Atmosphere Model 65
5.3.4 Multiple Sloped Atmosphere Models 67
5.3.5 Summary of Results 82
5.4 Simulating a Sequence of Long Range Mirages 83
5.4.1 Generating a Transfer Characteristic 83
5.4.2 Simulating an Average Mirage in the Sequence 86
5.4.3 Creating a Representative Mirage 91
5.4.4 Simulating a Gravity Wave Sequence 93
5.4.5 Summary of Results 102

Summary, Conclusions, and Recommendations 103
6.1 Conclusions 103
6.2 Recommendations for Future Research 105

Experimental Data 106

Ray Tracing and Graphing Program 130

Mirage Simulator Program 180

Bibliography 223

List of Figures

Fig. 1.1a Highway without a Mirage 2
Fig. 1.1b Water Mirage on the Highway 2
Fig. 1.1c Ship on the Sea 2
Fig. 1.1d Flying Dutchman Mirage [Vin1] 2
Fig. 1.1e Walrus in the Ocean 2
Fig. 1.1f Sea Monster Mirage 2

Fig. 2.1 Temperature Structure of the Atmosphere [Don1] 7
Fig. 2.2 Types of Temperature Profiles 9
Fig. 2.3 Dry and Moist Adiabatic Lapse Rates 11
Fig. 2.4 Stable Air with Forced Vertical Motion 12
Fig. 2.5 Unstable Air with Forced Vertical Motion 12
Fig. 2.6 Saturation of Rising Air 13
Fig. 2.7 Daily Heating of the Earth 14
Fig. 2.8 Uneven Heating and Cooling [Don1] 15
Fig. 2.9 Horizontal Convergence 16
Fig. 2.10 Horizontal Divergence 16
Fig. 2.11 Topographical Uplifting 17
Fig. 2.12 Warm Front [Don1] 18
Fig. 2.13 Cold Front 18
Fig. 2.14 Forced Convergence in Stable Air 19
Fig. 2.15 Forced Convergence in Unstable Air 19
Fig. 2.16 Isolated Heating in Stable Air 19
Fig. 2.17 Isolated Heating in Unstable Air 19
Fig. 2.18 Topographic Lift in Stable Air 20
Fig. 2.19 Topographic Lift in Unstable Air 20
Fig. 2.20 Movement of Warm Air Over Cooler Air 20

Fig. 3.1 Lines of Sight in a Normal Atmosphere 25
Fig. 3.2 Trans-Canada Highway East of Winnipeg 25
Fig. 3.3 Hendrickson Island Viewed from a Helicopter 26
Fig. 3.4 Somerset Island South of Resolute Bay 26
Fig. 3.5 Light Rays Associated with an Inferior Mirage 27
Fig. 3.6 Inferior Mirage on the Highway 28
Fig. 3.7 Mirage of Truck 28
Fig. 3.8a The Inverted Image of an Inferior Mirage 29
Fig. 3.8b The Upright Image of an Inferior Mirage 29
Fig. 3.9 The Hillingar Effect on Light Rays 29
Fig. 3.10a Observing the Horizon in a Normal Atmosphere 30

Fig. 3.10b Observing the Horizon in a Hillingar Atmosphere 30
Fig. 3.11 Hendrickson Island seem from Tuktoyuktuk 31
Fig. 3.12 Light Rays associated with the Hafgerdingar Effect 31
Fig. 3.13 A Hafgerdingar Mirage 32
Fig. 3.14 Light Rays Associated with the Novaya Zemlya Effect 32
Fig. 3.15a Sun Picture 33
Fig. 3.15b Sun Picture 33
Fig. 3.16 Inversion Mirage of Somerset Island 33
Fig. 3.17 The Effect of an Inversion of Light Rays 34

Fig. 4.1 Example of a Transfer Characteristic 37
Fig. 4.2 Example of a Temperature Profile 37
Fig. 4.3a Original Image 38
Fig. 4.3b Mirage Image 38
Fig. 4.4 A Temperature Profile with Layer Boundaries 39
Fig. 4.5 Flat Atmosphere Model [Leh1] 40
Fig. 4.6 Tracing a Ray in a Concentric Shell Atmosphere [Leh1] 42
Fig. 4.7 Mirage Simulation in a Atmosphere with Known Temperatures 43
Fig. 4.8 Mirage Simulation Procedure After Ray Tracing 44
Fig. 4.9 Simulated Mirage Image 45
Fig. 4.10 Multiple Atmosphere 46
Fig. 4.11 Atmosphere with Gravity Waves 50
Fig. 4.12 Four Layer Model 51
Fig. 4.13 Mirage Simulation in a Sloped Atmosphere 53
Fig. 4.14a Flat Atmosphere Model 53
Fig. 4.14b Raise Eye and Lowered Ray Angles in Flat Model 54
Fig. 4.14c Sloped Atmosphere Model 54
Fig. 4.15 Initial Two Region Atmosphere 54
Fig. 4.16 Light Ray is Raised and then Traced 55
Fig. 4.17 Two Region Atmosphere with Sloped Second Region 55

Fig. 5.1 Map of Resolute Bay Area 60
Fig. 5.2 Mirage of Somerset Island at Resolute Bay, NWT 61
Fig. 5.3 Close-up Photograph of Somerset Island 61
Fig. 5.4 Original Transfer Characteristic (original.tc) 62
Fig. 5.5 Simulated Mirage of Somerset Island 62
Fig. 5.6 Simulation #1 - Single Region Flat Atmosphere 64
Fig. 5.7 Simulation #2 - Single Region Sloped Atmosphere 66
Fig. 5.8 Simulation #3 - Multiple Region Atmosphere 68
Fig. 5.9 Ray Trace of Two Region Sloped Atmosphere 69
Fig. 5.10 Simulation #4 - New Width for Both Regions (39.9 km - 37.8 km) 70
Fig. 5.11 Simulation #5 - New Width for Both Regions (30.1 km - 47.6 km) 71
Fig. 5.12 Comparison of Temperature Profiles 72

Fig. 5.13 Comparison of Transfer Characteristics 73
Fig. 5.14 Simulation #6 - New Slope for Region Two (10.5 arc minutes) 74
Fig. 5.15 Simulation #7 - New Slope for Region Two (12.633 arc minutes) 75
Fig. 5.16 Simulation #8 - New Slope for Region Two (13.5 arc minutes) 76
Fig. 5.17 Unacceptable Simulations 77
Fig. 5.18 Acceptable Simulations 77
Fig. 5.19 Comparing Region Two Atmospheres 77
Fig. 5.20 Simulation #9 - Three Region Atmosphere (30.1 km, 23.8 km, 23.8 km) . . . 79
Fig. 5.21 Simulation #10 - Three Region Atmosphere (35.0 km, 4.9 km, 37.8 km) . . . 80
Fig. 5.22 Simulation #11 - Three Region Atmosphere (30.8 km, 4.9 km, 42.0 km) . . . 81
Fig. 5.23 Observed Sequence of Mirages 84
Fig. 5.24 Average Mirage of Lowther Island Sequence 85
Fig. 5.25 Lowther Island without a Mirage 85
Fig. 5.26 Transfer Characteristic for Lowther Mirage 85
Fig. 5.27 Simulated Mirage of Lowther Island 86
Fig. 5.28 Simulation A - Slope Simulation (0 arc minutes) of Region Two 87
Fig. 5.29 Simulation B - Slope Simulation (5 arc minutes) of Region Two 88
Fig. 5.30 Simulation C - Slope Simulation (6 arc minutes) of Region Two 89
Fig. 5.31 Simulation D - Slope Simulation (8 arc minutes) of Region Two 90
Fig. 5.32 Simulation E - Representative Mirage 92
Fig. 5.33 Ray Trace through Representative Atmosphere 93
Fig. 5.34 Four-Layer Approximation 93
Fig. 5.35 Wave Sequence #1 98
Fig. 5.36 Wave Sequence #2 99
Fig. 5.37 Wave Sequence #3 100
Fig. 5.38 Wave Sequence #4 101

CHAPTER 1

Introduction

1.1 Introduction

Definition: **mirage** n. **1.** An optical phenomenon that creates the illusion of water often with inverted reflections of distant objects. **2.** Something that is illusory or insubstantial.

- *American Heritage Dictionary*

Definition: **mirage** n. An optical illusion caused by atmospheric conditions, especially making sheets of water seem to appear in a desert or on a hot road.

- *The Oxford Dictionary*

Definition: **mirage** n. deceptive image in the atmosphere, *eg* of a lake in the desert

- *Collins Gem English Dictionary*

Although the above definitions are quite accurate in their characterization of mirages it is easy to be mislead into thinking mirages are strictly imaginary. Mirages are, in fact, veritable images that present themselves to the observer as a result of physical phenomena in the atmosphere.

A mirage is an image of an object, seen at a distance through an atmosphere that acts to distort the image in shape, size and orientation. Often it is impossible to distinguish the identity of the original object from the resulting mirage. The word mirage comes from the French word *mirer* which means to look at or to be reflected. This implies that a mirage is a reflection.

Mirages are a common occurrence in our world. They appear to us in many shapes,

sizes and forms. In cartoons we see characters crawling desperately through the desert towards what they think is an oasis, only to get there and have it disappear. Everyone has heard about the Flying Dutchman, a ghost-ship flying upside-down across the horizon. Other stories describe sea monsters and mermaids. These apparently supernatural occurrences can be explained by mirages. The oasis in the desert is similar to what we see on a hot day on the highway, when the highway looks wet often resembling a lake across the road. As

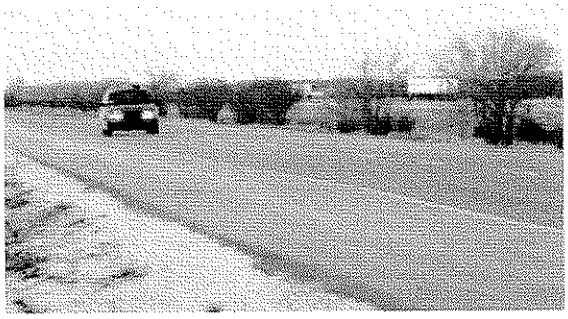


Fig. 1.1a Highway without a Mirage

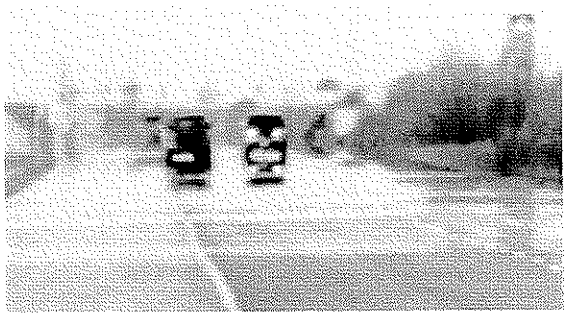


Fig. 1.1b Water Mirage on the Highway

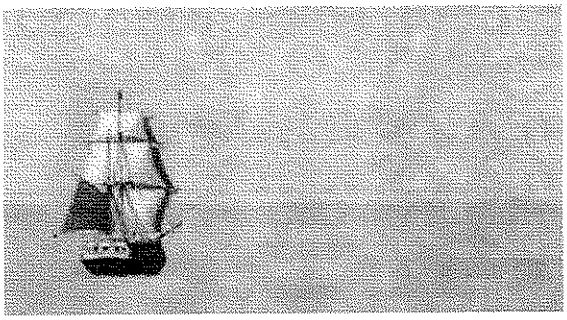


Fig. 1.1c Ship on the Sea

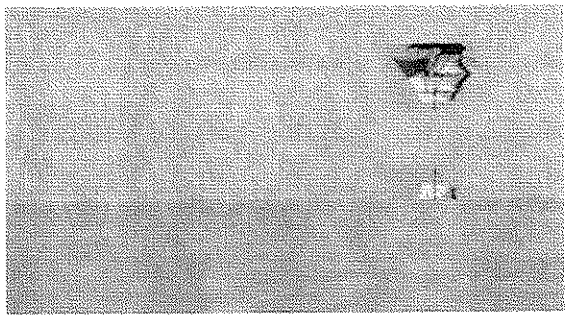


Fig. 1.1d Flying Dutchman Mirage [Vin1]

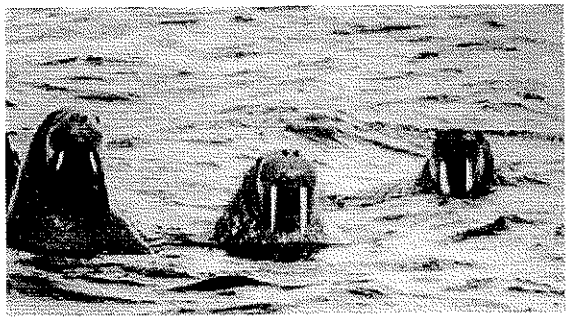


Fig. 1.1e Walrus in the Ocean

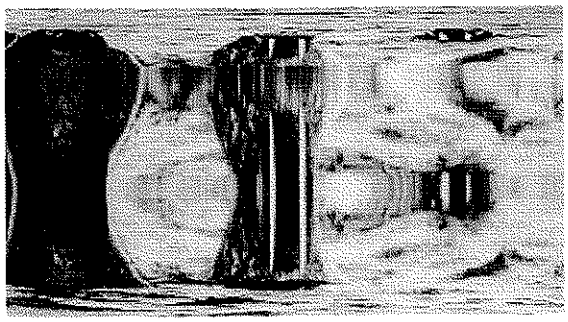


Fig. 1.1f Sea Monster Mirage

illustrated in Fig. 1.1 the water is a mirage of the sky reflecting from the road. The flying Dutchman is the mirage or inverted image of a ship that is beyond the horizon. The ship

appears as an upside-down image with the original hidden from sight by the curvature of the earth. Floating debris or sea animals are distorted to look like huge sea creatures or beautiful mermaids to the sea-weary sailor.

Flying ships, mermaids, lakes in the desert... why do these phenomena occur? The human race has always wanted to explain things that are difficult to understand. Without adequate knowledge of what was really occurring, supernatural explanations were postulated. The explanations however, are in reality more mundane. Mirages occur naturally, like lightning, a rainbow or the northern lights. All have scientific explanations.

A mirage is created by the bending of light rays through the atmosphere. This bending of light produces images similar to those seen in curved mirrors. However, instead of the mirror bending the rays, it is the atmosphere. The atmosphere, like a mirror, can create tall, short, fat or twisted shapes. Atmospheric mirages are known by the following names: inferior, superior, desert, Arctic, Hillingar, Hafgerdingar, Novaya Zemlya and Fata Morgana. Each type has its own unique twist or curve.

The atmosphere can be modelled to simulate mirages. These models alter the vertical temperatures above the earth's surface to distort or bend rays of light passing through the atmosphere. A temperature profile is the common term used to describe the atmosphere's vertical distribution of temperature with height. Like the shape of the mirror, the temperature profile controls what image people see.

The basic model used to trace light rays through the atmosphere assumes that temperatures remain constant along the viewing path of an observer. It also assumes that the earth is relatively flat over these short viewing distances. This type of atmosphere is called a flat atmosphere. This simple model presents obvious problems. For example, at long distances objects are not visible because they are over the horizon. A more advanced model describes the earth as a curved surface, with temperature layers in the atmosphere parallel

to the curved earth. These basic models have worked well to describe simple mirages, but sometimes mirages are more complicated and these models prove inadequate. More complex models can be constructed by accounting for horizontal temperature changes or for changes in temperature over time in the atmosphere.

1.2 Problem

The atmosphere behaves according to the basic model on calm days over flat areas such as lakes, oceans and plains. In other environments this model breaks down. What happens when the ground slopes up or down in the direction of the mirage from the observer? What happens when the temperatures are not constant in the horizontal or when the atmosphere is sloping? These considerations will be necessary to explain some mirages, especially when they appear over long distances.

This thesis proposes to develop a "sloping atmosphere" model and to test it on long range mirages that so far have not been realistically modelled. It also proposes to extend this sloping model to allow for horizontal temperature discontinuities or multiple changes in the atmosphere over distance. This more complex multiple sloping atmosphere model will then be used to examine sequences of mirages and how gravity waves can create them.

1.3 Scope

In developing a model for sloped atmosphere mirages this thesis will begin with a discussion of the atmospheric physics responsible for mirages and the basic models currently used for simulating them. Chapter 2 will describe the atmosphere and some of its characteristics including the mechanisms responsible for producing a sloped atmosphere. Chapter 3 includes a brief history of mirages, the common types of mirages observed, and

a quick review of previous work on mirages. In Chapter 4, methods of mirage simulation for single, multiple and wavy atmospheres are examined. This chapter then presents the sloped atmosphere. The next chapter describes experiments, simulations, and results on long range mirages. The sloped atmosphere model is then tested on gravity waves. Finally, the thesis concludes (Chapter 6) with a summary of important results and recommendations for future research. The appendix contains the data in numerical and image format along with a listing of the mirage simulation program.

CHAPTER 2

Characteristics of the Atmosphere

2.1 Introduction

According to modern theory, the earth was born out of the cosmos as a solid chunk of rock without an atmosphere. Slowly over millions of years the chemical soup that formed the solar system coalesced around the planet. The heavy chemicals were deposited on the planet's surface and lighter chemicals were drawn into an atmosphere. These were the humble beginnings of earth's atmosphere. As life evolved, biological processes liberated oxygen from carbon dioxide and water, and over the span of two to three billion years the earth produced the gaseous mix we have today.

The gases surrounding the earth are called the atmosphere. As gravity decreases at greater distances from the earth's surface, the atmosphere's density also decreases. The decrease in density is exponential. Half of the total earth's atmosphere exists below the height of 5.5 kilometres, and half of the remaining atmosphere is found in the next 5.5 kilometres. Because of this exponential decrease, the upper limit of the atmosphere is difficult to define.

One of the most important classifications of the atmosphere is based on thermal stratification (see Fig. 2.1). This classification divides the atmosphere into four levels: the troposphere, the stratosphere, the mesosphere, and the thermosphere (or exosphere), and the boundaries between them are called the tropopause, the stratopause, and the mesopause. The troposphere found in the lower 11 kilometres above the earth, contains the bulk of the atmosphere. Temperatures decrease nearly uniformly up through this layer which in part is

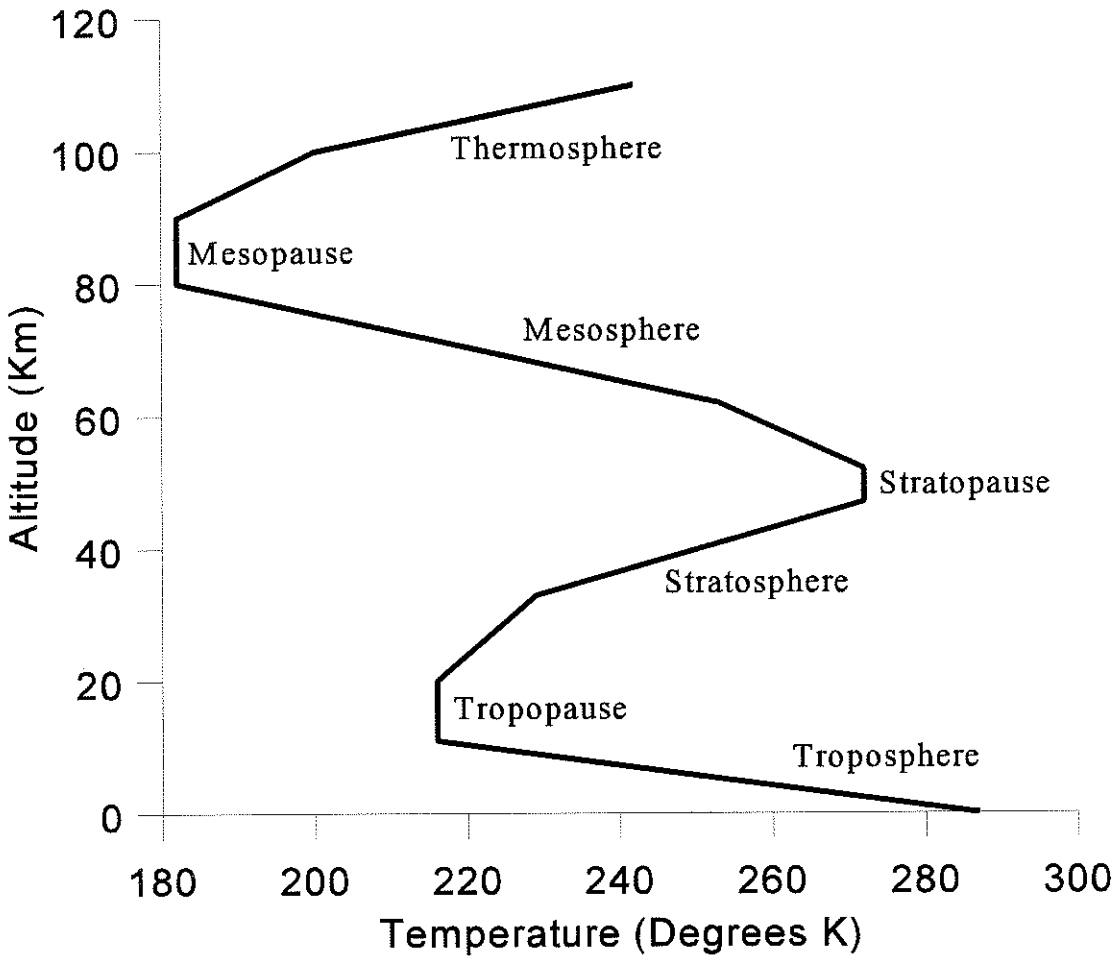


Fig. 2.1 Temperature Structure of the Atmosphere [Don1]

responsible for the earth's weather patterns. In fact, nearly all of the earth's weather events occur in the troposphere. Above the troposphere is the stratosphere, which is characterized by nearly constant or increasing temperatures with height. Above the stratosphere, temperatures decrease again within the layer called the mesosphere, and above the mesopause, the thermosphere is marked by a steady increase in temperature.

Other important parts of the atmosphere not labelled in Fig. 2.1, are the ionosphere, the ozonosphere, and the magnetosphere. The ionosphere, consisting of ionized gases that make long distance radio communication possible, makes up the lower part of the

thermosphere. The ozonosphere is a part of the stratosphere which contains the earth's protective ozone layer. This layer stops the transmission of harmful ultraviolet radiation from reaching the earth's surface. The layer of the thermosphere above the ionosphere is called the magnetosphere.

2.2 The Lower Atmosphere

Of all the regions of the atmosphere, the troposphere is the most important to us. This is where we live. Mount Everest just reaches to the top of the troposphere, and the weather we experience is limited to this region. Only the tallest thunderstorms penetrate the tropopause. The troposphere is also the region of the atmosphere where mirages occur.

2.2.1 General Characteristics of the Lower Atmosphere

In the study of mirages, we are mostly interested in temperature variations within the atmosphere, because these changes influence the bending of light rays. Other factors such as moisture variability also play a role, but the effects of temperature are the most important. Temperature effects can be isolated by holding factors such as humidity, visibility, clouds, precipitation, and atmospheric pressure constant over the viewing distance. A discussion of these other factors will be undertaken in Chapter 3.

Temperature variations are examined in two ways: horizontal temperature variation and vertical temperature variation. Horizontal temperature variation is the change of temperature from place to place on the earth's surface or along a surface of constant height above the earth's surface. Higher latitudes and altitudes on average have cooler temperatures. Coastal locations are often different from those found in the interior of a continent, and alpine temperatures are different from other geographical areas. These

horizontal variations are nearly constant along the earth's surface when considering the short distance normally involved with mirages. Nevertheless, some special cases are important for mirage production and are discussed in later sections.

Vertical temperature variation plays the more important role in the study of mirages. The average decrease in temperature with height is called the *normal lapse rate*, and is about 6.5 °C per kilometre. In other words, for every increase in the altitude of 1000 metres, the temperature on average decreases by 6.5 °C. Temperatures decrease with height because nearly all of the heat within the troposphere is obtained from the earth's surface. Clouds, precipitation and air motions are important in distributing the heat vertically. Any rising air will cool because it is allowed to expand as it is subjected to lower pressures.

Temperatures do not always decrease at a constant rate as described above. Sometimes the rate of decrease changes, and occasionally the temperature may increase with altitude. A layer in which the temperature increases with height is called an inversion.

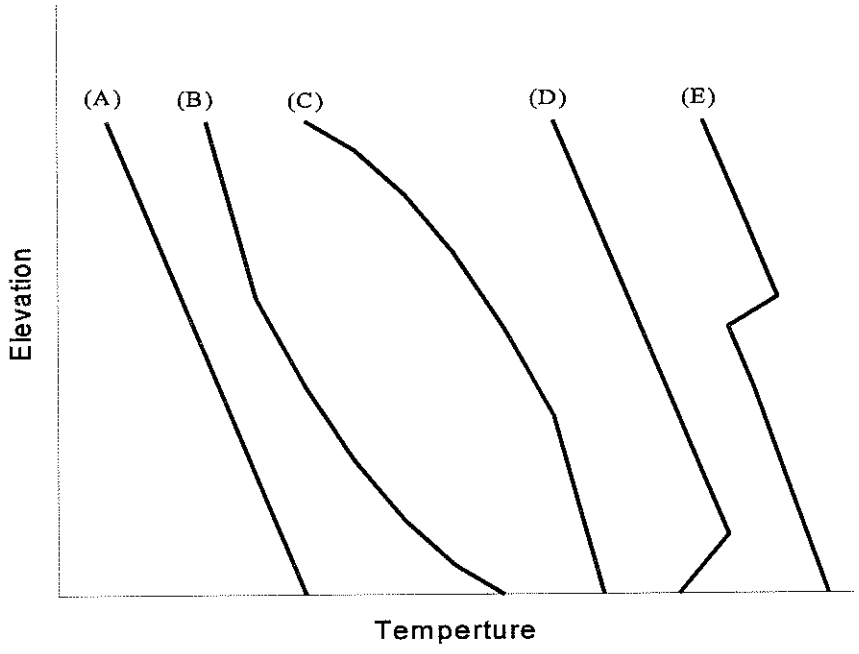


Fig. 2.2 Types of Temperature Profiles

Figure 2.2 shows several different lapse rates. Curve A, shows a constant lapse rate, and curves B and C show lapse rates that decrease and increase respectively with altitude. Possible reasons for the occurrence of such curves will be discussed in following sections. The last two show temperature inversions. Curve D is known as a ground-based inversion and Curve E as an upper-air inversion.

An inversion may result from air near the ground cooling off faster than the overlying air due to heat loss to the cold land; from an actual warm layer passing over a lower cold one; from warming by subsidence; or from turbulence [Don1]. Surface-based inversions occur frequently at night as clear skies allow the earth to radiate heat into space. Other than direct heating or cooling of the lower troposphere by contact with the earth's surface, two other processes act to alter the atmosphere's lapse rate. They are ascending or descending vertical motions and horizontal movements (advection) of cooler or warmer air at different rates in the vertical.

2.2.2 Atmospheric Stability

The stability of the atmosphere is defined by the lapse rate, and is important in the formation of mirages. A stable atmosphere is explained as a body of air that is at rest and in equilibrium. Any vertical motion in this stable atmosphere does not cause a loss of equilibrium but a return to equilibrium. An unstable atmosphere is air at rest, in which vertical motion causes that air to change from its rest state to another state.

Stable air as described above, is air that resists any type of vertical displacement. Assessing airmass stability requires a comparison between actual lapse rates in a layer and the adiabatic lapse rates. If a parcel of air is lifted, it will expand and cool because it is subjected to lower pressure. Dry air will cool as it rises at the dry adiabatic lapse rate or

about 10°C per kilometre in the lower troposphere. The amount of water vapour that a volume of air can hold depends on the temperature of the air. Cool air holds less water vapour than warmer air, and cooling of the air will eventually result in saturation and condensation of water vapour. Condensation of vapour releases latent heat.

Lifting a saturated parcel of air will cause cooling, condensation and the release of latent heat. As a result, saturated air will cool less quickly than dry air when it is displaced upward. It will cool at the moist adiabatic lapse rate or about 6°C per kilometre. A sketch of the dry and moist adiabatic lapse rates are shown in Fig. 2.3.

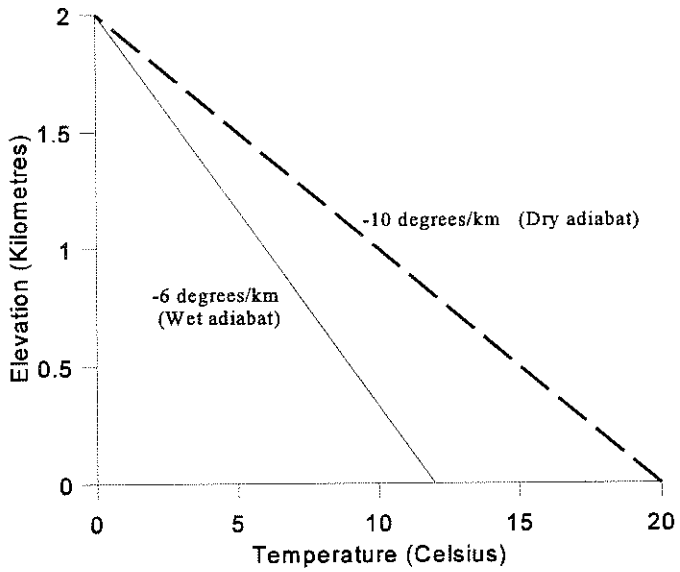


Fig. 2.3 Dry and Moist Adiabatic Lapse Rates

If a parcel of air is lifted vertically and it becomes warmer than the surrounding air, it will continue to rise because it is less dense than that air. Such an airmass would be unstable. Stable air becomes cooler than its environment when it is lifted and would tend to sink because of its greater density.

Dry air that has a lapse rate greater than or equal to the dry adiabatic lapse rate is unstable. Saturated air is unstable when its lapse rate is greater than or equal to the moist

adiabat. Except for layers of the atmosphere very near the ground, vertical mixing ensures that the maximum sustainable lapse rate is that of the dry adiabat. An airmass with a lapse rate equal to or greater than the moist adiabat is conditionally unstable. Lifting such a parcel may cause it to be warmer or more buoyant than its environment.

Figure 2.4 and 2.5 illustrate stable and unstable air masses. In Fig. 2.4, dry air is lifted

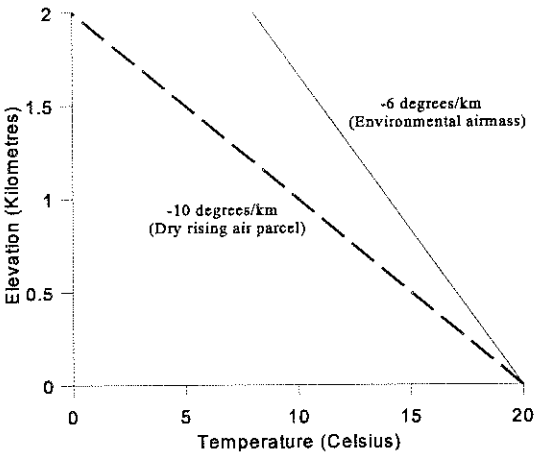


Fig. 2.4 Stable Air with Forced Vertical Motion

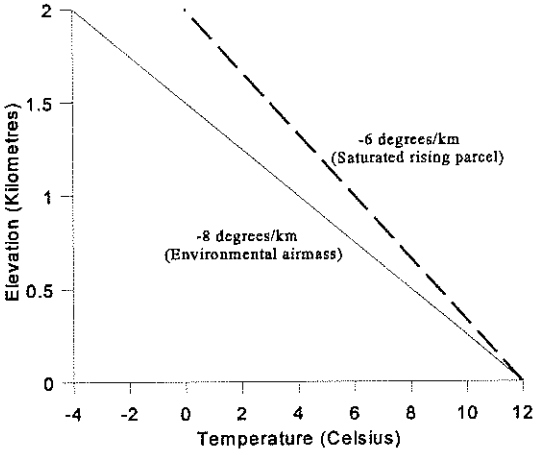


Fig. 2.5 Unstable Air with Forced Vertical Motion

and cools at ten degrees per kilometre while its surrounding air cools at only 6 degrees. After any upward vertical displacement, it will be cooler than its environment and tend to sink. Figure 2.5 illustrates the unstable case where the lifted air is warmer and more buoyant then its surrounding.

An airmass that is conditionally unstable will require a parcel to be lifted to some height above the surface before the instability is realized. For example (Fig. 2.6),an airmass with a lapse rate of 10°C per kilometre requires lifting to 0.5 km to become saturated. As the air is lifted further, it will now cool at 6 degrees per kilometre. However, it remains cooler than its surroundings until it reaches about 1.0 kilometres above the surface. Any more lift will cause the parcel to be warmer than its environment and unstable.

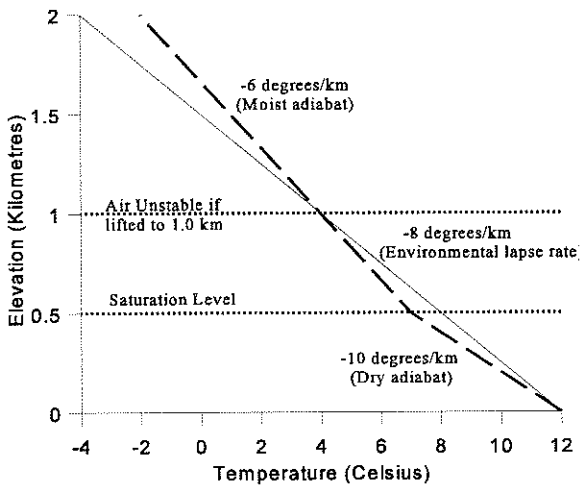


Fig. 2.6 Saturation of Rising Air

Two additional factors that affect stability are worth mentioning. Inversions promote stable conditions and reduce vertical motion. Thus, if the lower atmosphere is unstable, rising air reaches an inversion and slows. A second factor that affects stability is the possibility that a whole layer of air, not just a parcel as described earlier, is forced to rise. This air may initially be stable but the air will become unstable due to internal temperature-humidity conditions. If an airmass possesses these conditions, it is "potentially unstable". The opposite is also possible. An initially unstable layer of air when lifted will become stable. In this case the air is not potentially unstable.

In summary, an air mass can be absolutely unstable, conditionally unstable or potentially unstable with respect to vertical displacements. Temperature lapse rates and the moisture content of the air determine the airmass stability. Factors causing vertical motions in the atmosphere and their effects on stable and unstable air will be discussed in following sections.

2.2.3 Heating and Cooling

The sun is the source of heat for the earth. During the day the sun warms the earth and at night the earth cools releasing its heat. The graph in Fig. 2.7 shows a curve representing heat loss by the earth. Minimum heat loss occurs just before sunrise and

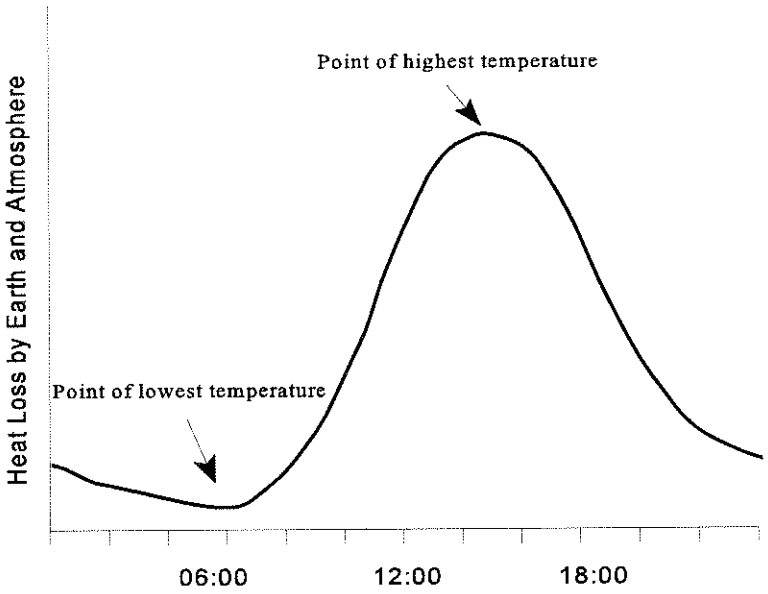


Fig. 2.7 Daily Heating of the Earth

represents the point of lowest temperature during the day. The maximum temperature for the day coincides with the point of maximum heat loss, a couple of hours past noon. To understand this curve, we note that the sun begins to warm the earth at sunrise and reaches its maximum input of heat at noon. The earth continues to warm for a couple of hours, then as the sun approaches the horizon, the earth begins to cool. The cooling process continues until the next sunrise. The steepness of the heating and cooling curves as well as the range between maximum and minimum depend on the length of day and night. In a 24-hour daylight region, this curve is almost flat.

The heating and cooling process described above influences how the atmosphere

heats and cools. When the sun warms the earth, the earth retains the heat and slowly releases it, warming the air. When the sun sets, the earth cools, cooling the atmosphere with it. This is one reason for the development of inversions. The atmosphere lying close to the earth is cooled below the temperature of layers above. Morning cooling inversions are responsible for many of the mirages we experience. Strong heating of black surfaces can produce super-adiabatic lapse rates in a shallow layer of air above the surface. These super-adiabatic lapse rates produce the inferior mirages frequently observed over roads and other surfaces.

Uneven or isolated heating and cooling creates vertical motion in the atmosphere. If one location on the planet warms more quickly than another location, air above the heated surface will be warmer and more buoyant than the surrounding air. This process is common across boundaries between different earth surfaces such as the sea and the land, grasslands and forest, or some other combination. Black earth absorbs more heat than a highly reflective surface like ice or snow. For example, an island in the arctic is surrounded by frozen ocean (Fig. 2.8). The island absorbs sunlight and heats quickly while the surrounding ice reflects much of the short wave radiation and remains cool. Thus the air over the island

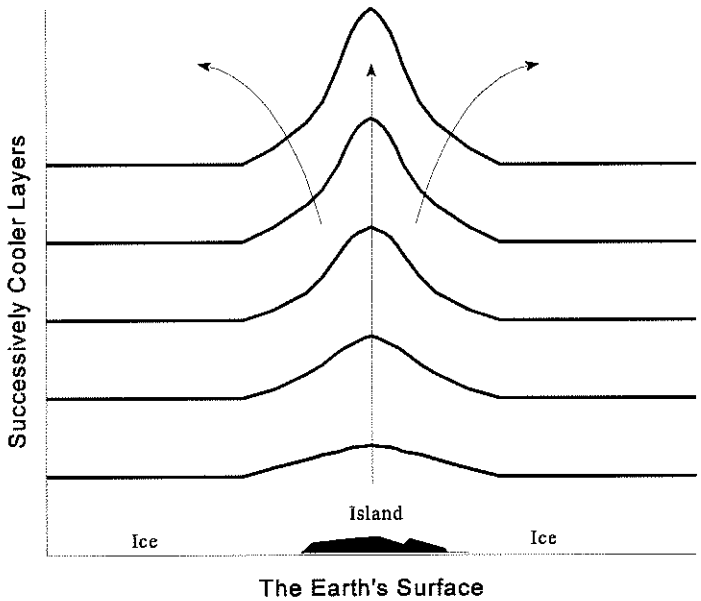


Fig. 2.8 Uneven Heating and Cooling [Don1]

heats and rises causing constant temperature surfaces to rise. The extent and magnitude of the lift and temperature change will depend on whether the atmosphere is stable or unstable. In a stable atmosphere, lift will decrease with height, but in an unstable atmosphere, the lift will increase with altitude and the warm air will spread outward as depicted in Fig. 2.8.

2.2.4 Horizontal Convergence and Divergence

Horizontal convergence and divergence is produced when air is forced upward or downward. However, converging or diverging airflows can be the forcing mechanism which produces the vertical motion. For example, a trough of low pressure or an axis of maximum surface winds are areas of convergence that produce lift. The figures (Fig. 2.9 and Fig. 2.10) below illustrate the effects of convergence and divergence. In the convergence process, air is forced horizontally across the surface of the earth from different directions to converge at a single point. Since air cannot accumulate at any point, it is forced upward and results in

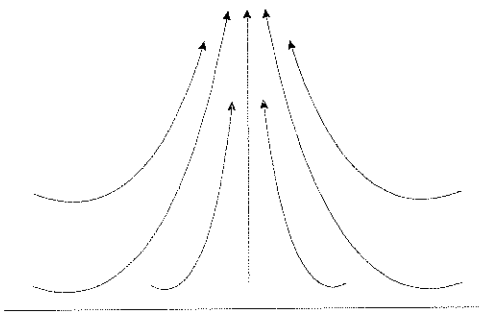


Fig. 2.9 Horizontal Convergence

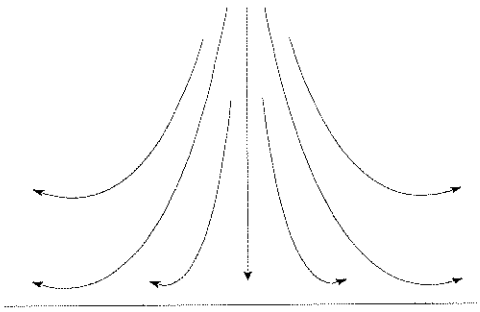


Fig. 2.10 Horizontal Divergence

vertical motion. Divergence has the opposite effect. As air leaves a point on the earth's surface, air is drawn down from above to replace the vacated divergent flow. Thus, downward vertical motion is created. Both these effects are shown in the figures above, where arrows represent the direction of air flow.

2.2.5 Topographic Uplifting

Topographical uplifting, like convergence and divergence, causes air to rise. This process is illustrated in Fig. 2.11. When air moves toward a range of mountains, it is forced to rise on the windward slopes. As it rises it loses most of its water, until it reaches the top of the mountain range. The dry air then descends on the leeward side warming and further

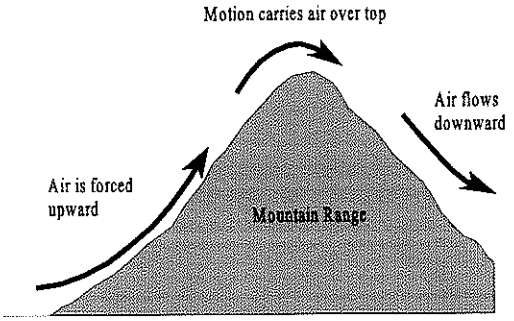


Fig. 2.11 Topographical Uplifting

drying as it loses altitude. These processes are responsible for dry leeward slopes and for warm chinook winds.

2.2.6 Fronts and Air Masses

Fronts cause air to be forced upward just like topographical uplift but on a much larger scale. A front is the boundary between two large air masses, typically between masses of cold and warm air. Two common fronts are the warm front and the cold front. Figure 2.12 shows a warm front moving towards a mass of cooler air. The cooler air acts a wedge

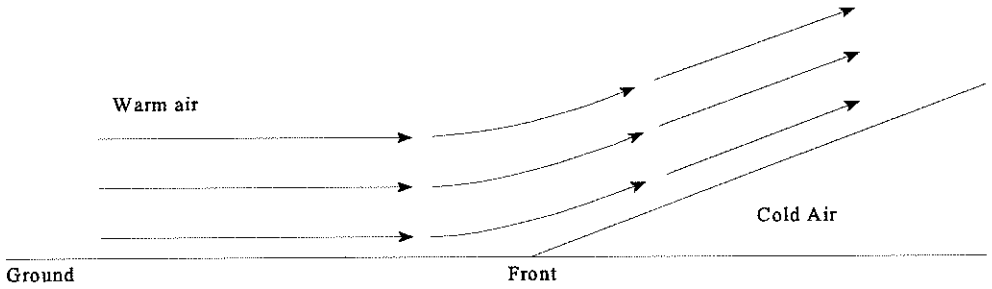


Fig. 2.12 Warm Front [Don1]

forcing the warm air to rise. Figure 2.13 shows a cold front moving toward a mass of warmer air. The cold air again acts as a wedge forcing the warm air upward. The main

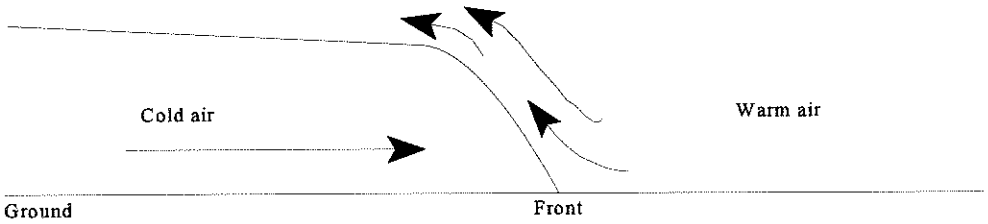


Fig. 2.13 Cold Front

difference between fronts and other types of vertical air motion is that fronts do not lift air vertically. The air is usually lifted at a incline. The slope of the incline depends on the strength of the front. Frontal surface slopes vary from 1:100 to 1:500 (6-35 minutes of arc) with cold fronts being steeper than warm fronts [Don1, P.269].

2.3 Sloping Atmospheres

Temperature levels or isotherms can develop slopes even over the short distances normally involved in mirage viewing. Sloping temperature profiles which are favourable for mirage formation result from vertical motion processes.

Convergence and divergence cause parcels of air to rise or descend. These vertical

motions in turn produce an overall cooling or warming of the column of air above the current. For a stable atmosphere, this would create the effect shown in Fig. 2.14. Similarly, an unstable atmosphere produces the effect in Fig. 2.15. In the stable atmosphere, isothermal

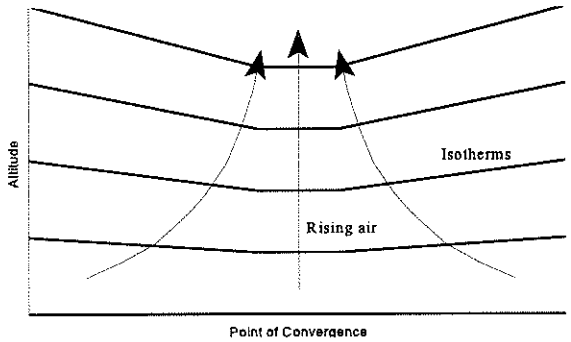


Fig. 2.14 Forced Convergence in Stable Air

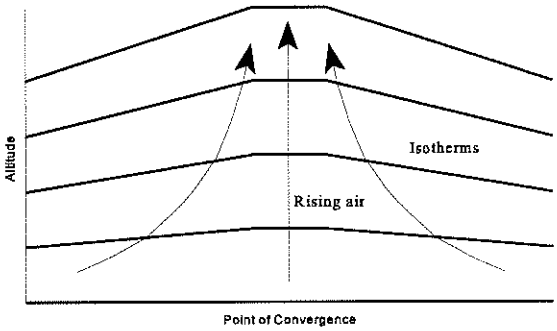


Fig. 2.15 Forced Convergence in Unstable Air

surfaces increase in slope with altitude, sloping down toward the point of convergence. In the unstable atmosphere, isotherms will increase in slope with altitude, sloping up toward the point of convergence. Similar examples could be shown for divergence with arrows pointing downward.

Isolated heating and cooling can produce thermal slopes that are different from those of forced convergence and divergence. Figures 2.16 and 2.17 below show the possible effects of isolated heating with relatively stable and relatively unstable air. The slopes for

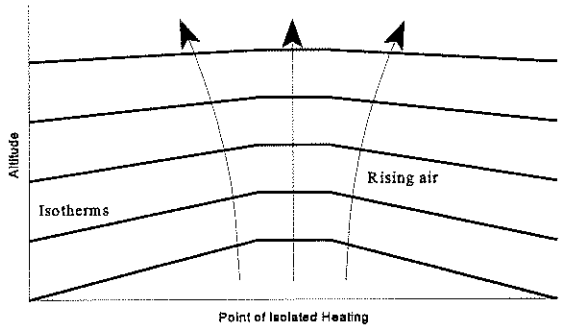


Fig. 2.16 Isolated Heating in Stable Air

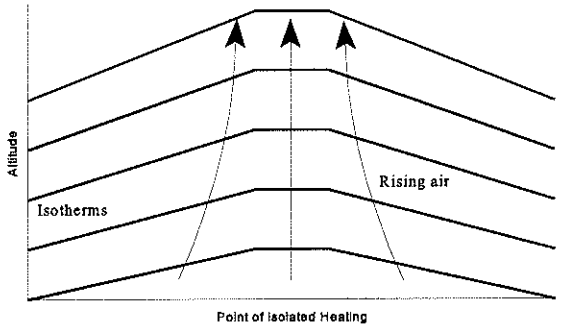


Fig. 2.17 Isolated Heating in Unstable Air

both the stable and unstable atmosphere angle upward toward the point of isolated heating.

However, in stable air, the slope decreases with height while the slope increases with height in the unstable air.

Topographical uplifting changes the isothermal structure in a similar way to that of forced convergence. In both cases parcels of air are forced upwards. In a stable atmosphere, the slope of the isotherms will decline toward the topographic feature. In an unstable atmosphere, the slope of the isotherms will increase toward the topographic feature. These two effects are illustrated below in Fig. 2.18 and 2.19 respectively.

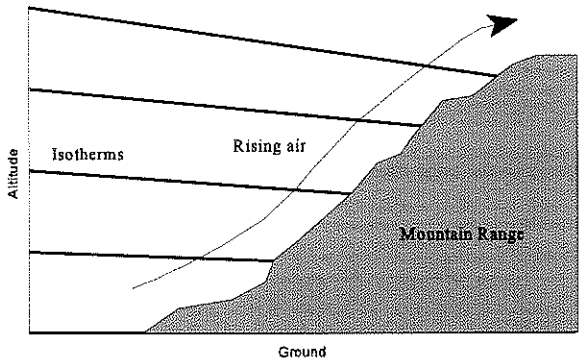


Fig. 2.18 Topographic Lift in Stable Air

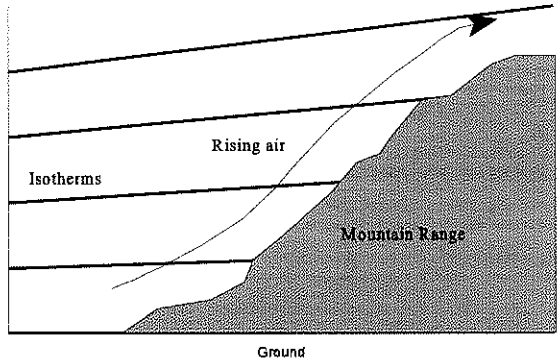


Fig. 2.19 Topographic Lift in Unstable Air

Weather fronts can also create sloping thermal surfaces, but weather fronts are large scale features which generally can be ignored in the study of mirages. Smaller scale lake breeze or sea breeze fronts, however, could produce sloping temperature surfaces at the

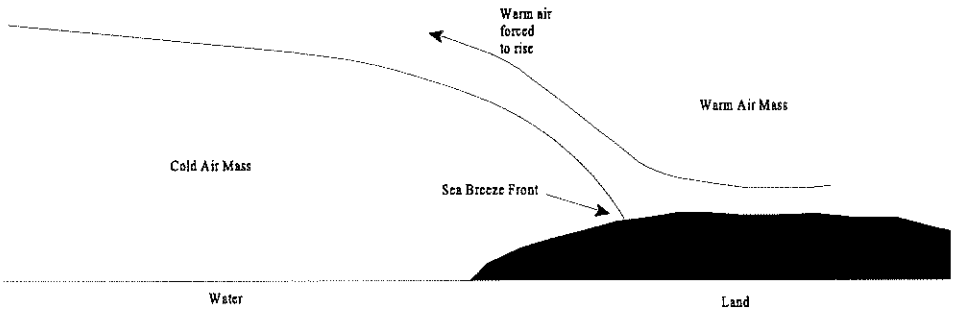


Fig. 2.20 Movement of Warm Air Over Cooler Air

appropriate scale for mirages. Figure 2.20 shows a sea breeze front with warmer air moving

over the cooler air below the frontal surface. This effect would produce a sloping atmosphere in the same manner as topographical uplift.

CHAPTER 3

Mirages

3.1 History of Mirages

Mirages have puzzled the human race, for as long as there have been eyes to see them. Early Christian, Buddhist and Greek writings allude to strange occurrences such as people walking across water [Fra1]. Remarkable observations such as these can possibly be explained as examples of mirages. During the era of Alexander the Great, a monument was built as a tribute to the sun and included a blue and green rim. The blue and green rim was a representation of the phenomenon called the green flash. The green flash is a type of mirage called the Novaya Zemlya effect [Fri1]. Other mirages already cited include mermaids, sea monsters, and the Flying Dutchman. All of these early references to mirages are an integral part of our history and legends. The scientific documentation of mirages is where we begin our history of mirages.

In 1596 Captain Willem Barentz led a mission to the North in search of a northeast passage to Asia. It was during this mission that Barentz was forced to winter on an island called Novaya Zemlya in the Barents Sea which is now a part of Russia. Gerrit de Veer, a crewmate of Barentz, recorded in his journal that a sunrise was observed on January 24, 1597. This observation was rather remarkable because at their latitude of 76° 12' N, the first sunrise for the season was still two weeks away [Leh3]. Although the observation was questioned by the scientific community, it is now believed to be the first recorded sighting of the type of mirage created by the Novaya Zemlya effect.

In 1643 Father Angelucci was one of the first to document the type of mirage known

as the Fata Morgana mirage. The Fata Morgana mirage was named for the fabled King Arthur's sister Morgana who had the power to create castles in the air. On the morning of August 14, the priest watched an amazing display over the ocean from his window in the city of Reggio, Italy. He wrote "the ocean that washes the coast of Sicily rose up and looked like a dark mountain range." He also observed that in front of the mountains, "there quickly appeared a series of more than 10000 pilasters which were a whitish-gray colour", but then "the pilasters shrank to half their height and built arches like those of Roman aqueducts". Before it all vanished, castles appeared above the aqueduct, each with towers and windows [Fra1]. This vivid display was created by mirages over the ocean. The mountain, pilasters, aqueducts and castles were only distorted images of waves on the sea.

Another Fata Morgana mirage led explorer Robert E. Peary in 1906 to believe that a land of "snow-clad summits above the ice horizon" existed off Cape Columbia on Ellesmere Island. Donald B. MacMillan also observed a similar sight in 1913, prompting an expedition to this new "Crocker Land". He wrote "There could be no doubt about it. Great heavens, what a land! Hills, valleys, snow-capped peaks extending through at least 120 degrees of the horizon." When his expedition hiked across the ice to get to this land, it disappeared. "Crocker Land" was a mirage [Fra1].

These observations by Peary and MacMillan are called superior or Arctic mirages. This is also the general name for the Fata Morgana and Novaya Zemlya mirages. In 1915, E. Shackleton made further observations of the Novaya Zemlya effect on a trip to the Antarctic [Sha1]. Three decades later, S.Visser showed that temperature inversions could produce these mirages [Vis1]. In 1952, G. Liljequist made observations to support these conclusions. He also observed other types of mirages [Lil1]. S. Visser's demonstration and G. Liljequist's observations launched the modern study of mirages [Leh3].

3.2 Types of Mirages

Desert mirages and Arctic mirages are the two basic types of mirages. The desert mirage, also known as an inferior mirage, is the more common of the two types. This is the mirage seen on the highway when it looks as if there is water in the distance. The inferior mirage occurs where there is a hot surface such as a road below cooler air. The less common Arctic mirage is also called a superior mirage. Hillingar, Hafgerdingar and Novaya Zemlya are specific types of superior mirages. These mirages require an increase in temperature with height and most often occur in the Arctic or over the ocean. Ice or cold ocean water cools the relatively warm air from below producing the necessary inversion.

Understanding how a mirage is created requires a discussion of the behaviour of light rays as they pass through media of different density. The *index of refraction* is defined as the ratio of the velocity of an electromagnetic wave in one medium such as air to its velocity in a second medium such as a vacuum [Fri1]. In looking at mirages, we are interested in the rays of light travelling from an object to an observer.

Light will travel in straight lines when it passes through a medium with a constant index of refraction. When the index of refraction in the path of light changes, the rays will be bent. Since pressure and temperature are properties of the atmosphere that determine the index of refraction, a volume of air with constant pressure and temperature will also have a constant index. Since pressure decreases at a relatively constant rate with altitude and is relatively constant over short distances, temperature is the main factor that affects the index of refraction.

The remainder of this chapter will discuss the normal undistorted atmosphere as well as the abnormal atmospheres required to produce the different types of mirages.

3.2.1 Normal Atmosphere

The normal atmosphere is the atmosphere we observe most of the time. Temperatures and pressures are relatively constant for all lines of sight. The normal atmosphere with its constant index of refraction is depicted in Fig. 3.1. The path taken by any ray of light from any object to an observer is a straight line. In reality these rays have

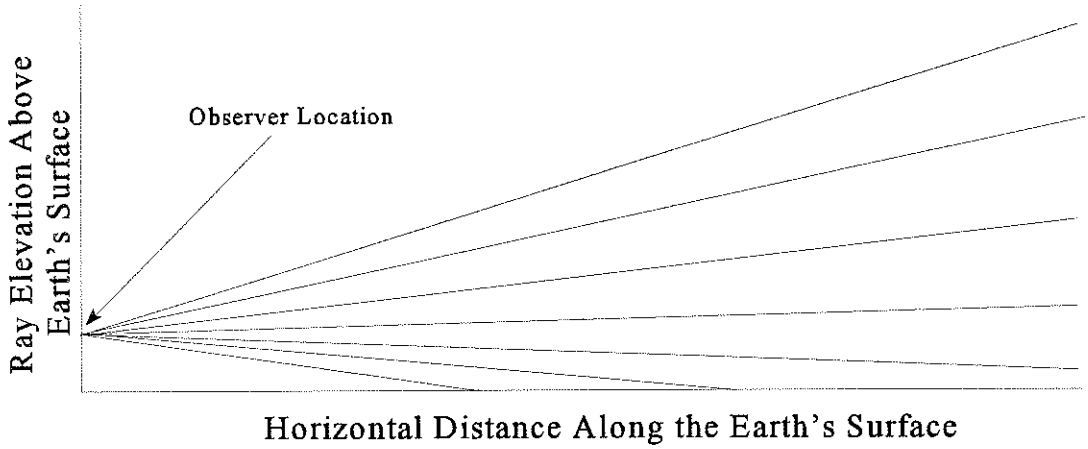


Fig. 3.1 Lines of Sight in a Normal Atmosphere

a slight downward curvature caused by the normal temperature lapse rate above the earth's surface. However, assuming straight line rays of light is a good approximation of reality.

For later comparisons, a number of images in a normal atmosphere are presented.

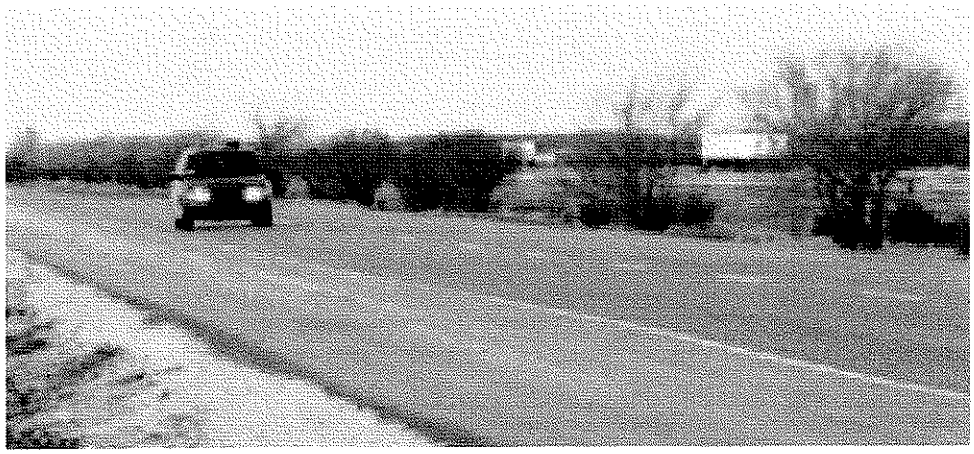


Fig. 3.2 Trans-Canada Highway East of Winnipeg

Figure 3.2 is a picture of the Trans-Canada Highway just east of Winnipeg, Manitoba. The photograph showing a vehicle approaching from the west exhibits no signs of distortion or appearance of water. Hendrickson Island, a small island 22 kilometres west of Tuktoyuktuk

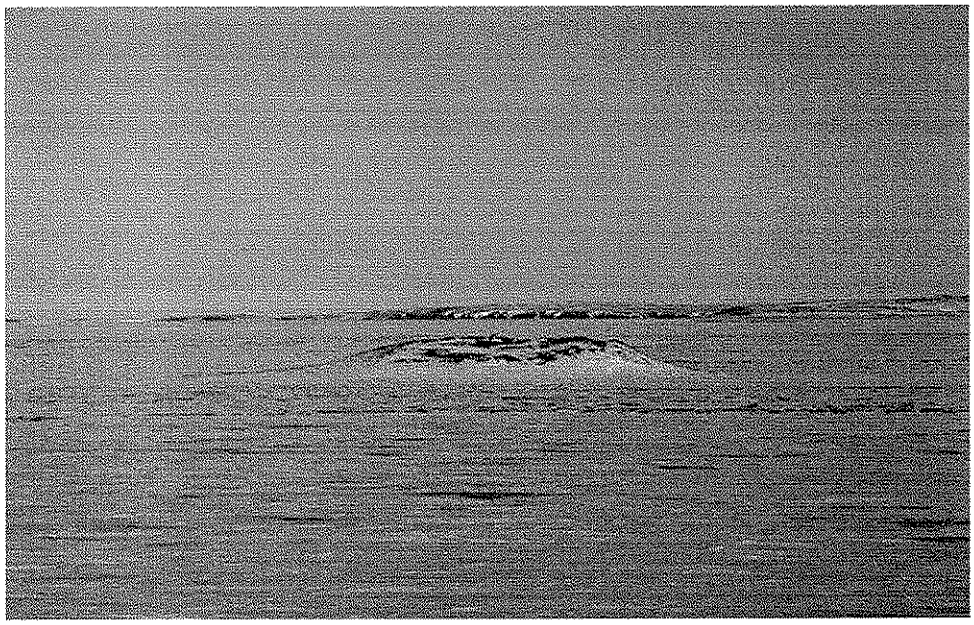


Fig. 3.3 Hendrickson Island Viewed from a Helicopter

in the Northwest Territories (NWT) is shown in Fig. 3.3. The mainland is located behind the island on the other side of the bay about eight kilometres away. The photograph was taken at a distance of three kilometres in a helicopter at low altitude. Finally, Fig. 3.4 shows

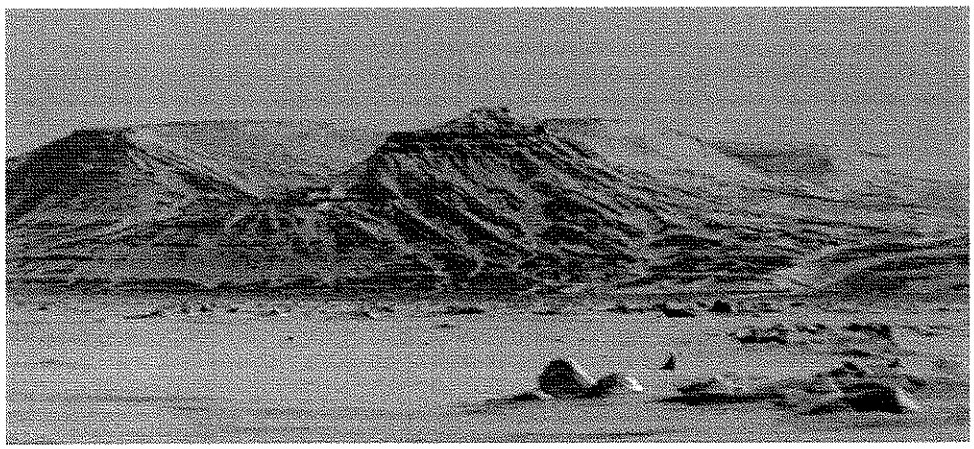


Fig. 3.4 Somerset Island South of Resolute Bay

Somerset Island which is located 75-80 kilometres South of Resolute Bay, NWT. The picture is taken from the ice about 16 kilometres from the island.

Each of these pictures is characterized by clear crisp lines and fine details. This is typical of a normal atmosphere with a constant refractive index. The pictures were taken at relatively close range. At longer distances the atmosphere becomes less consistent and distortions or mirages begin to occur.

3.2.2 Inferior Mirages

An inferior mirage is one that is characterized by upward curving rays of light as shown in Fig. 3.5. Upward curving rays are produced by temperatures that decrease with

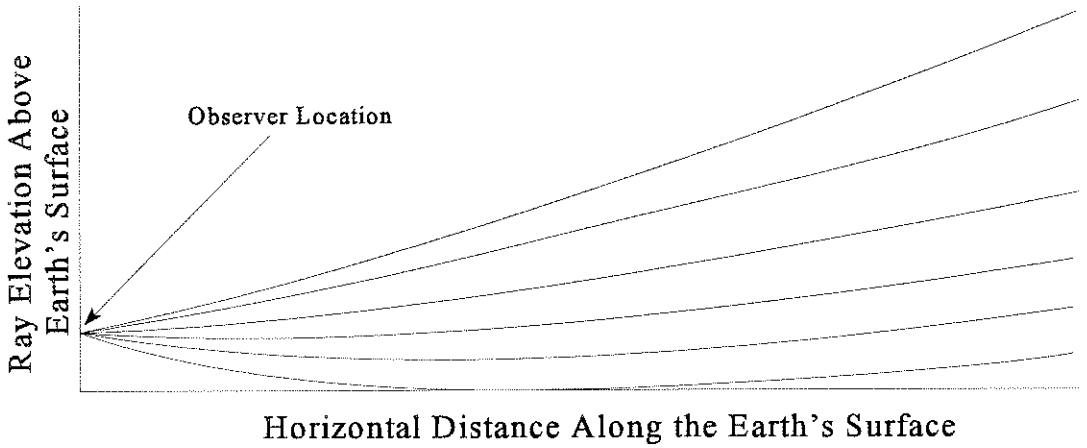


Fig. 3.5 Light Rays Associated with an Inferior Mirage

height. The strong refractive index gradient near the ground causes light rays to bend upward.

Figure 3.6 shows an inferior mirage along the highway. When compared to Fig. 3.2, we see that the road has a watery appearance in the distance. This watery appearance is the sky appearing to be reflected in the road. Typically, with this type of mirage the rays that

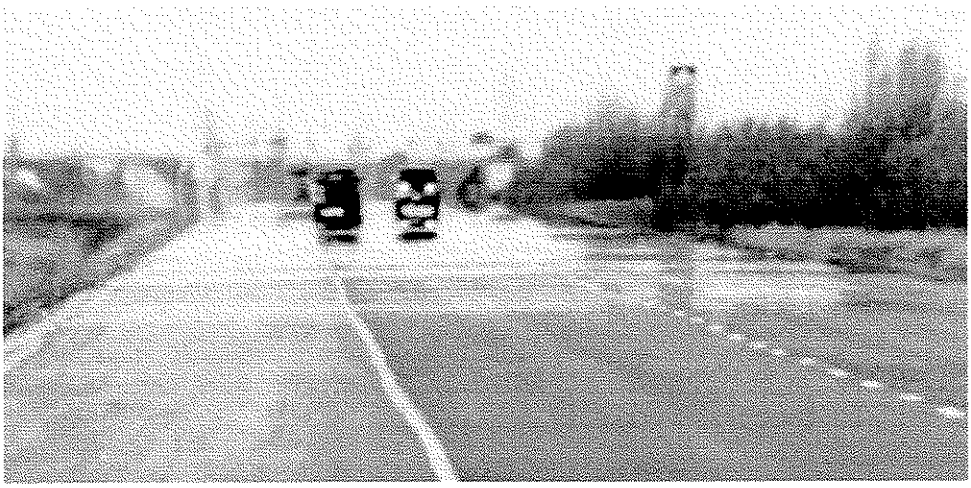


Fig. 3.6 Inferior Mirage on the Highway

pass closer to the ground are bent to a greater extent. This creates the mirror reflection effect of an upside down image. Figure 3.7 is an enlarged view of Fig. 3.6 showing a truck with its headlights turned on. The truck appears with its inverted mirror image underneath it.

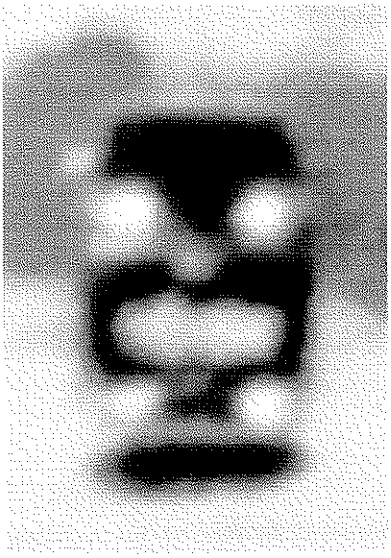


Fig. 3.7 Mirage of Truck

Figure 3.8 shows how the bending of light rays makes the truck appear upside down. In Fig. 3.8a, light rays from the top of the truck take a curved path towards the ground and are then bent upward toward the observer. The truck appears to be inverted on the ground as the projected lines (dashed lines) of sight from the observer illustrate. In Fig. 3.8b, other

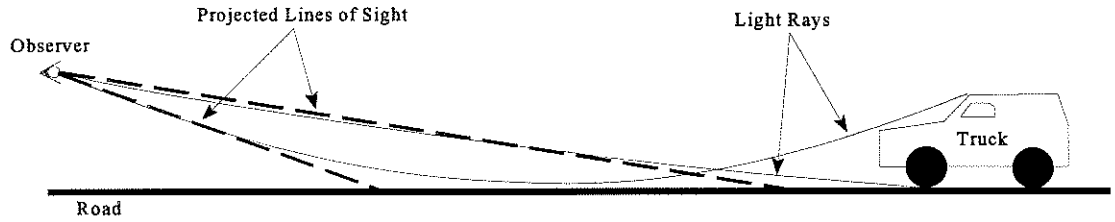


Fig. 3.8a The Inverted Image of an Inferior Mirage

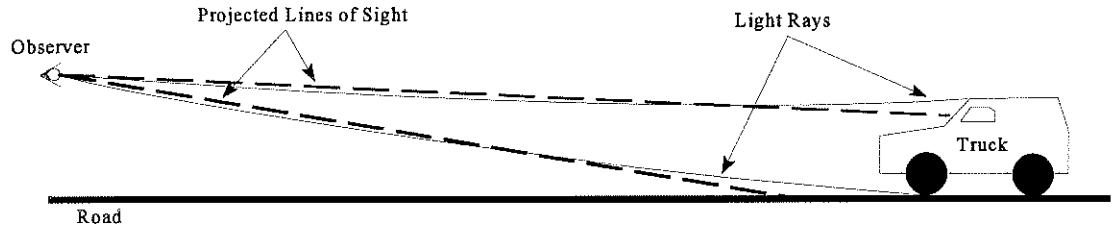


Fig. 3.8b The Upright Image of an Inferior Mirage

light rays travel directly from the top of the truck to the observer. These rays do not pass close to the ground and are bent less. As a result, the observer also sees the upright truck as shown by the projected dashed lines of sight.

3.2.3 Hillingar Mirages

The opposite effect to the inferior mirage, where light rays are bent downward instead of upward, is known as a superior mirage. The type of superior mirage is determined by the

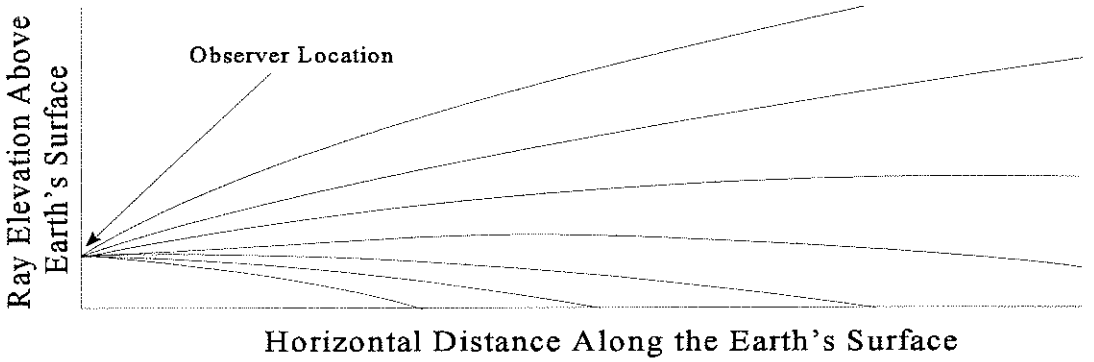


Fig. 3.9 The Hillingar Effect on Light Rays

nature and the degree to which the rays are bent. The Hillingar superior mirage is characterized by mild changes in the index of refraction resulting in only slight bending of light rays as shown in Fig. 3.9. Light rays originating from beyond the horizon are bent, allowing an extended range of sight to the observer. In Fig. 3.10a, an observer stands in a normal atmosphere where lines of sight go to the horizon but not beyond. In Fig. 3.10b, the observer sees beyond the horizon as light rays are bent downward before reaching the eye.

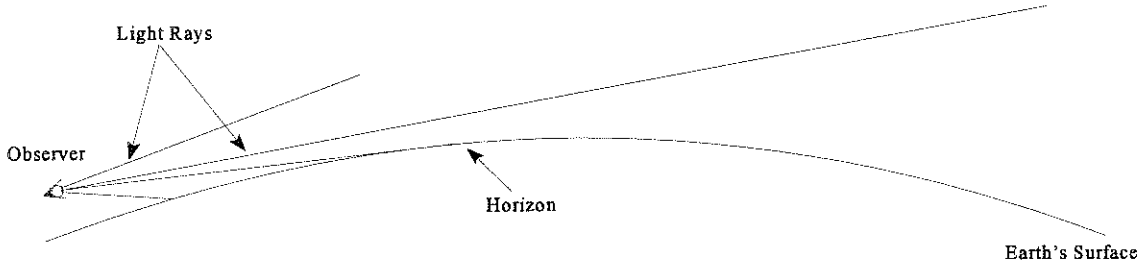


Fig. 3.10a Observing the Horizon in a Normal Atmosphere

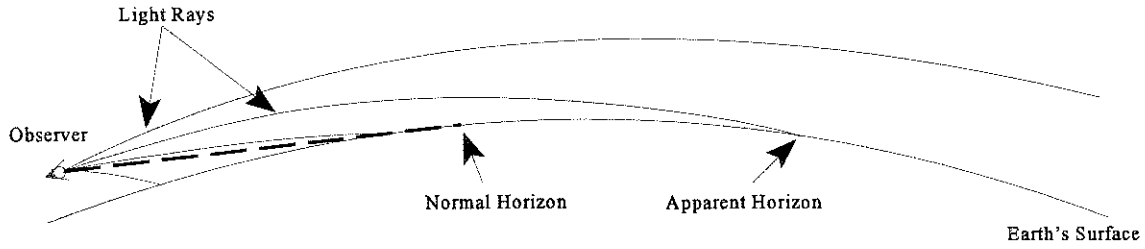


Fig. 3.10b Observing the Horizon in a Hillingar Atmosphere

Figure 3.11 shows a picture of Hendrickson Island with the mainland behind it as viewed from Tuktoyuktuk, NWT. Normally, this picture would show nothing but a horizon of ice. The Hillingar effect, however, bends light rays emanating from Hendrickson Island so that it can be seen at a distance of over 20 kilometres. The land behind the island appears even higher in elevation. This picture can be compared to the undistorted image of Fig. 3.3. The direction of view is the same, but in Fig. 3.3 the observer is only three kilometres away.

Fig. 3.11 Hendrickson Island seem from Tuktoyuktuk

3.2.4 Hafgerdingar Mirages

A Hafgerdingar superior mirage is a strong non-uniform refraction under which the earth's surface appears irregular with vertical discontinuities. Hafgerdingar is an Icelandic word meaning "sea fences" [Leh3]. An example of light rays associated with this type of mirage is shown in Fig. 3.12. The refractive index changes rapidly with height resulting in

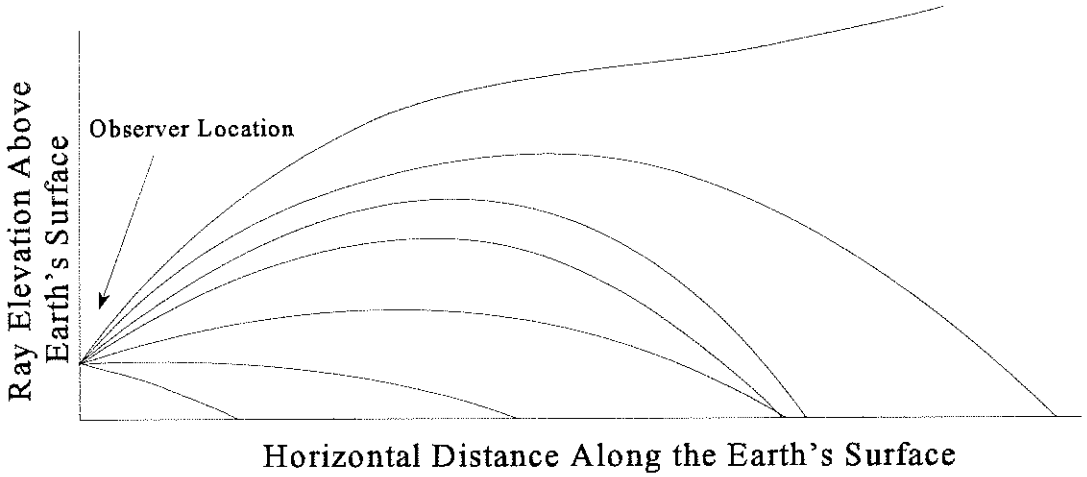


Fig. 3.12 Light Rays associated with the Hafgerdingar Effect

dramatic changes in light ray traces. Figure 3.13 is an example of a Hafgerdingar Mirage of Somerset Island where a palisade or fence effect exists. Figure 3.4 shows a similar view without the mirage effect.

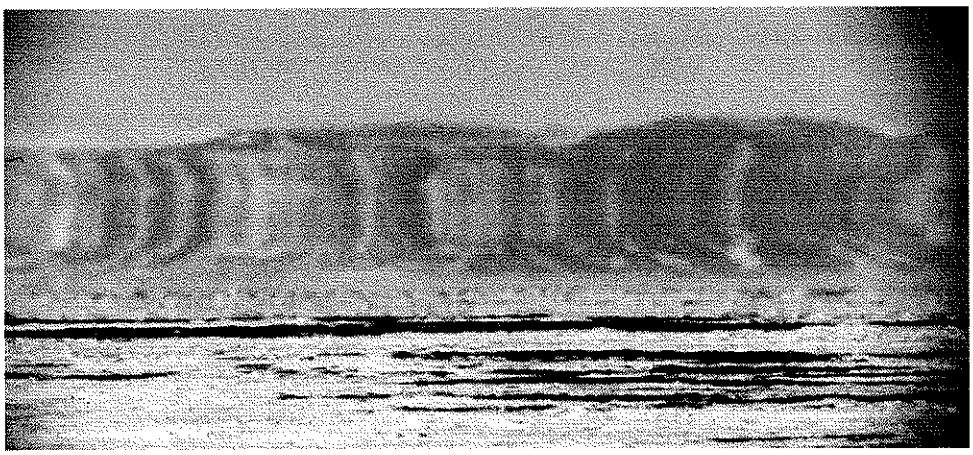


Fig. 3.13 A Hafgerdingar Mirage

3.2.5 Novaya Zemlya

The Novaya Zemlya effect is a type of long range mirage which was first observed from the island of the same name located in what is now North Russia. The atmosphere present for a Novaya Zemlya mirage possesses a strong refracting layer at some elevation above the surface with weaker refraction above and below the layer [Leh3]. Rays of light

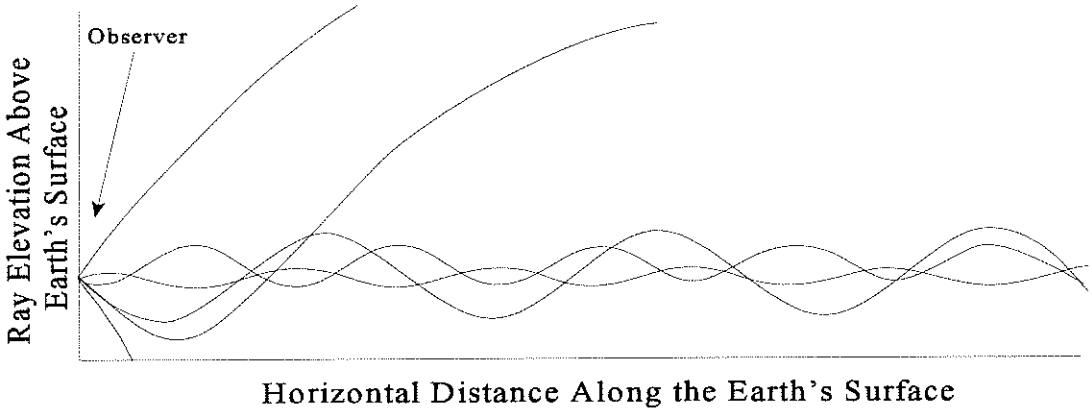


Fig. 3.14 Light Rays Associated with the Novaya Zemlya Effect

are trapped within this highly refractive layer for several hundred kilometres. The trapping of light or other electromagnetic radiation by a temperature inversion and their transmission over a distance is called "ducting". A diagram of this effect is shown in Fig. 3.14. The result effectively creates a "window" in which the observer sees objects at some distance away [Leh3]. Figure 3.15a and 3.15b are Novaya Zemlya observations of the sun. In both

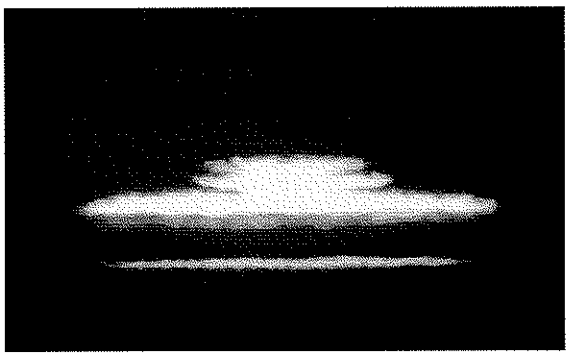


Fig. 3.15a Sun Picture

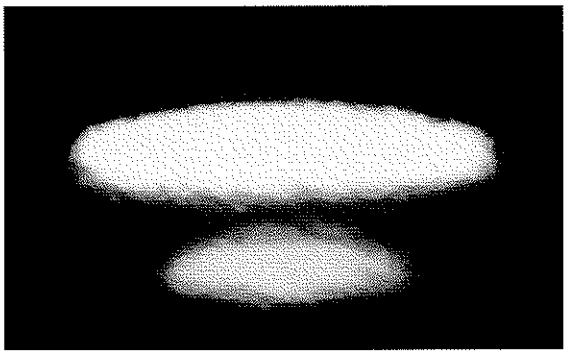


Fig. 3.15b Sun Picture

pictures the sun is below the horizon, but the layer of strong refraction allows it to be seen. The horizontal banding of sun in the two pictures is created by two separate windows of the Novaya Zemlya effect.

3.2.6 Other Superior Mirages

The possible types of mirages that can occur are not limited to the ones discussed

Fig. 3.16 Inversion Mirage of Somerset Island

above. Any combination of both superior and inferior effects can appear in a single mirage. One of these combination mirages was observed at Resolute Bay, NWT on June 6, 1994 (Fig. 3.16). This mirage was created by some inferior effects at lower elevations and superior effects likely caused by an inversion at higher elevations. This inversion acted like a mirror in the sky, flipping the mountain tops in the air above the actual image (Fig. 3.4). Figure 3.17 depicts possible light ray paths that could create such a mirage. This specific mirage is of particular interest to the work of this thesis.

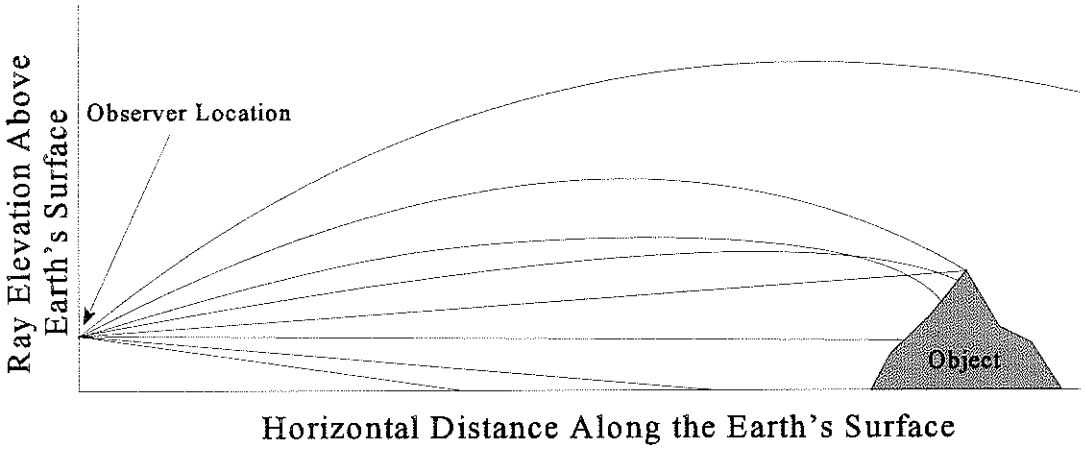


Fig. 3.17 The Effect of an Inversion of Light Rays

3.3 Previous Mirage Research

Although mirages have always been the source of much curiosity it was only the early part of this century when true scientific evaluation of these phenomena began. Much of the early work was done in Germany. The book "Meteorologische Optik", written in 1922, represents the hallmark of this period and remains a standard reference text in mirage study [Per1]. Another important German work, "Zur Theorie der Luftspiegelungen" was written by W. Schiele in 1935 [Sch1]. This paper is particularly valuable because it contains

a complete listing of mirage research prior to 1935.

Later in the century, technological advancements allowed more detailed studies of mirages. In 1964, G. Liljequist produced important atmospheric measurements on mirages [Lil1]. This research provided the actual temperatures in the atmosphere as mirages were taking place. Such measurements proved important in the understanding of how mirages occur.

Some more recent discussions of mirages and other atmospheric phenomena were presented by A. Fraser and R. Greenler [Fra1][Gre1]. These papers and the research outlined above were the basis for mirage research undertaken at the University of Manitoba.

Mirage research at the University of Manitoba began in the 1970's under the direction of Prof. W. Lehn. One of the first papers appeared in the October 1978 issue of *Applied Optics* [Leh6]. This study was written in conjunction with M. El-Arini. The following year a paper published in the *Journal of the Optical Society of America* discussed the theoretical concepts involved in the Novaya Zemlya effect [Leh3]. This is a valuable paper because it describes the effects the atmosphere has on light rays. These effects are central ideas in this thesis. Two additional papers of note used mirages in an attempt to explain the "supernatural phenomena" behind many Myths [Leh7][Leh8].

W. Lehn introduced the process of simulating a superior mirage in a 1983 paper published in the *Journal of the Optical Society of America* [Leh2]. This paper was followed by an article providing more detail on the aspects of tracing rays in the atmosphere [Leh1]. This information forms the basis of the ray tracing theory presented in Chapter 4. The first discussion on the use of image processing for mirage simulation appeared in *Applied Optics* in 1992 [Leh4]. This paper describes the process of simulating mirages with an IBM PC based system, similar to the one used for research on this thesis. In the same year another paper of interest, using the differential geometric approach to atmospheric refraction, was

published in the *Journal of the Optical Society of America* [Kro1]. Most recently, a paper published in *Applied Optics* by W. Lehn, W. Silvester and D. Fraser discussed the possibility of gravity waves and their effects on mirages [Leh5]. This paper provides the background theory of gravity waves presented in Chapter 4.

The previous work at the University of Manitoba represents a large body of research on mirage simulation. This thesis is an extension of that work.

CHAPTER 4

Simulation of Mirages

4.1 Introduction

A mirage can be simulated by tracing light rays from an object to an observer through an atmosphere where the index of refraction of the various atmospheric layers is known. Such a procedure produces a mapping of elevations from an undistorted object to elevations in a distorted image. This mapping is called a transfer characteristic and is used to generate a simulated mirage image from an undistorted picture.

In the real world neither the refraction characteristics of the air nor the temperature profile on which the refraction depends are known. The process of simulating a mirage starts by generating a transfer characteristic (Fig. 4.1) which is simply a map of real elevations against apparent mirage elevations. The temperature profile (Fig. 4.2) of the atmosphere can be inferred from an atmospheric mirage model and from the transfer characteristic through a trial and error process.

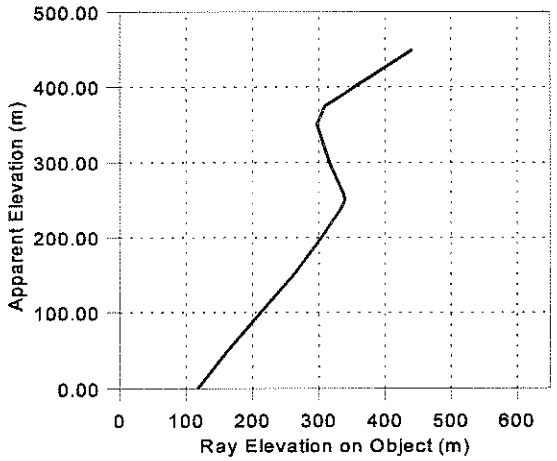


Fig. 4.1 Example of a Transfer Characteristic

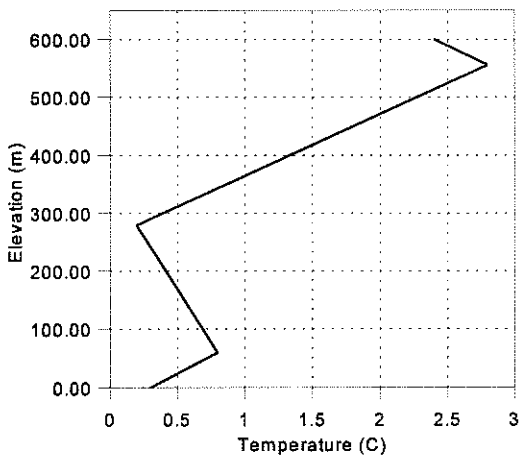


Fig. 4.2 Example of a Temperature Profile

An example of a mirage over Somerset Island is used to illustrate the transfer characteristic mapping procedure. Elevations on the original non-distorted image (Fig. 4.3a) of Somerset Island are compared with the distorted mirage image (Fig. 4.3b). Corresponding

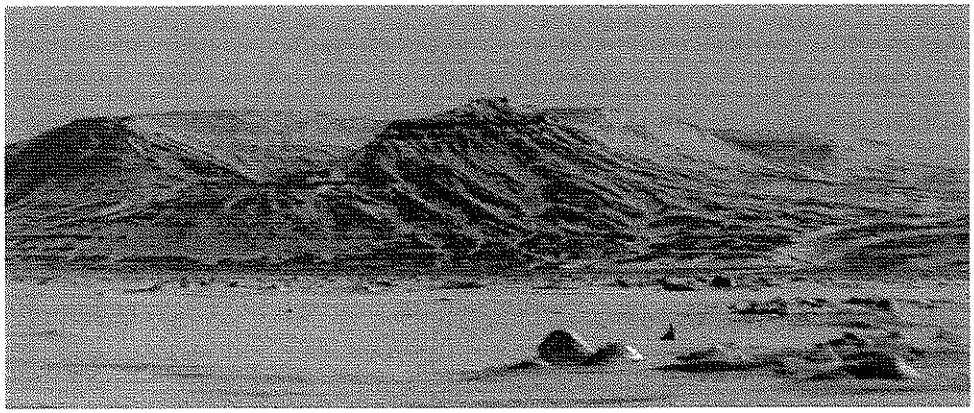


Fig. 4.3a Original Image

Fig. 4.3b Mirage Image

points on the original image of known elevation are mapped to points on the distorted image. The x-axis on the transfer characteristic (Fig. 4.1) shows the actual elevations of the landform, while the y-axis gives the apparent elevations of the mirage. The transfer characteristic explains the way the atmosphere bends light to create the mirage.

This chapter will detail the atmospheric models that have been developed to simulate mirages. Limitations of the flat atmosphere model in simulating some mirages are discussed, and a sloped atmosphere model is developed to overcome these limitations.

4.2 The Flat Atmosphere Model

The flat atmosphere is a model that allows the simulation of mirages in an atmosphere where all temperature levels or isotherms are parallel to the earth's surface. This section describes how the flat atmosphere is modelled, and how a mirage is simulated in this atmosphere. Multiple flat atmospheres over a horizontal distance are also discussed.

4.2.1 The Basic Model

The model developed below allows the tracing of light rays through an atmosphere with a given temperature profile that assumes horizontal isotherms. The atmosphere can be represented by a series of horizontal layers where the temperature either increases or decreases at a constant rate within the layer. Fig. 4.4 illustrates an atmosphere where discontinuities or changes in the slope of the temperature profile define the layer boundaries. Rays of light will bend at a constant rate through each individual layer.

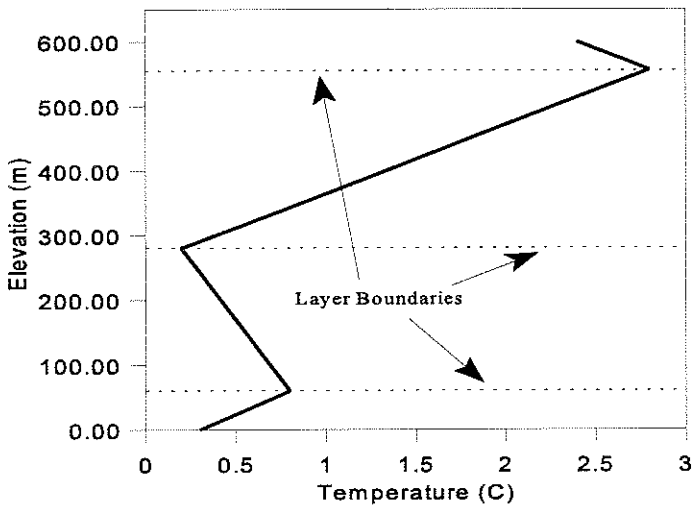


Fig. 4.4 A Temperature Profile with Layer Boundaries

To simplify model calculations, the circular arc of the earth's surface and those of the

layer boundaries are approximated by parabolas [Leh1]. In the xz coordinates illustrated in

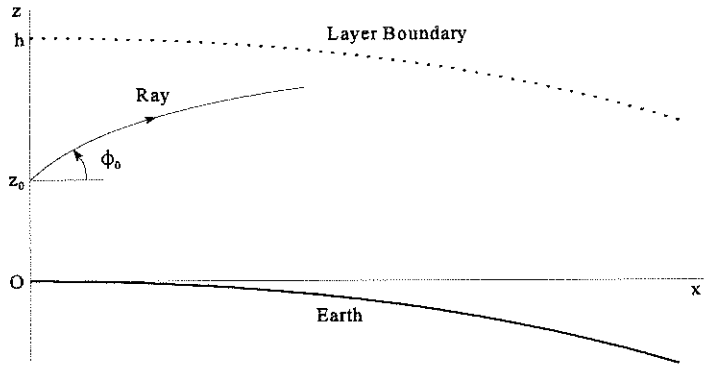


Fig. 4.5 Flat Atmosphere Model [Leh1]

Fig. 4.5, the earth surface is represented by the following parabolic equation:

$$z = -\frac{x^2}{2R_E} \tag{4.1}$$

where R_E represents the radius of the earth. Layer boundaries of the atmosphere can be represented by adding an elevation, h, to each successive surface as follows:

$$z = -\frac{x^2}{2(R_E + h)} + h \tag{4.2}$$

Since h is small compared to R_E, Eqn. 4.2 can be simplified as follows:

$$z = -\frac{x^2}{2R_E} + h \tag{4.3}$$

Lehn explains that the parabolic surfaces and this approximation lead to the same results as the more complicated circular arc model [Leh1].

4.2.2 Ray Tracing in the Flat Atmosphere Model

A light ray travelling through the atmosphere described above can be represented by the following equation:

$$n \sin\theta = A \tag{4.4}$$

where n is the refractive index of the air layer, θ is the angle between the ray and the vertical, and A is a constant for the ray. The ray's curvature (κ_{ray}) is calculated for the xz coordinate system shown in Fig. 4.5 [Leh1] as follows:

$$\kappa_{ray} = -\frac{\sin\theta}{n}\frac{dn}{dz} \tag{4.5}$$

where a positive curvature represents concavity towards the earth.

Since the refractive index depends on temperature, n can also be expressed in terms of temperature as follows:

$$n = 1 + \varepsilon\rho = 1 + \frac{\varepsilon\beta p}{T} \tag{4.6}$$

where ρ is density, p is pressure, T is the temperature at the point being considered, and ϵ, β are constants (226×10^{-6} $m^3 kg^{-1}$ and 3.48×10^{-3} kg K j^{-1}). Substituting Eqn. 4.6 into 4.5 gives a new equation for the curvature as follows:

$$\kappa_{ray} = \frac{\varepsilon\rho}{(1+\varepsilon\rho)T}\sin\theta\left[\frac{dT}{dz} + \beta g\right] \tag{4.7}$$

where g is the acceleration due to gravity [Leh1].

The equation representing the curvature of the ray can be further simplified by substituting the average temperature of each layer for T, and assuming that ρ, g and $\sin\theta$ are constant within each layer. In other words, κ_{ray} can be considered a constant within the layer. Tracing a ray with respect to the xz plane within a layer can now be expressed as follows:

$$z = -\frac{x^2}{2r} + x\tan\varphi_0 + z_0 \tag{4.8}$$

where the value r is equal to $1/\kappa_{ray}$ in the equation, and ϕ_0 is the angle of the ray with the horizontal (Fig. 4.6). The starting point of the ray is x=0 and $z=z_0$ where z_0 is the initial

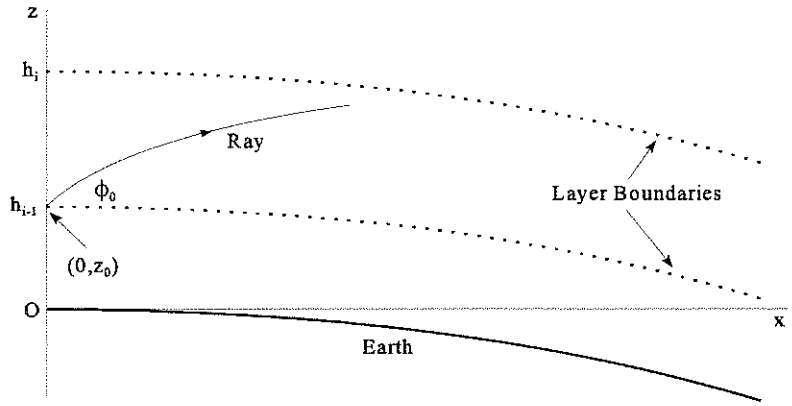


Fig. 4.6 Tracing a Ray in a Concentric Shell Atmosphere [Leh1]

elevation of the ray which is usually the height of the observer's eye above the ground.

Equation 4.8 gives the location of a ray in the xz coordinate system. As illustrated in Fig. 4.6, the xz coordinate system is only accurate over short distances due to the curvature of the earth. To compensate, the effective curvature of the earth, κ_{earth}, can be subtracted from the curvature of the ray, κ_{ray}. As a result, the value of r in Eqn. 4.8 can be replaced with the following:

$$r = \frac{1}{\kappa_{effective}} \quad where \quad \kappa_{effective} = \kappa_{ray} - \kappa_{earth}.$$

The same can be achieved by subtracting the equation for the earth's surface (Eqn. 4.1) from the equation for the ray (Eqn. 4.8).

If the average temperature for each layer is known, an iterative process can be used to trace a light ray from an observer to an object. Starting from z_0, layer intersection locations are calculated by equating and solving Eqns. 4.3 and 4.8. The process starts by changing a ray between two layers at elevations h_i and h_{i-1} a given angle ϕ. If ϕ_0 is positive, assume $h = h_i$, but if no solution exists, the ray must curve and intersect the bottom of the

layer and $h = h_{i-1}$. For negative ϕ_o the opposite process is performed. Ray intersections with the layer boundaries continue to be calculated until the ray reaches the target. The ray trace or path is produced by connecting the intersections with straight line segments. A more detailed discussion of the process and the error involved is given by Lehn [Leh1].

4.2.3 Simulating a Mirage

The previous section discussed a method for tracing rays through an atmosphere with parabolically curved horizontal temperature layers. Using this ray tracing method, it is now possible to simulate mirages. The model or methodology can be applied in two ways. The first begins with a known atmosphere from which a mirage is produced. The second starts with a mirage of known characteristics from which the structure of the atmosphere is inferred. The process of simulating a mirage from a known atmosphere is the simplest and will be discussed first.

If the temperature profile of the atmosphere is known, a parabolic layered model of the atmosphere can be constructed. Suitable angles must be chosen for the rays that leave the observer and strike the object. These angles are usually within +/- 15 arc minutes of the horizontal for the distance over which mirages occur. The height of the observer's eye and the distance to the target object must also be known. Fig. 4.7 illustrates the example

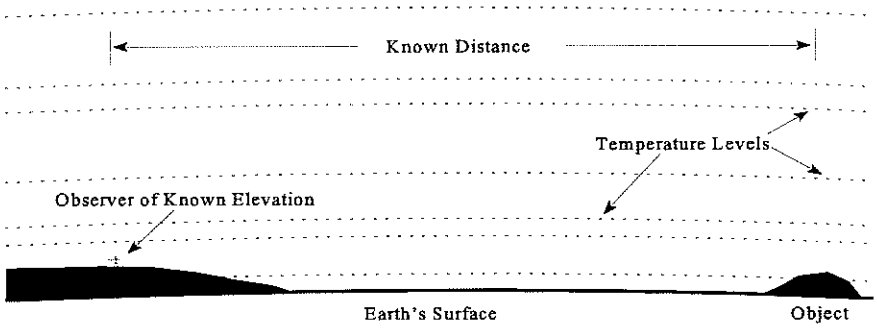


Fig. 4.7 Mirage Simulation in a Atmosphere with Known Temperatures

described above. Light rays leaving the observer at the chosen angle can be traced to the object using the process described in the previous section. Eventually, a suitable number of rays will have been traced as shown in Fig. 4.8.

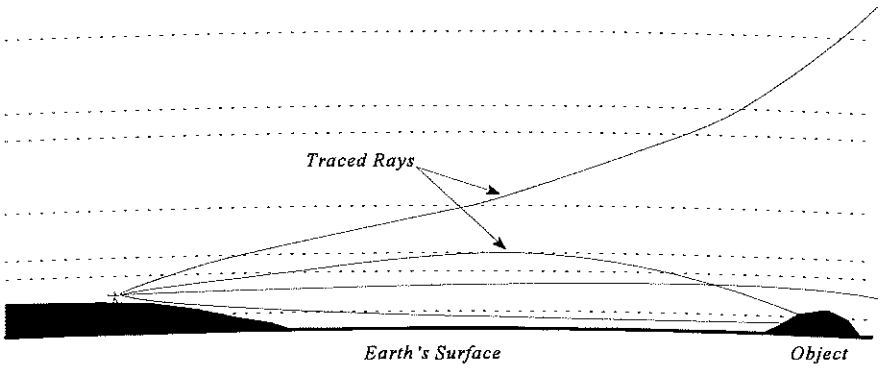


Fig. 4.8 Mirage Simulation Procedure After Ray Tracing

Once the ray tracing process is complete, a transfer characteristic is generated. For each ray striking the target, an apparent elevation of the object can be calculated as follows:

$$Elevation = ObjectDistance \times \tan(RayAngle\,at\,Observer) + ObserverElevation \quad (4.10)$$

Each ray trace from the observer to the object will yield a pair of apparent and actual elevations. This process produces a transfer characteristic such as the one shown earlier in Fig. 4.3.

Once the transfer characteristic has been generated, simulating the mirage is a simple process involving mapping elevations of the original image into a mirage. This can be done by digitizing the original image and mapping rows of pixels at specific elevations to their apparent elevations as determined by the transfer characteristic. The original image of Somerset Island (Fig. 4.3a) is remapped using the transfer characteristic in Fig. 4.1 to yield

the mirage shown in Fig 4.9.

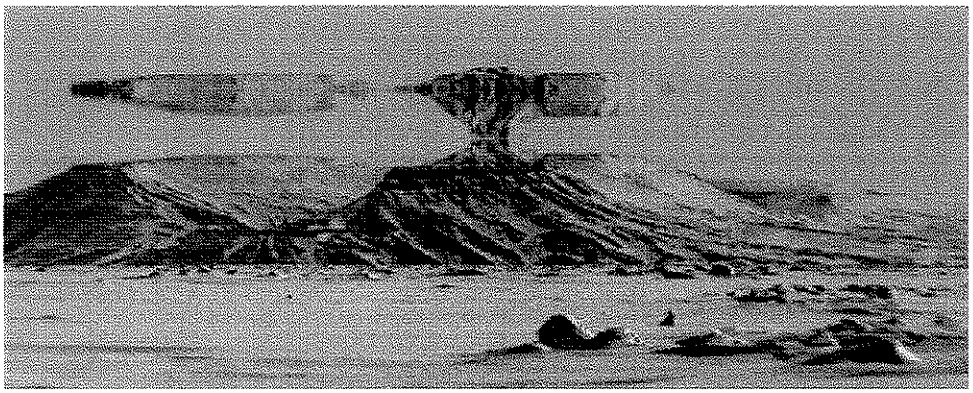


Fig. 4.9 Simulated Mirage Image

This process is relatively simple in comparison to the opposite process of determining a temperature profile from a known transfer characteristic. To determine the temperature profile we begin with a guess of what the profile might look like. Usually the guess is anchored at the surface by a temperature reading taken by the observer. This gives us at least one correct level in the profile. Using our first-guess profile, the mirage is simulated by following the process described above. The resulting transfer characteristic is then compared to the known one and differences between the two are noted. The temperature profile is adjusted and the mirage is simulated once again. The process continues until the simulated transfer characteristic closely matches the known one, and the resulting temperature profile is assumed to be the actual profile of the atmosphere. A more systematic approach of determining a temperature profile from a known transfer characteristic is presented by W. Lehn [Leh2].

4.2.4 Multiple Atmospheres

The discussion presented above assumed a single horizontally uniform atmosphere. Temperature levels remained the same from the observer to the target object. In reality, the

temperature levels of an atmosphere are likely to vary somewhat. A multiple atmosphere model provides a method to account for horizontally changing temperature layers. Figure 4.10 shows an example of a multiple atmosphere. This example utilizes two different

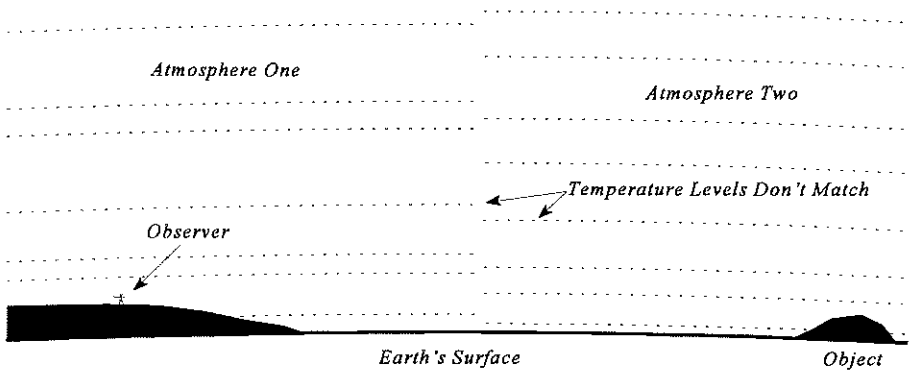


Fig. 4.10 Multiple Atmosphere

atmospheres, but it is possible to have as many atmospheres as required. The example is somewhat crude for simulation purposes because there is a large jump in temperature at the transition between the two atmospheres. Additional layers or additional atmospheres could be used to smooth the transition regions.

To trace a ray in a multiple atmosphere, the ray is traced in the first atmosphere until it reaches the boundary between it and the next. The ray angle and elevation are recorded and used as the starting point in the next atmosphere. Once the rays are traced, the simulation is completed as that described for the single atmosphere model.

4.3 The Wavy Atmosphere Model

Mirages exhibiting periodicity or regular changes over time have been observed on occasion. W. Lehn and W. Silvester have shown that a sequence of mirages that has a periodicity can be modelled with atmospheric gravity waves [Leh5][Sil1]. A gravity wave

or buoyancy wave is a periodic disturbance that propagates through the atmosphere. It has been shown that gravity wave frequencies are low enough to fit mirage observations. These waves are normally small scale features lasting only minutes to tens of minutes [Leh5].

This section will present the atmospheric model for gravity waves and describe how a periodic sequence mirage is simulated.

4.3.1 Atmospheric Model

The development of the gravity wave atmospheric model begins with the flat atmosphere modelled by the concentric parabolic arcs as described in Sec. 4.2.1. As with the concentric arc model, horizontal variations from the observer to the target object are considered negligible. Variations in other factors during a mirage sequence, such as pressure and temperature are also considered constant compared to the more rapid changes caused by the gravity wave. These assumptions allow the waves to be modelled as propagations in the two dimensional xz plane. The following differential equations describe the gravity wave perturbations in the plane:

$$\frac{\delta u}{\delta t} = -\frac{1}{\rho_0}\frac{\delta p}{\delta x} \tag{4.11}$$

$$\frac{\delta w}{\delta t} = -\frac{1}{\rho_0}\frac{\delta p}{\delta z} - g\frac{\rho}{\rho_0} \tag{4.12}$$

$$\frac{\delta p}{\delta t} + w\frac{\delta \rho_0}{\delta z} = 0, \qquad \frac{\delta u}{\delta x} + \frac{\delta w}{\delta z} = 0 \tag{4.13}$$

where u, w, are the x and z velocity perturbations from equilibrium, ρ and p are the density and pressure perturbations, ρ_0 is the equilibrium density, and g is the acceleration due of gravity [Leh5].

The equations of motion for u, w, and p are satisfied by the wave solutions as follows:

$$w = w_z e^{\pm i(kx - \omega t)} \tag{4.14}$$

$$p = p_z e^{\pm i(kx - \omega t)} \tag{4.15}$$

where w_z and p_z are functions of z only as follows:

$$e^{\frac{1}{2}\alpha z}[Ae^{i\eta z} + Be^{-i\eta z}] \tag{4.16}$$

The spatial frequency η is related to the Vaisala-Brunt frequency N by

$$\eta^2 = \frac{k^2}{\omega^2}(N^2 - \omega^2) - \Gamma^2 \tag{4.17}$$

where

$$N^2 = \frac{g}{T}\left(\frac{g}{c_p} + \frac{dT}{dz}\right), \tag{4.18}$$

and the function Γ is given by

$$\Gamma = -\frac{N^2}{2g} + \frac{g}{2c_s^2}. \tag{4.19}$$

The parameter α, which is given by

$$\alpha = -\frac{1}{\rho_0}\frac{\delta\rho_0}{\delta z}, \tag{4.20}$$

is assumed to be constant since it is used as an exponent and $1/\alpha$ is small in the lower levels of the atmosphere. The values c_s and c_p are the speed of sound and the specific heat of air at constant pressure and temperature respectively [Leh5].

An atmosphere could have an infinite number of different gravity waves based on changes in temperature or pressure gradient. Nevertheless, a four layer model is sufficient to model the inversions required for superior mirages [Leh5][Sil1]. Based on the temperature profile, the atmosphere is divided into four layers each with a constant temperature gradient. Since all four layers have the same x displacement and time dependence, the dispersion equation is found by applying continuity on w_z and $\delta w_z/\delta z$ at the boundaries. The solution to these differential equations gives a series of ω, k pairs. W. Lehn and W. Silvester chose to select only one k value per ω value to simplify the problem because a linear combination of different k values creates an overly complex ray tracing problem. Selecting a single pair of ω, k values means that the amplitude functions of w_z for the four layers are constant multiples of each other [Leh5]. Integration with respect to t gives the vertical displacement of the isotherms from equilibrium as follows:

$$z = -\frac{1}{\omega} e^{\frac{\alpha z}{2}} W_z \sin(kx - \omega t) \tag{4.21}$$

where

$$\alpha = \frac{1}{T}(\beta g + \frac{dT}{dz}). \tag{4.22}$$

The displacement equations for each of the isotherms is given by:

$$z_i = -\frac{1}{\omega} e^{\frac{\alpha_i h_i}{2}} W_{z_i} \sin(kx - \omega t) + h_i \tag{4.23}$$

where h_i is the elevation from the temperature profile, i is the wave boundary number, and W_z depends on the individual layer of the 4 layer model [Sil1]. A simple diagram of the resulting atmosphere is shown in Fig. 4.11.

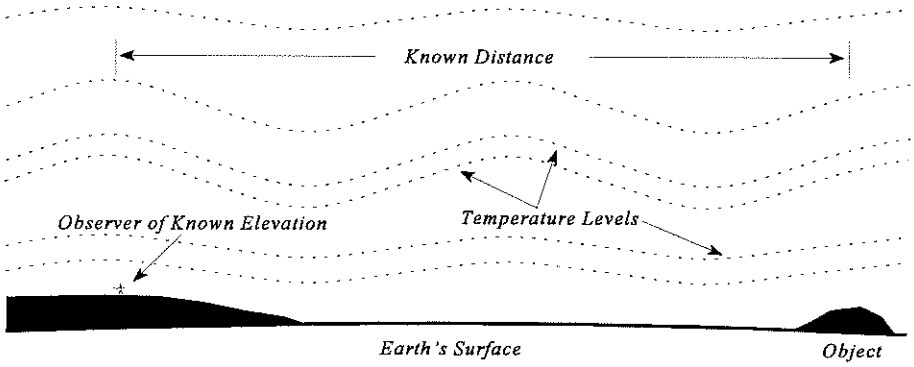


Fig. 4.11 Atmosphere with Gravity Waves

4.3.2 Ray Tracing in a Gravity Wave Atmosphere

Ray tracing through a wavy atmosphere follows the same process used for the flat atmosphere. In the flat model the equation for the ray (Eqn. 4.8) and the equation for the layer boundary (Eqn. 4.3) were equated to each other and solved. The solution gave the boundary intersections for the trace. Equation 4.22 is used instead of Eqn. 4.3 for the gravity wave model.

4.3.3 Simulating a Mirage

To simulate a mirage with gravity waves, the profile of the atmosphere must be known. Consequently, a representative mirage in the wave sequence of mirages must be chosen. The temperature profile for the representative mirage is determined using another mirage model such as the flat model. This temperature profile is then used for the gravity wave mirage simulation.

Since the gravity wave model is limited to four layers, the profile is examined and divided into four appropriate layers. For example, the temperature profile in Fig. 4.12 (solid line) is broken into the four sections resulting in a profile (dashed line) with a constant lapse

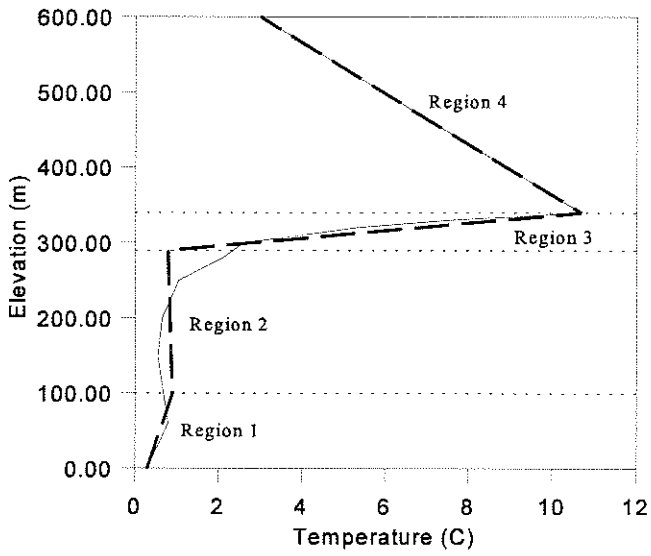


Fig. 4.12 Four Layer Model

rate in each layer. The temperatures and elevations at the boundaries are used in solving the differential wave equations (4.13).

Solving Eqn. 4.13 gives values for W_z in each of the four regions shown in Fig. 4.12. Once values for ω and k are selected to represent the time period and wave number, Eqn. 4.22 can be solved. Time, t, is set to zero in Eqn. 4.22 to obtain the first mirage in the sequence that repeats when $t=2\pi/\omega$. All intervening mirages occur between these two times.

A more detailed discussion on choosing ω, k and calculating W_z values is provided by W. Lehn and W. Silvester [Leh5][Sil1].

4.4 Inadequacies in Current Simulation Models

The mirage simulation models described in the previous sections do not always provide an adequate solution to the problem. For example, the mirage shown in Fig. 4.2b cannot be simulated properly with a flat atmosphere model. The model requires a temperature profile with 20-30 degree changes over a few metres of elevation. These strong

lapse rates are not realistic in the atmosphere. Sometimes a mirage cannot be simulated at all with this model. In these cases, a limitation in the model causes rays to oscillate out of control no matter what temperature gradients are chosen. These problems occur when mirages are observed over long distances or when they are observed over rough terrain or over changing surfaces such as land to ocean/ice. Since the wavy atmosphere model is based on the flat model, it has the same inadequacies and problems.

Some of these mirage simulation problems can be overcome with the multiple atmosphere model described in Sec. 4.2.4, but often even it fails. Some of the long range mirages may involve the Novaya Zemlya effect, but a few mirages will require a new model to be adequately simulated. The new model must also realistically represent the earth's atmosphere. Section 4.5 will present a new model that allows thermal layers of the atmosphere to slope. The next chapter will show that the mirage in Fig. 4.2b can be simulated with this new model.

4.5 The Sloped Atmosphere Model

The sloped atmosphere mirage model can be developed in a number of ways. One possibility is to extend the multiple atmosphere to a large number of atmospheres to represent any slope or curve in the atmosphere. This solution would be computationally intensive because each individual atmosphere requires its own temperature profile. Another possibility is to develop new equations to define temperature layers with a slope. This method would require the mathematical development of a new model and new implementation in software. The third option is to simply tilt the flat atmosphere. This section describes how this tilting is accomplished for both a single atmosphere and for multiple atmospheres.

4.5.1 Atmospheric Model

The flat and wavy atmospheres described in previous sections are depicted in Fig. 4.7 and Fig. 4.11. The new sloped atmosphere model being considered is shown below in Fig. 4.13. Note that the temperature levels are no longer parallel to the earth's surface. A gravity wave atmosphere could similarly be tilted so that its wave axis was no longer parallel

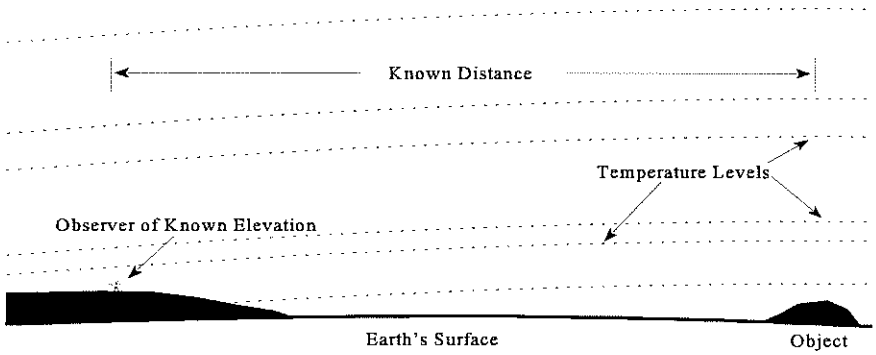


Fig. 4.13 Mirage Simulation in a Sloped Atmosphere

to the earth's surface. Multiple region atmospheres are also possible with any combination of flat, sloped or gravity wave regions.

To slope an atmosphere as shown in Fig. 4.14, the height of the observer's eye is increased and the initial ray angle is reduced. Figure 4.14a illustrates a flat atmosphere; 4.14b, a raised observer; and 4.14c the resultant tilting of the isotherms when the observer's height is adjusted with respect to the earth's surface.

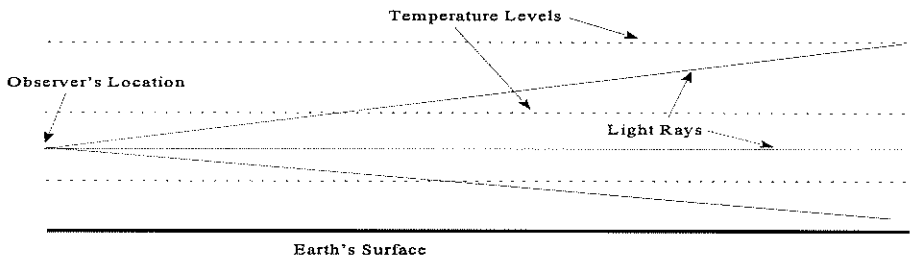


Fig. 4.14a Flat Atmosphere Model

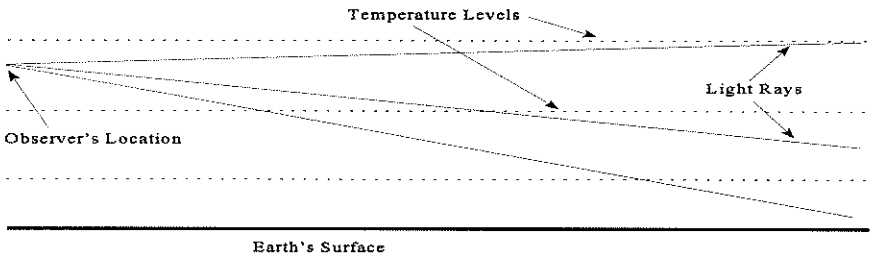


Fig. 4.14b Raise Eye and Lowered Ray Angles in Flat Model

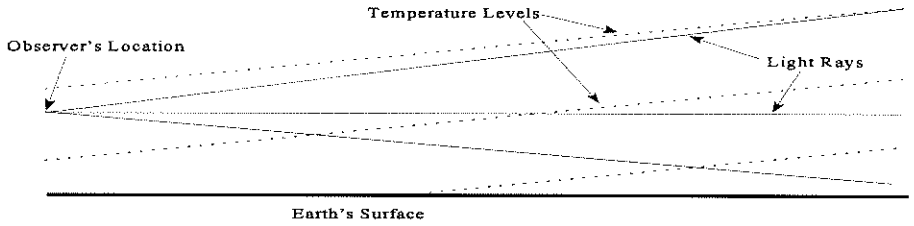


Fig. 4.14c Sloped Atmosphere Model

4.5.2 Ray Tracing in a Sloped Atmosphere

Unlike the gravity wave modification of the flat atmosphere model, the sloped atmosphere model changes the initial conditions of the ray. As a result, this model uses the same equations (Eqn. 4.3 and 4.22) as the previous models. To trace a ray in a sloped atmosphere we simply have to adjust the initial conditions of the ray.

To describe the sloping atmosphere model, an example of a two region multiple

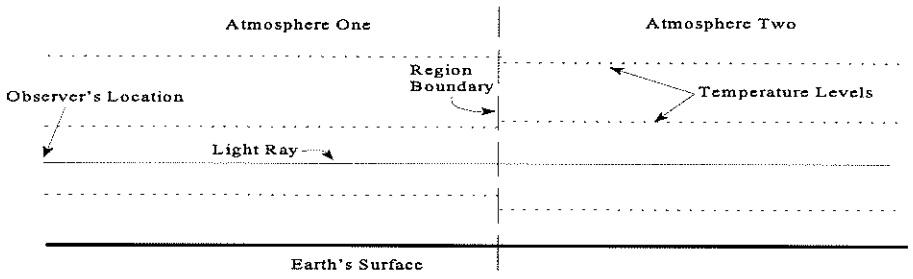


Fig. 4.15 Initial Two Region Atmosphere

atmosphere will be used. This example will then be used to later explain other possible sloping regions. Figure 4.15 shows the initial two atmospheres without any slope, and a ray is traced through region one using the process described in the flat atmosphere model. When the ray reaches the boundary with region two, its angle and elevation are adjusted for the slope of atmosphere two. The new angle and elevation in this region are given by the following equations:

$$New\,Angle = Old\,Angle - Slope\,of\,Atmosphere \tag{4.24}$$

$$New\,Elevation = Old\,Elevation + Width\,of\,Region \times \tan(Slope\,of\,Atmosphere) \tag{4.25}$$

The ray is then traced through the flat temperature layers of region two (Fig. 4.16) until it reaches the target. This in effect creates a sloped atmosphere in region two.

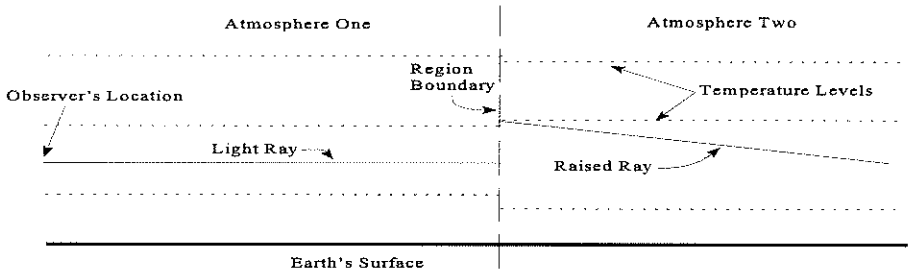


Fig. 4.16 Light Ray is Raised and then Traced

If the traced ray and temperature levels in region two are reduced in elevation at the region boundary using Eqn. 4.24, the model can now be illustrated by Fig. 4.17. The temperature levels in region two are now sloped and the ray has been traced.

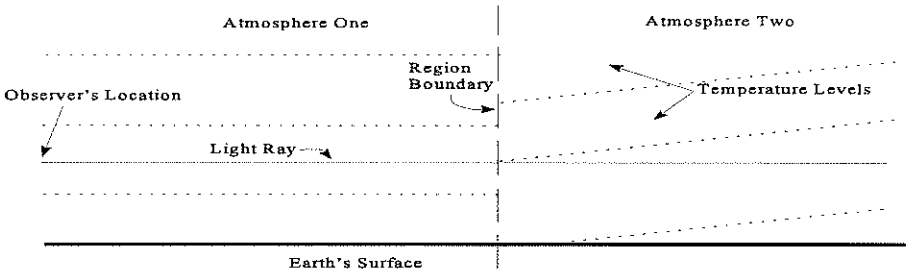


Fig. 4.17 Two Region Atmosphere with Sloped Second Region

This example describes two atmospheres, but the single atmosphere is just the general case of a multiple atmosphere with only one region. To trace a ray through a single atmosphere, the initial ray elevation and angle at the observer are adjusted as described above.

4.5.3 Simulating the Mirage

To simulate a mirage in the sloped atmosphere model, we must trace rays from the observer to the target using the method described in Sec. 4.5.2. Since this model only modifies the ray being traced, any combinations of flat or gravity wave atmospheres can be sloped at any angle. Thus, the sloped atmosphere model modifies but does not replace the previous models.

Chapter 5 describes a software simulation program that can simulate all of the models presented in this chapter. In fact, the software allows the simulations of mirages in a single or multiple atmosphere with any combination of flat, sloped, gravity waves, or sloped gravity wave atmospheres.

CHAPTER 5

Experimental Results

5.1 Software Used in Research

A number of software programs were used in simulating the mirages in this thesis. They are listed as follows:

(1) *maketc* - This is a program used for the construction of a transfer characteristic from a mirage image. The program takes a mirage image and an original image to allow the user to map points from the mirage to the original. The program then generates and saves an approximate transfer characteristic as it maps apparent ray elevations on the mirage to actual elevations of the original.

Developed By: W. Silvester (1991)

(2) *edittc* - This program allows a user to edit the transfer characteristic generated by *maketc*. It allows fine adjustments to the transfer characteristic that are not possible with the *maketc* program.

Developed By: W. Silvester (1991)

(3) *steer* - This is a flat atmosphere ray tracing program. The program allows a user to load a temperature profile into the software and trace a number of light rays through the atmosphere as defined by the profile. The program graphs these traced rays and generates a transfer characteristic. It also allows the data for the graphs to be saved.

Developed By: W. Lehn (1991)

(4) *makeatm* - This is a utility program that creates an *.atm* file for use with the *hexray* ray

tracing program. It uses a data file that specifies the four-layer approximations of a temperature profile for generating an atmosphere model with gravity waves. The output *.atm* file contains the magnitude of the waves for each temperature layer.

Developed By: W. Silvester (1992)

(5) *hexray* - This is a flat, wavy, and multiple atmosphere ray tracing program. The program inputs either a temperature profile (*.tp*), an atmosphere region file (*.arf*), or an atmosphere file (*.atm*) to define a flat, multiple, or wavy atmosphere respectively. The program then traces rays through the defined atmosphere and outputs a file containing ray data. If desired, the program also saves the transfer characteristic generated by the ray trace.

Developed By: W. Silvester (1992)

(6) *multiray* - This is a revision of the *hexray* program allowing sloped atmospheres to be traced in addition to the other atmospheres. The revision also has the ability of graphing the rays and transfer characteristic on a computer monitor in addition to saving this information.

Developed By: T. Legal (1995) - see Appendix B

(7) *rgbmir* - This program is a colour mirage simulator. The program allows an original colour image to be loaded along with a transfer characteristic. It then converts the actual to apparent elevation mapping to generate a mirage image. This image can be saved as a Windows bitmap for later use. All of the mirage simulations introduced in this chapter were generated using this program.

Developed By: T. Legal (1994) - see Appendix C

These software programs are available in the software addendum to this thesis in both code and executable form.

5.2 General Procedure for Mirage Simulation

This section outlines the general procedures and software routines used in simulating a mirage.

The process begins by generating a transfer characteristic of a mirage image using *maketc*. The *maketc* program is used to map a mirage image against a non-distorted image of the same scene. Once an initial transfer characteristic is produced by the *maketc* program, a simulated mirage image is created using the *rgbmir* software. The simulated mirage and the mirage picture are compared. Differences between the two are adjusted by editing the transfer characteristic using the *edittc* program.

The next step in the procedure is to model an atmosphere with a similar transfer characteristic. The modelled atmosphere will be based on the flat atmosphere, multi-region atmosphere, sloped atmosphere or some combination of the three atmospheres. A model configuration and appropriate temperature regimes are chosen as a starting point. The *multiray* program is used to produce a transfer characteristic based on tracing rays through the defined atmosphere. The resultant transfer characteristic is compared to the mapped transfer characteristic. Changes in the model configuration are made to minimize the difference in the two transfer characteristics.

If the mirage displays a sequence of periodic changes, the process is similar except that the *makeatm* utility is also run to generate an atmosphere with gravity waves. In this process one mirage in the sequence is used to generate a representative mirage for the group. This mirage is mapped and a transfer characteristic is generated as described above. The resulting temperature profile is modelled with four layers and points from these four layer are used as input into the *makeatm* program. The *mulitray* program is then executed with the resulting *.atm* file and a mirage sequence is generated.

5.3 Simulating a Complex Long Range Mirage

A mirage observed from Resolute Bay on June 4, 1994 promised to be an interesting simulation problem because of the long distance from which it was viewed. This section presents simulations of the mirage using a flat atmosphere, a single sloped atmosphere and a multiple region atmosphere.

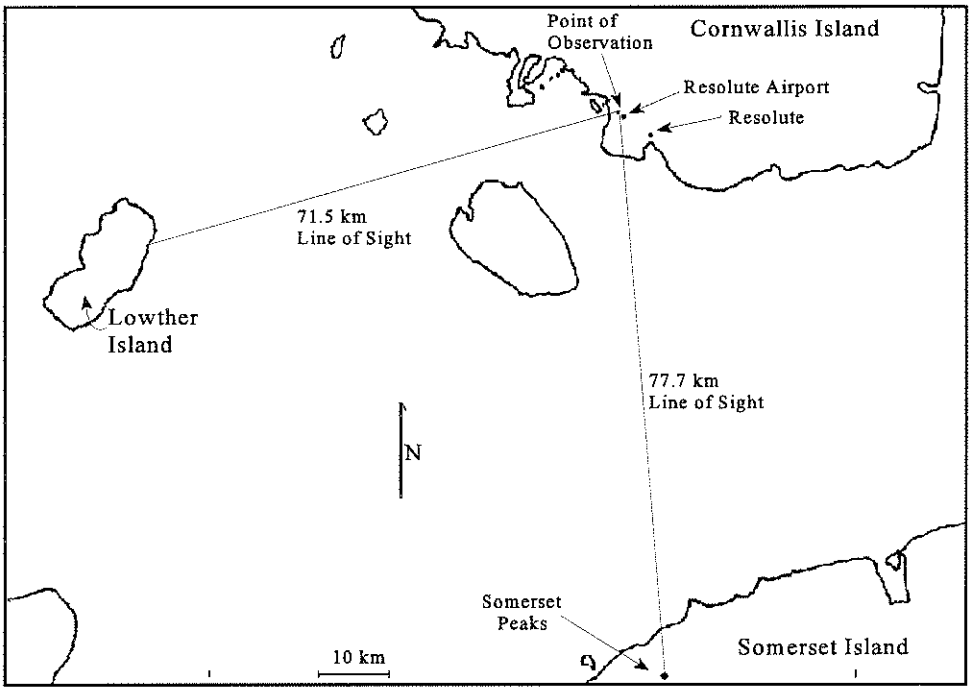


Fig. 5.1 Map of Resolute Bay Area

5.3.1 Generating a Transfer Characteristic

At 07:35 CDT on June 4, 1994, a mirage of Somerset Island (Fig. 5.2) was observed from Resolute Bay, NWT. Somerset Island is located about 78 kilometres south of Resolute Bay in the arctic. It can best be described as an upside-down mountain floating in the air above other mountain peaks. This mirage is interesting because of the large distance from which it was observed, and the fact that islands and other features between Resolute and

Somerset Island showed no evidence of a mirage.

Fig. 5.2 Mirage of Somerset Island at Resolute Bay, NWT

To generate the transfer characteristic, a non-distorted picture of Somerset (Fig. 5.3) was taken from a much closer distance. Since this picture is only a few kilometres from the island, the horizon does not obstruct the view of the base of the mountain. The ice horizon in this photograph is considered to be at zero metres above sea level. The elevation of the peak in the centre of the picture is 340 metres above sea level.

The first step in the simulation process is to generate a transfer characteristic that maps the non-distorted image to the distorted mirage image. The two images (Fig. 5.2 and Fig. 5.3) are loaded into the *maketc* software package and key elevations for both the mirage and non-distorted image are entered into the program. In this example, these elevations are zero metres at the horizon and 340 metres at the mountain peaks. The software then displays

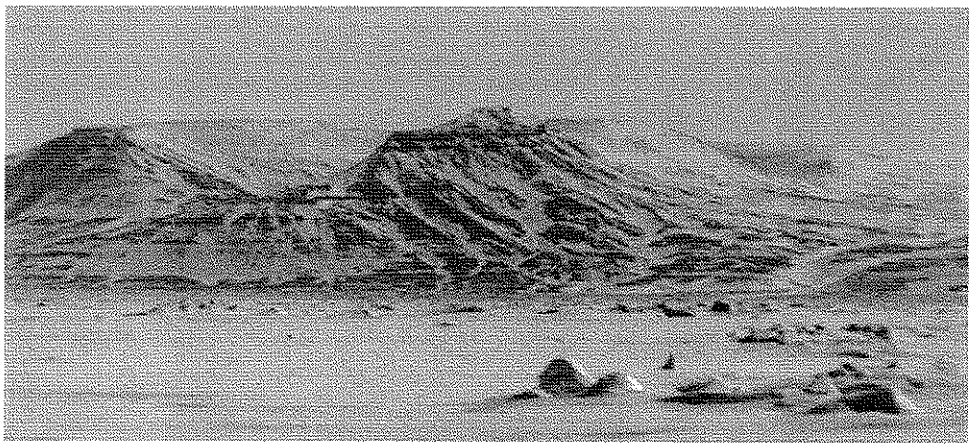


Fig. 5.3 Close-up Photograph of Somerset Island

the two images beside each other and the user maps locations on the original image to the mirage image by clicking on corresponding points with the mouse. After a number of features have been mapped, the mirage's transfer characteristic is generated.

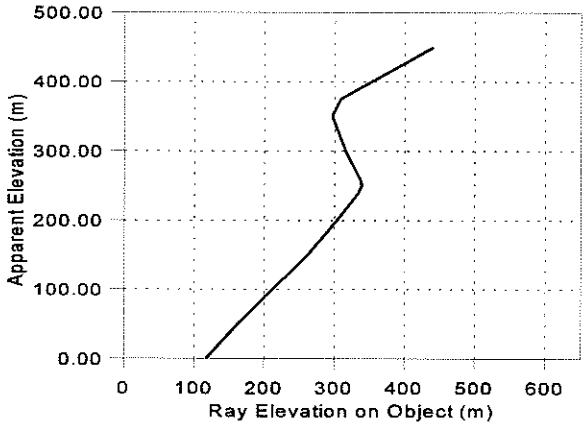


Fig. 5.4 Original Transfer Characteristic (original.tc)

Once a transfer characteristic is produced, the mirage can be simulated. The transfer characteristic and the original non-mirage image (Fig. 5.3) are loaded into the colour mirage simulator software (*rgbmir*). The program is executed and the program produces a simulated mirage image. This mirage simulation is not always perfect on the first attempt. If the simulation does not look right, the transfer characteristic is edited using the *edittc* program and the simulation is done again with the new transfer characteristic. This process continues until a satisfactory simulation is produced. Figure 5.4 shows the transfer characteristic

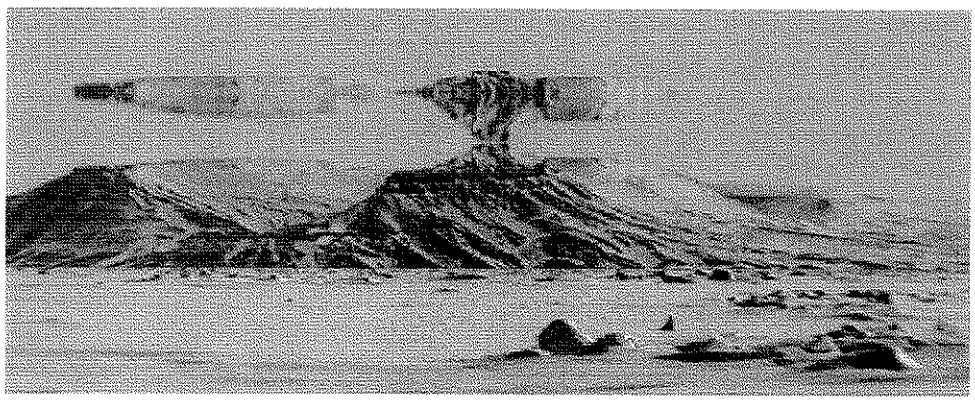


Fig. 5.5 Simulated Mirage of Somerset Island

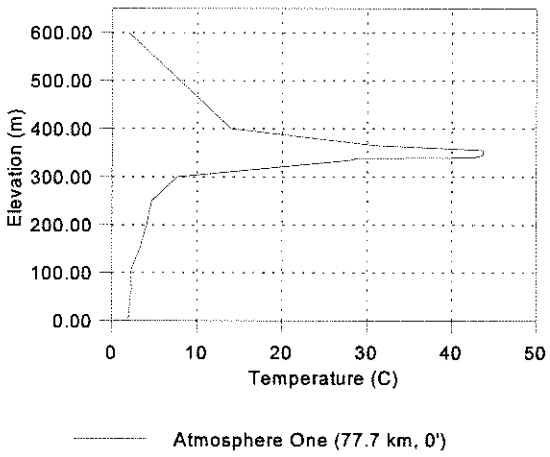
generated and Fig. 5.5 shows the resulting mirage simulation.

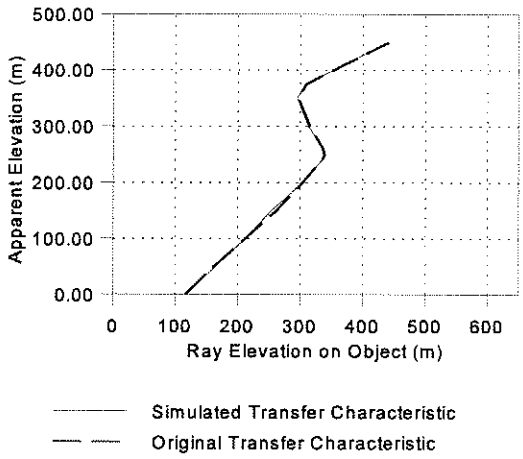
5.3.2 Single Flat Atmosphere Model

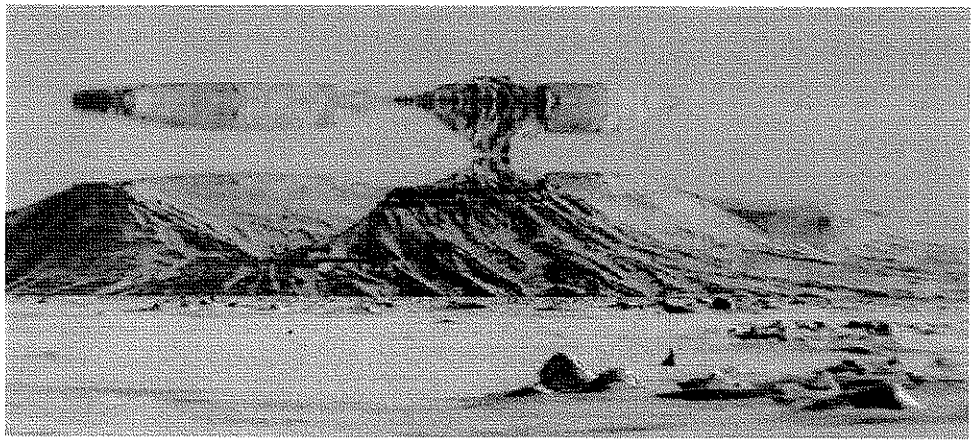
The next step in the simulation process is to model a realistic atmosphere that can produce a similar transfer characteristic to the one shown in Fig. 5.4. A straight flat atmosphere for the whole distance between the observer and the mirage is assumed. As described in Chapter 4, the process of creating a temperature profile with ray tracing is undertaken. The elevation of the observer is 67 metres, the distance from the observer to Somerset Island is 77.7 kilometres, and ray tracing is done in steps of 700 metres. The rays selected start at minus 13.3 arc minutes. This value for the horizon elevation was measured using a theodolite. The first simulation results are shown in Fig. 5.6. The graph in the upper left corner shows the temperature profile required. The second graph shows the transfer characteristic generated by this temperature profile along with the original transfer characteristic shown in Fig. 5.4. The resulting mirage simulation, atmosphere characteristics and ray values are also included.

This simulation shows that the flat atmosphere model fails for this mirage. Although the simulation and transfer characteristic look exactly alike, the temperature profile is unrealistic. The temperature of the atmosphere must increase more than 30°C from the 300-metre level to the 350-metre level in elevation. This is unrealistic in the real atmosphere. Inversions usually involve temperature changes of no more than 15 degrees. Also, the temperature above any inversion cannot decreases any faster then the dry adiabatic lapse rate. Figure 5.6 shows the temperature decreasing 30 degrees over 50 metres of elevation.

The flat atmosphere produces an unrealistic simulation of the mirage.







Temperature Profile (a14.tp)

Atmosphere: Width: 77.7 kilometres Slope: 0 arc minutes

Transfer Characteristic (a14.tc)

Ray Angles: -13.3, -11.09, -8.88, -6.67, -4.46, -2.692, -2.25, -1.808, -0.04, 2.17, 3.275,3.938, 4.38, 4.822, 6.59

Fig. 5.6 Simulation #1 - Single Region Flat Atmosphere

5.3.3 Single Sloped Atmosphere Model

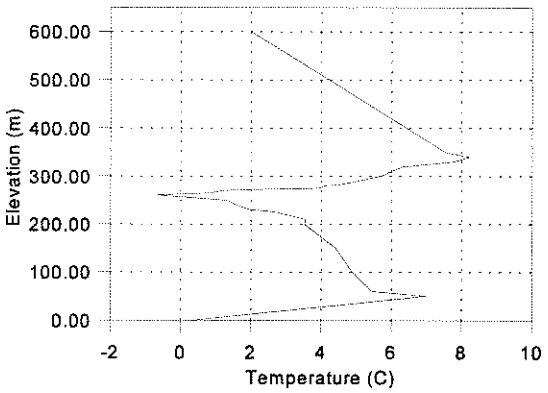
The next simulation assumes a single sloped atmosphere between the observer and Somerset Island. The set up is the same as that used for the flat atmosphere (eye elevation, 67 metres; distance, 77.7 kilometres; step size, 700 metres; and horizon angle minus 13.3 arc minutes).

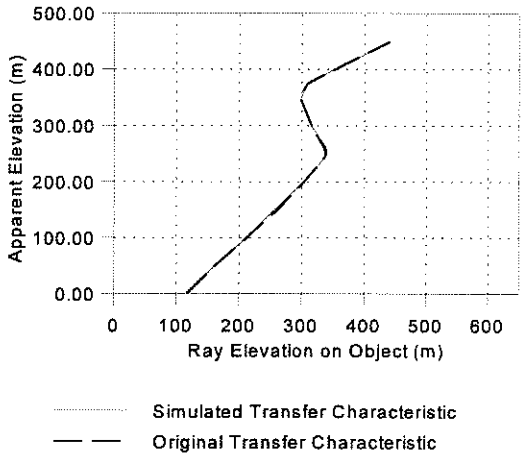
The first slope selected for the atmosphere was five arc minutes. The task of simulating this mirage proved to be very difficult. Small changes in the temperature profile produced large changes in the transfer characteristic. This sensitivity is an unrealistic feature of the real atmosphere since temperatures always vary a small amount even in a thermally stratified atmosphere. As a result, a new slope of ten arc minutes was selected. This slope exhibited the same problems as the five arc minute simulation. A successful simulation was finally achieved with a slope of 11.5 arc minutes.

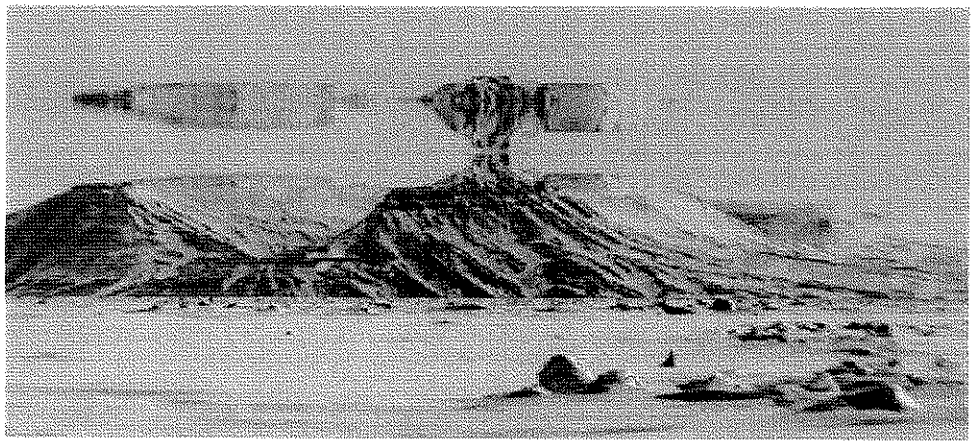
The results from the 11.5 minute simulation are shown in simulation two (Fig. 5.7). The transfer characteristic and the resultant mirage image indicate a good solution. However, the temperature profile is still problematic. It shows two inversion layers at a distance of 200 metres apart. Although this is possible in the real atmosphere, it is not probable. The surface inversion would need significant night time cooling, but Resolute Bay has a midnight sun during the "Arctic Summer" when this picture was taken. The lapse rates above the inversions are also greater than those theoretically allowed.

Another simulation was attempted with a slope of 12.5 arc minutes, but it also failed. The larger slope prevented the rays from being bent significantly, which caused a flattened transfer characteristic.

The single sloped atmosphere like the single flat atmosphere failed to adequately model this mirage.







Temperature Profile (a13.tp)

Atmosphere: Width: 77.7 kilometres Slope: 11.5 arc minutes

Transfer Characteristic (a13.tc)

Ray Angles: -13.3, -11.09, -8.88, -6.67, -4.46, -2.692, -2.25, -1.808, -0.04, 2.17, 3.275,3.938, 4.38, 4.822, 6.59

Fig. 5.7 Simulation #2 - Single Region Sloped Atmosphere

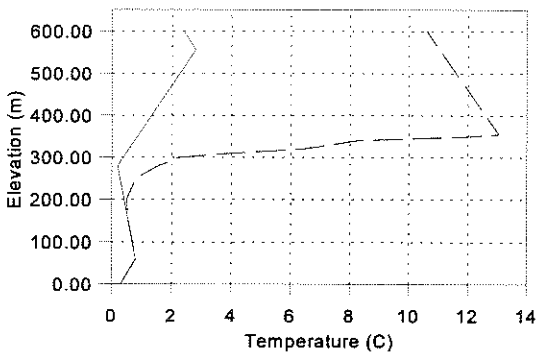
5.3.4 Multiple Sloped Atmosphere Models

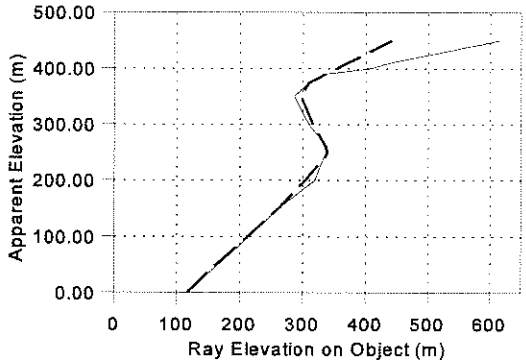
In the previous two sections, single flat and single sloped atmosphere models both failed to adequately simulate the Somerset Island mirage. These results are not surprising because the real atmosphere would not be expected to remain homogeneous over such a long distance. Since the mirage was observed from land across the sea ice to more land, it might be reasonable to postulate that the atmosphere might slope upwards towards the land while remaining relatively flat over the ice. The discussion in Chapter 2 indicates why such a scenario is probable.

The next set of simulations will assume a flat atmosphere from the observer extending over the ice and then an upward sloping atmosphere toward Somerset Island. The simulations begin with a two-region atmosphere. The horizontal distance of the flat atmosphere and the sloped atmospheres are varied, and different slopes for the second region are also explored. As well, a third region is added to test its effect on the simulations. In all of the simulations, the profile measured by Environment Canada at Resolute (a5.tp - Appendix A) near the time of the mirage was used to define the first atmosphere.

Initial Simulation

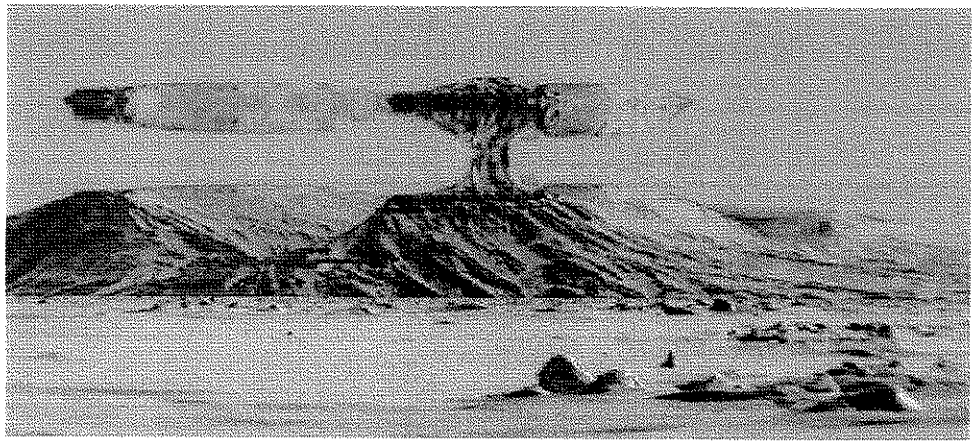
As a starting point, the atmosphere used in the initial multi-atmosphere model (Fig. 5.8) was defined as flat for 35.0 kilometres and sloped at 11.5 arc minutes for the remaining 42.7 kilometres. The other conditions, eye level, step size and horizon angle were the same as the single sloped atmosphere case. The distances, 35.0 kilometres and 42.7 kilometres, were chosen arbitrarily, and the slope of 11.5 arc minutes was chosen because it produced the best results in the single sloped atmosphere. The temperature profile shown in simulation three consists of two parts; region one, defined by the observations, is shown





Atmosphere One (35.0 km, 0')
Atmosphere Two (42.7 km, 11.5')

Simulated Transfer Characteristic
Original Transfer Characteristic



Temperature Profile (a5.tp, a10.tp)

Atmosphere 1: Width: 35.0 kilometres Slope: 0 arc minutes
Atmosphere 2: Width: 42.7 kilometres Slope: 11.5 arc minutes

Transfer Characteristic (a10.tc)

Ray Angles: -13.3, -11.09, -8.88, -6.67, -4.46, -2.692, -2.25, -1.808, -0.04, 2.17, 3.275,3.938, 4.38, 4.822, 6.59

Fig. 5.8 Simulation #3 - Multiple Region Atmosphere

as a solid line and region two is a dashed line. The transfer characteristic and the simulated mirage image show that this atmosphere closely resembles the desired one. It differs only slightly at the higher elevations where a difference is inconsequential. The ray trace that produced the transfer characteristic is illustrated in Fig. 5.9, where the solid lines represent the rays and the short dashed lines represent the slope of the atmosphere.

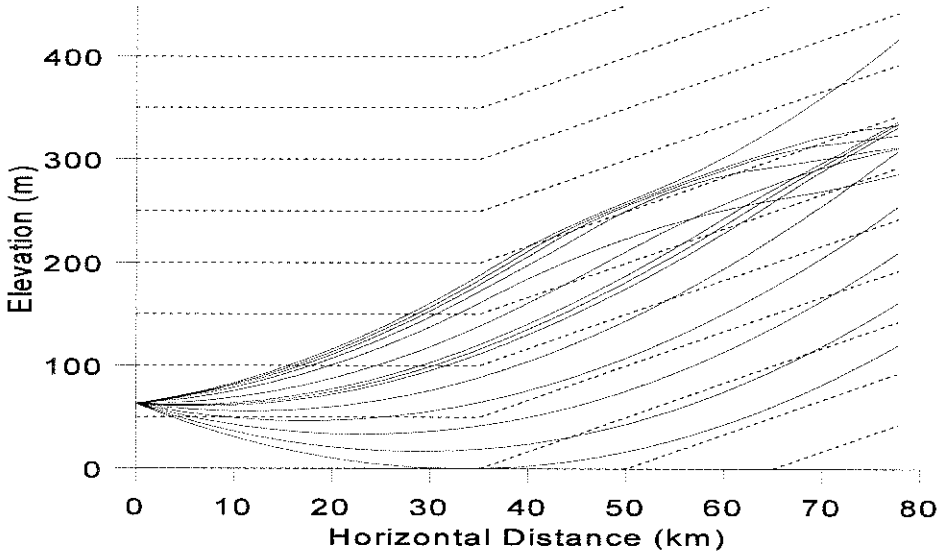
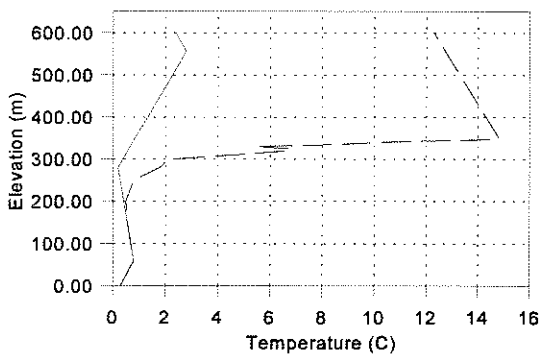
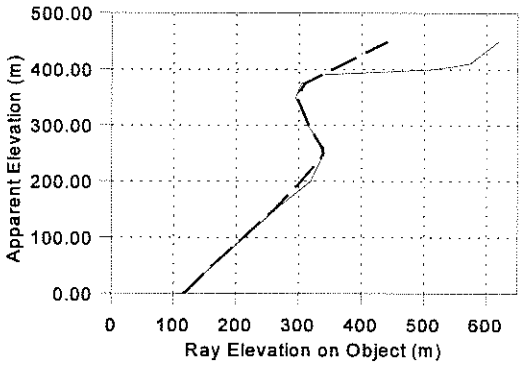


Fig. 5.9 Ray Trace of Two Region Sloped Atmosphere

Size of Regions

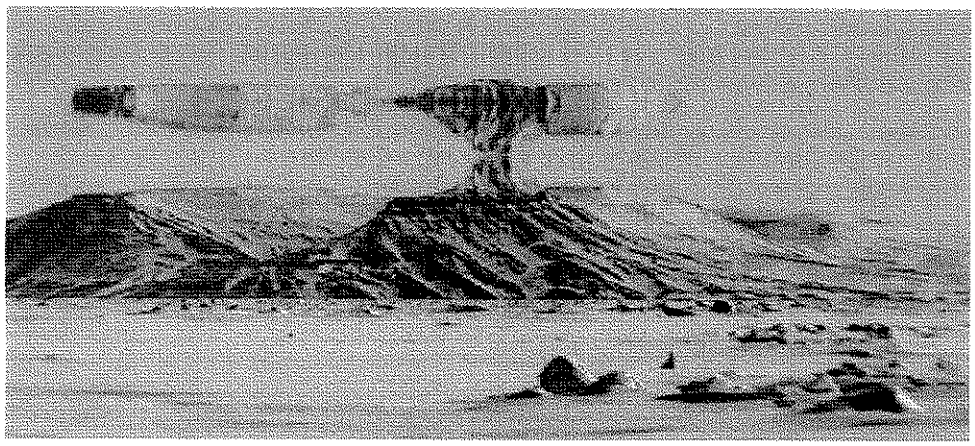
Although the initial simulation using a two-region atmosphere looked reasonably good, more possibilities were explored. The next set of simulations altered the size of the two regions. The first increased the width of the flat region to 39.9 kilometres (Fig. 5.10), and the second shortened it to 30.1 kilometres (Fig. 5.11).





Atmosphere One (39.9 km, 0')
Atmosphere Two (37.8 km, 11.5')

Simulated Transfer Characteristic
Original Transfer Characteristic



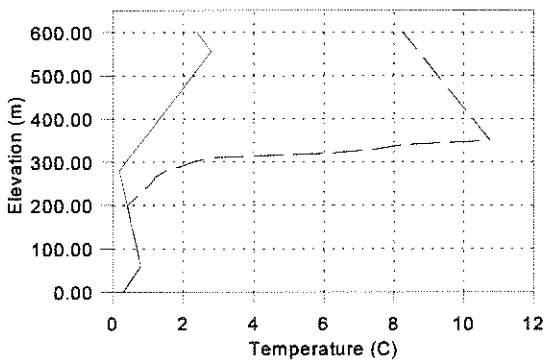
Temperature Profile (a5.tp, a11.tp)

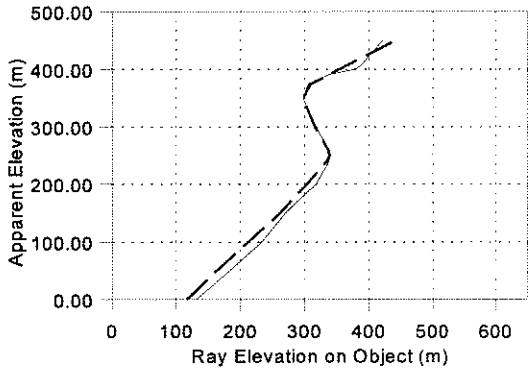
Atmosphere 1: Width: 39.9 kilometres Slope: 0 arc minutes
Atmosphere 2: Width: 37.8 kilometres Slope: 11.5 arc minutes

Transfer Characteristic (a11.tc)

Ray Angles: -13.3, -11.09, -8.88, -6.67, -4.46, -2.692, -2.25, -1.808, -0.04, 2.17, 3.275,3.938, 4.38, 4.822, 6.59

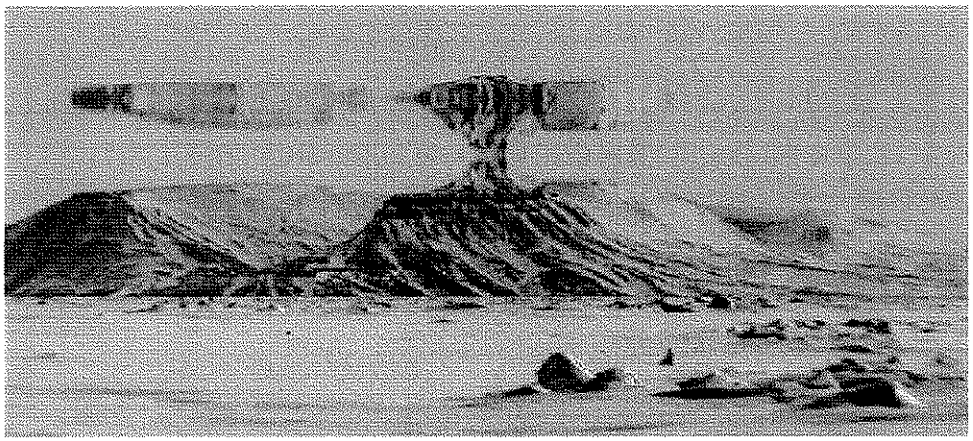
Fig. 5.10 Simulation #4 - New Width for Both Regions (39.9 km - 37.8 km)





Atmosphere One (30.1 km, 0')
Atmosphere Two (47.6 km, 11.5')

Simulated Transfer Characteristic
Original Transfer Characteristic



Temperature Profile (a5.tp, a12.tp)

Atmosphere 1: Width: 30.1 kilometres Slope: 0 arc minutes
Atmosphere 2: Width: 47.6 kilometres Slope: 11.5 arc minutes

Transfer Characteristic (a12.tc)

Ray Angles: -13.3, -11.09, -8.88, -6.67, -4.46, -2.692, -2.25, -1.808, -0.04, 2.17, 3.275,3.938, 4.38, 4.822, 6.59

Fig. 5.11 Simulation #5 - New Width for Both Regions (30.1 km - 47.6 km)

Figure 5.12 shows the temperature profiles of the sloped atmosphere for each of the three simulations. Region one profiles were held constant. Figure 5.13 shows the transfer characteristics for the three simulations.

In both figures, the solid lines represent the first simulation (Simulation #3); the long dashes, the second (Simulation #4); and the short dashes, the third (Simulation #5). The thicker dotted line in Fig. 5.13 is the original transfer characteristic.

Selecting the best simulation of the three is difficult. Overall, it appears that Simulation #5 has the best transfer characteristic, but this is not really the case. Only elevations below the peak height of 340 metres are important in the comparison because atmospheric layers above the peak are not involved in the mirage and are undefined. The transfer characteristic of Simulation #4 with the widest flat region fits the original the best. Simulation #5 with the shortest flat region actually has the worst fit of the three. Fig. 5.12

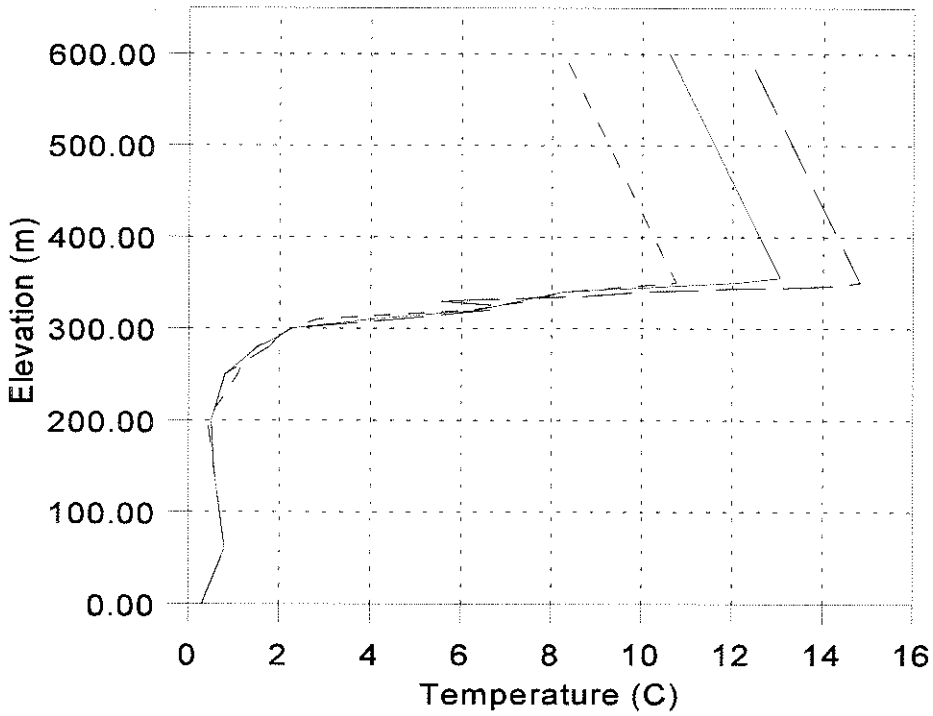


Fig. 5.12 Comparison of Temperature Profiles

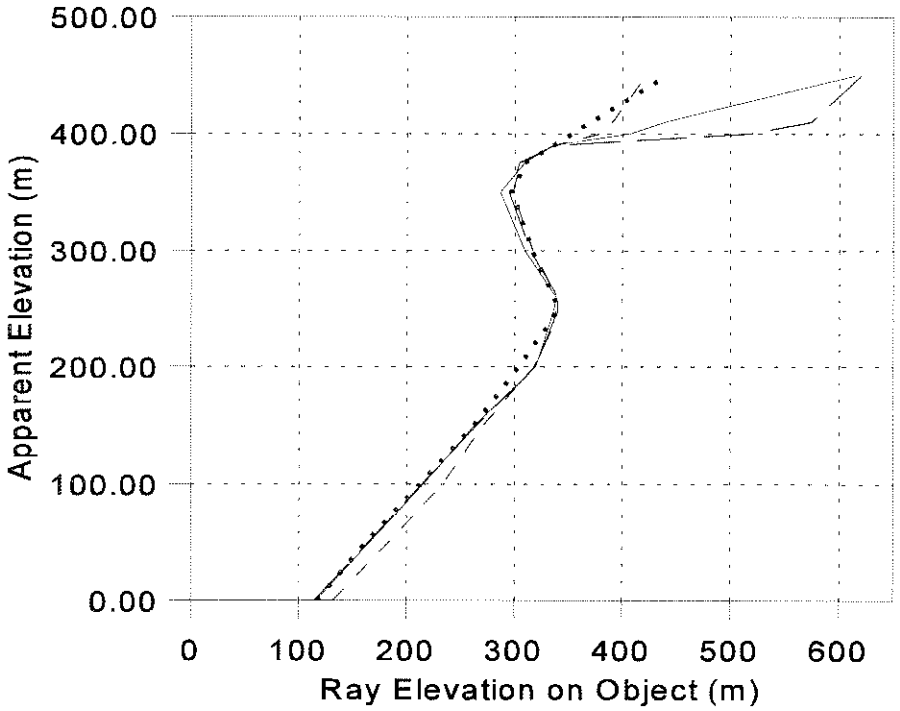


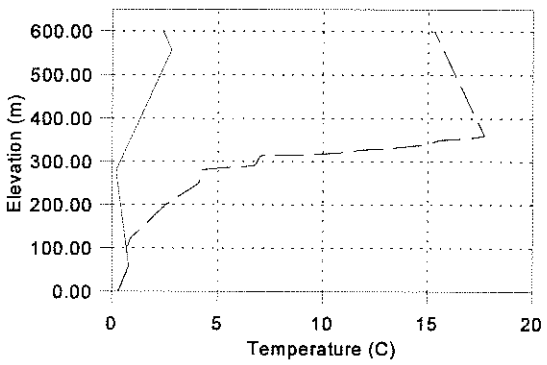
Fig. 5.13 Comparison of Transfer Characteristics

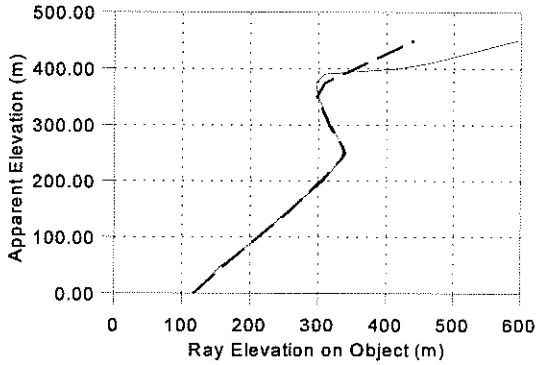
shows that Simulation #5 (short dashed lines) has the smallest temperature inversion of the three sloped atmospheres. Although a 10 degree inversion is more likely than a 14 degree one, it is impossible to indicate which is most reasonable.

Simulation #3 is chosen as the best fit when both the transfer characteristic curves and temperature profiles are both considered. This configuration has a smaller inversion than Simulation # 4, and its transfer characteristic is nearly the same.

Angle of Sloped Atmosphere

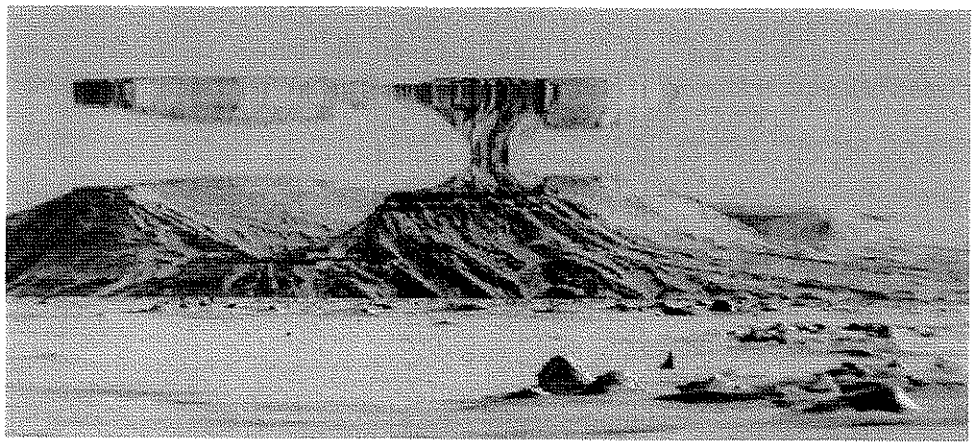
This section compares the effect of changing the slope of region two of Simulation #3 (Fig. 5.9). Atmosphere one is flat for 35.0 kilometres and atmosphere two is sloped at 11.5 arc minutes for 42.7 kilometres. Simulations #7, #8, and #9 (Fig. 5.14, 5.15, and 5.16) use a slope of 10.5, 12.633, and 13.5 arc minutes respectively.





Atmosphere One (35.0 km, 0')
Atmosphere Two (42.7 km, 10.5')

Simulated Transfer Characteristic
Original Transfer Characteristic



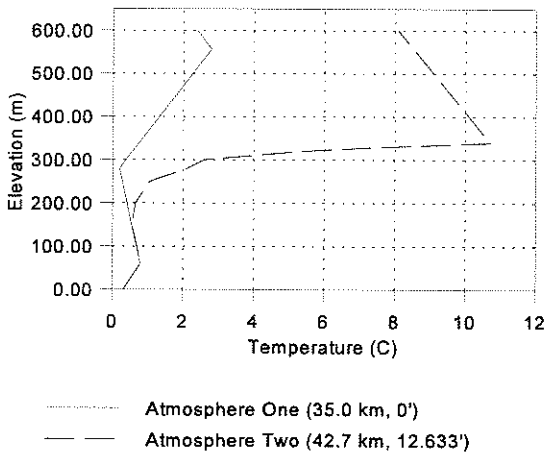
Temperature Profile (a5.tp, a17.tp)

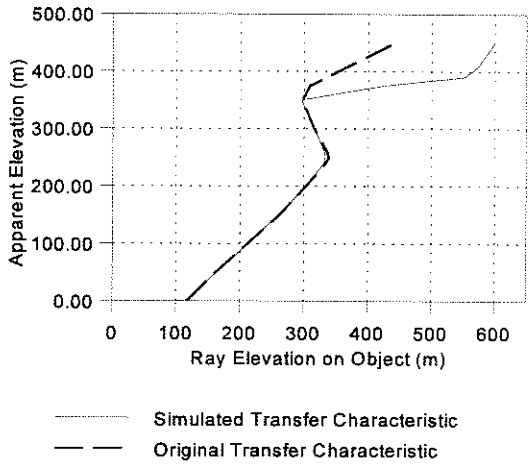
Atmosphere 1: Width: 35.0 kilometres Slope: 0 arc minutes
Atmosphere 2: Width: 42.7 kilometres Slope: 10.5 arc minutes

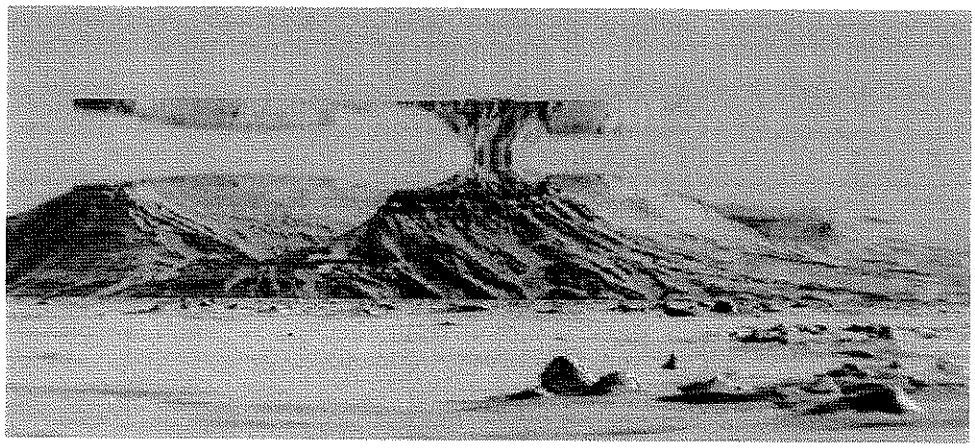
Transfer Characteristic (a17.tc)

Ray Angles: -13.3, -11.09, -8.88, -6.67, -4.46, -2.692, -2.25, -1.808, -0.04, 2.17, 3.275,3.938, 4.38, 4.822, 6.59

Fig. 5.14 Simulation #6 - New Slope for Region Two (10.5 arc minutes)







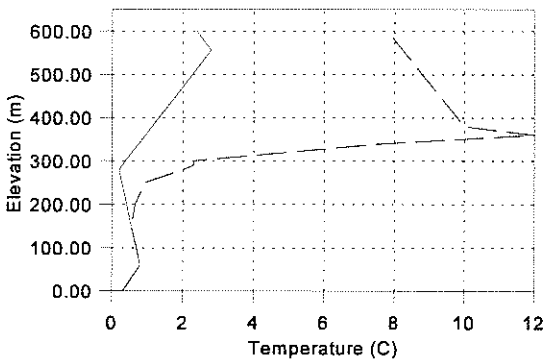
Temperature Profile (a5.tp, a7.tp)

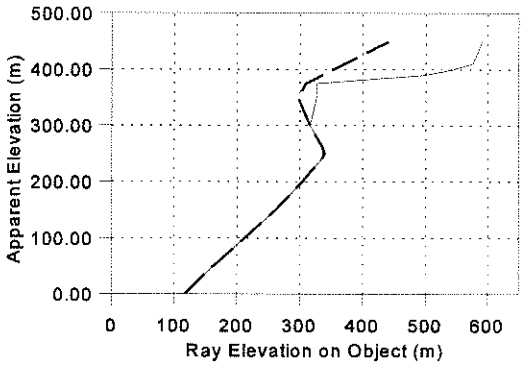
Atmosphere 1: Width: 35.0 kilometres Slope: 0 arc minutes
Atmosphere 2: Width: 42.7 kilometres Slope: 12.633 arc minutes

Transfer Characteristic (a7.tc)

Ray Angles: -13.3, -11.09, -8.88, -6.67, -4.46, -2.692, -2.25, -1.808, -0.04, 2.17, 3.275,3.938, 4.38, 4.822, 6.59

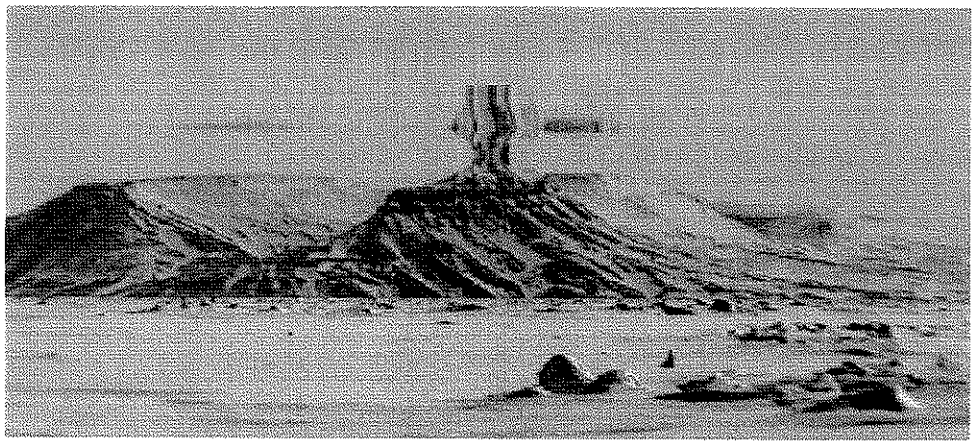
Fig. 5.15 Simulation #7 - New Slope for Region Two (12.633 arc minutes)





Atmosphere One (35.0 km, 0')
Atmosphere Two (42.7 km, 13.5')

Simulated Transfer Characteristic
Original Transfer Characteristic



Temperature Profile (a5.tp, a9.tp)

Atmosphere 1: Width: 35.0 kilometres Slope: 0 arc minutes
Atmosphere 2: Width: 42.7 kilometres Slope: 13.5 arc minutes

Transfer Characteristic (a9.tc)

Ray Angles: -13.3, -11.09, -8.88, -6.67, -4.46, -2.692, -2.25, -1.808, -0.04, 2.17, 3.275,3.938, 4.38, 4.822, 6.59

Fig. 5.16 Simulation #8 - New Slope for Region Two (13.5 arc minutes)

It is clear that from Fig. 5.17 Simulation #7 and Simulation #8 produce poorer transfer characteristics than Simulation #3. However, Simulation #6 compares favourably with Simulation #3 (Fig. 5.18). Examining the temperature profiles of Simulation #3 and

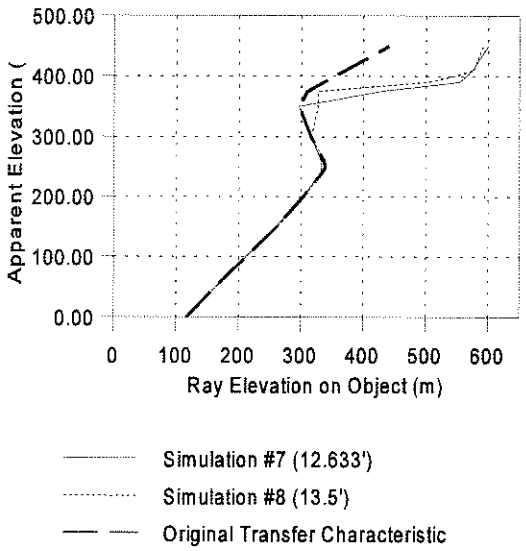


Fig. 5.17 Unacceptable Simulations

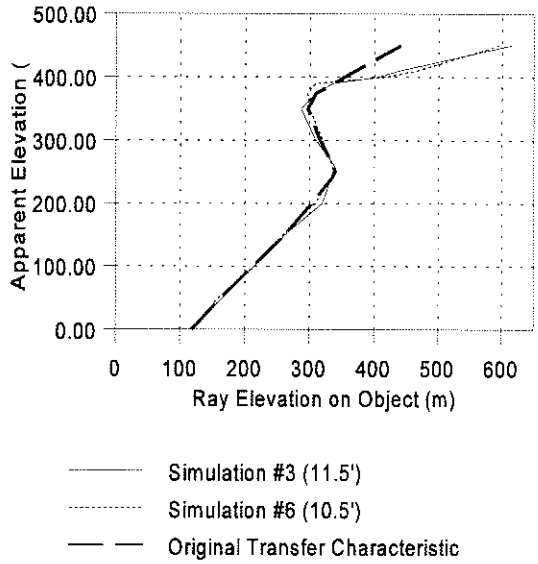


Fig. 5.18 Acceptable Simulations

#6, indicates that the former's inversion is much weaker (Fig. 5.19). On this basis, Simulation #3 appears to be the best overall. It should also be noted that the size of the

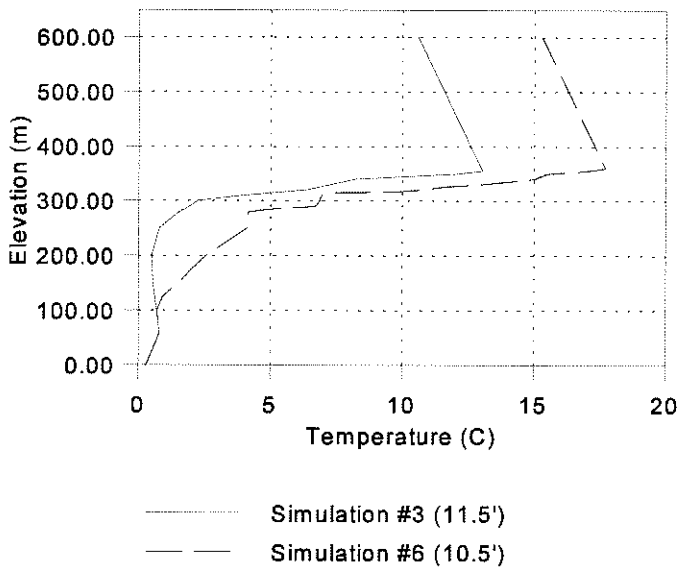


Fig. 5.19 Comparing Region Two Atmospheres

inversions increases as the slope of the atmosphere is reduced below 10.5 arc minutes. For slopes increasing beyond 13.5 arc minutes, the transfer characteristics get worse compared to the original (Fig. 5.17). A slope of 11.5 arc minutes appears to provide the best model atmosphere.

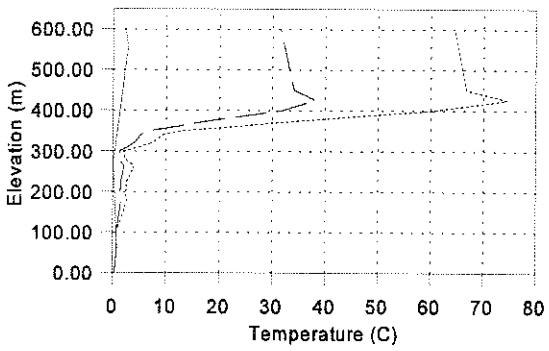
Multiple Regions

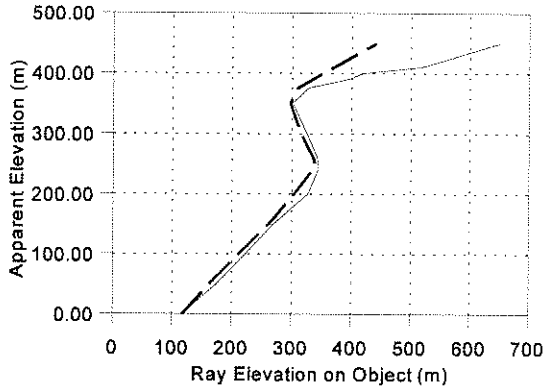
This is section will explore the effects of additional atmosphere regions on modelling the Somerset Island mirage. Simulations #9, #10, and #11 (Figs. 5.20 - 5.22) explore a three region atmosphere.

Simulation #9 (Fig. 5.20) shows a three region atmosphere simulation where the first region is flat with a width of 30.1 kilometres, and the second two regions have slopes of 11.5 arc minutes and widths of 23.8 kilometres each. The temperature profile for region two is chosen to be the average of region one and three. Results (Fig. 5.20) indicate that a very large inversion is required to generate an acceptable transfer characteristic.

Figure 5.21 (Simulation #10) models a three region atmosphere where the first region is flat with a width of 35.0 kilometres, and the other two regions are sloped at 11.5 arc minutes. The second region has narrow width of 4.9 kilometres, leaving the remaining third region with a width of 37.8 kilometres. The narrow intermediate atmosphere was chosen to provide a smooth transition between region one and three. Once again, the temperature profile of region two was chosen as an average between region one and three. Simulation #10 produces an acceptable temperature profile, but the transfer characteristic and resulting image were not very good.

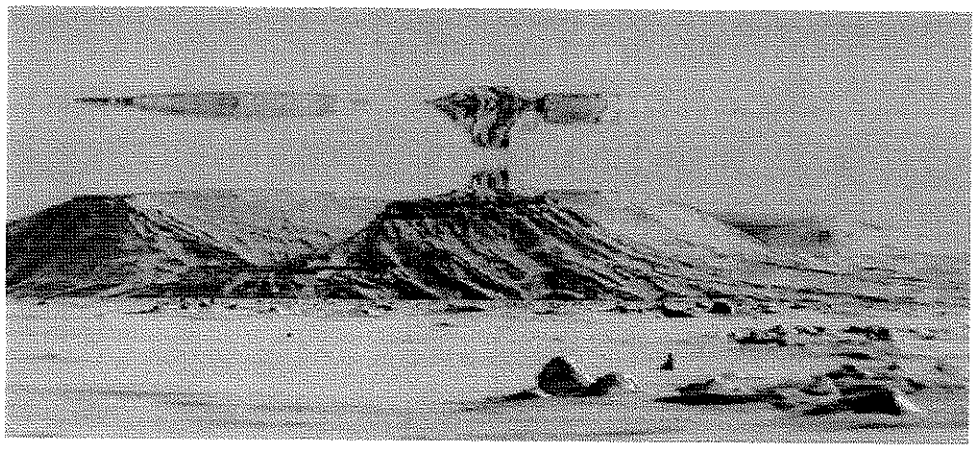
Simulation #11 (Fig. 5.22) used the same parameters as Simulation #10 except the middle transition region was shifted so that the flat atmosphere was 4.2 kilometres narrower





Atmosphere One (30.1 km, 0')
Atmosphere Two (23.8 km, 11.5')
Atmosphere Three (23.8 km, 11.5')

Simulated Transfer Characteristic
Original Transfer Characteristic



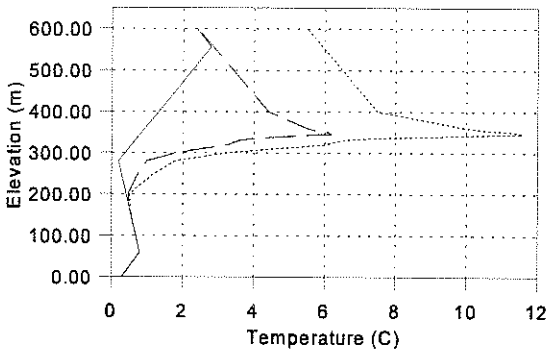
Temperature Profile (a19.tp, a19a.tp, a19b.tp)

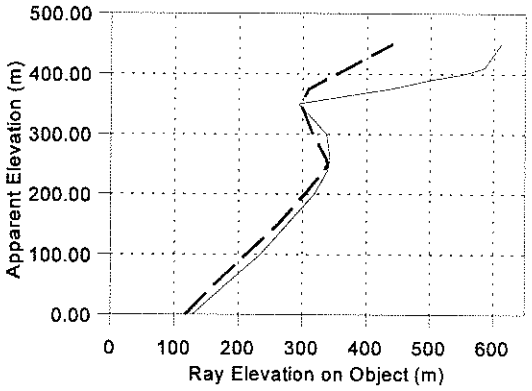
Atmosphere 1:	Width: 30.1 kilometres	Slope: 0 arc minutes
Atmosphere 2:	Width: 23.8 kilometres	Slope: 11.5 arc minutes
Atmosphere 3:	Width: 23.8 kilometres	Slope: 11.5 arc minutes

Transfer Characteristic (a19.tc)

Ray Angles: -13.3, -11.09, -8.88, -6.67, -4.46, -2.692, -2.25, -1.808, -0.04, 2.17, 3.275,3.938, 4.38, 4.822, 6.59

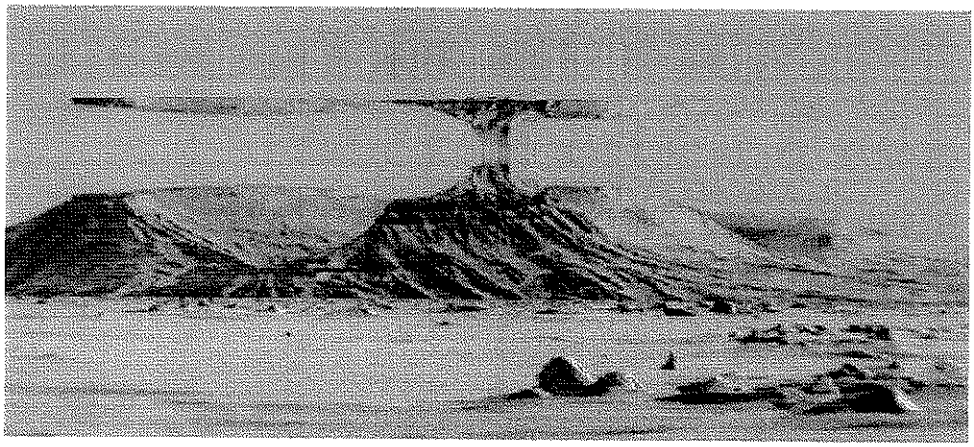
Fig. 5.20 Simulation #9 - Three Region Atmosphere (30.1 km, 23.8 km, 23.8 km)





Atmosphere One (35.0 km, 0')
Atmosphere Two (4.9 km, 11.5')
Atmosphere Three (37.8 km, 11.5')

Simulated Transfer Characteristic
Original Transfer Characteristic



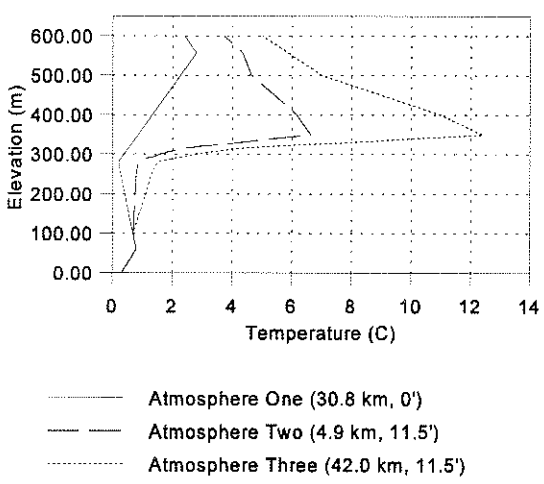
Temperature Profile (a20.tp, a20a.tp, a20b.tp)

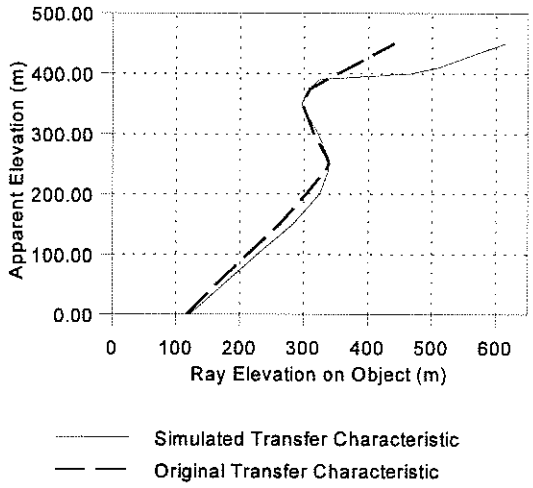
Atmosphere 1:	Width: 35.0 kilometres	Slope: 0 arc minutes
Atmosphere 2:	Width: 4.9 kilometres	Slope: 11.5 arc minutes
Atmosphere 3:	Width: 37.8 kilometres	Slope: 11.5 arc minutes

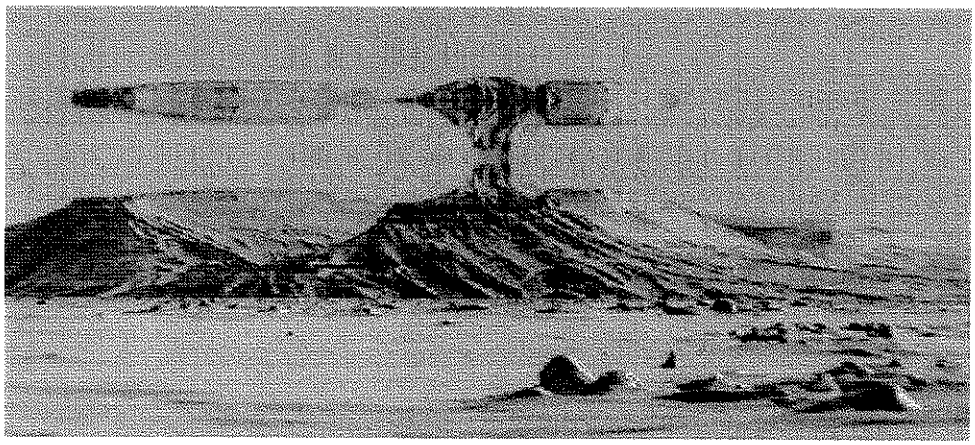
Transfer Characteristic (a20.tc)

Ray Angles: -13.3, -11.09, -8.88, -6.67, -4.46, -2.692, -2.25, -1.808, -0.04, 2.17, 3.275,3.938, 4.38, 4.822, 6.59

Fig. 5.21 Simulation #10 - Three Region Atmosphere (35.0 km, 4.9 km, 37.8 km)







Temperature Profile (a21.tp, a21a.tp, a21b.tp)

Atmosphere 1:	Width: 30.8 kilometres	Slope: 0 arc minutes
Atmosphere 2:	Width: 4.9 kilometres	Slope: 11.5 arc minutes
Atmosphere 3:	Width: 42.0 kilometres	Slope: 11.5 arc minutes

Transfer Characteristic (a21.tc)

Ray Angles: -13.3, -11.09, -8.88, -6.67, -4.46, -2.692, -2.25, -1.808, -0.04, 2.17, 3.275,3.938, 4.38, 4.822, 6.59

Fig. 5.22 Simulation #11 - Three Region Atmosphere (30.8 km, 4.9 km, 40.0 km)

and atmosphere three was 4.2 kilometres wider. The resulting simulation and temperature profile were both acceptable.

The results of these simulations indicate that adding an atmosphere region can produce a better simulation, but changing their slopes has less effect.

5.3.5 Summary of Results

This section consisted of eleven simulations of the long range mirage of Somerset Island. Results indicate that a single flat or sloped atmosphere cannot simulate the mirage with a realistic temperature profile. This was not unexpected because an atmosphere does not stay uniform over long distances. Better simulations were observed by introducing a multi-region model atmosphere. The widths of both regions were varied and the slope of region two was also altered to test their effects on the simulation. A model consisting of a flat atmosphere for 35.0 kilometres and a second region of 42.7 kilometres sloped at 11.5 arc minutes produced the best results. In addition, it was shown that an even better model could be produced by introducing an intermediate third region to provide a more gradual transition from one region to another.

These results indicate that the Somerset Island mirage can be best simulated with a multiple sloped atmosphere. Simulation #3 and Simulation #11 (Fig. 5.8 and Fig. 5.22) were the best of the eleven atmospheric configurations tested. The meteorological phenomena responsible for producing this atmosphere were likely a combination of heating over Somerset Island and movement of this elevated warmer air towards Resolute. Somerset Island was mostly snow free and able to absorb considerable radiant energy from the sun. The white snow cover between Resolute and the island would remain cold as it reflected most of the sun's short wave radiation back to space.

5.4 Simulating a Sequence of Long Range Mirages

On June 11, 1987, a sequence of mirages of Lowther Island was observed from Resolute Bay, NWT. This sequence looks as if it could be periodic and explained by gravity waves. Previous attempts were made to simulate this mirage sequence with a flat atmospheric model, but these failed due to the long distance our which the mirage was observed. Lowther Island is located 71.5 kilometres west of Resolute (Fig. 5.1).

This section will generate the transfer characteristic of a mirage near the mid-point of the sequence as a representative mirage. An atmosphere will be modelled for the mirage and the resulting temperature profile will be smoothed to create a new representative mirage. Using this new mirage, a number of gravity wave sequences will be generated to find the best simulation of the mirage sequence.

5.4.1 Generating a Transfer Characteristic

Figure 5.23 shows the sequence of mirages observed on June 11, 1987. The first picture was taken at 10:28 am CDT with subsequent pictures every 30 seconds. The total number of pictures in the sequence is 25, but only every third picture (every 90 seconds) is shown.

An average mirage illustrated in Fig. 5.24 was chosen from near the middle of the sequence. The distance from the observer to the mirage was 71.5 kilometres, the observer's eye elevation was 57 metres above sea level, and the initial ray angle for the horizon was measured at about minus 13 arc minutes.

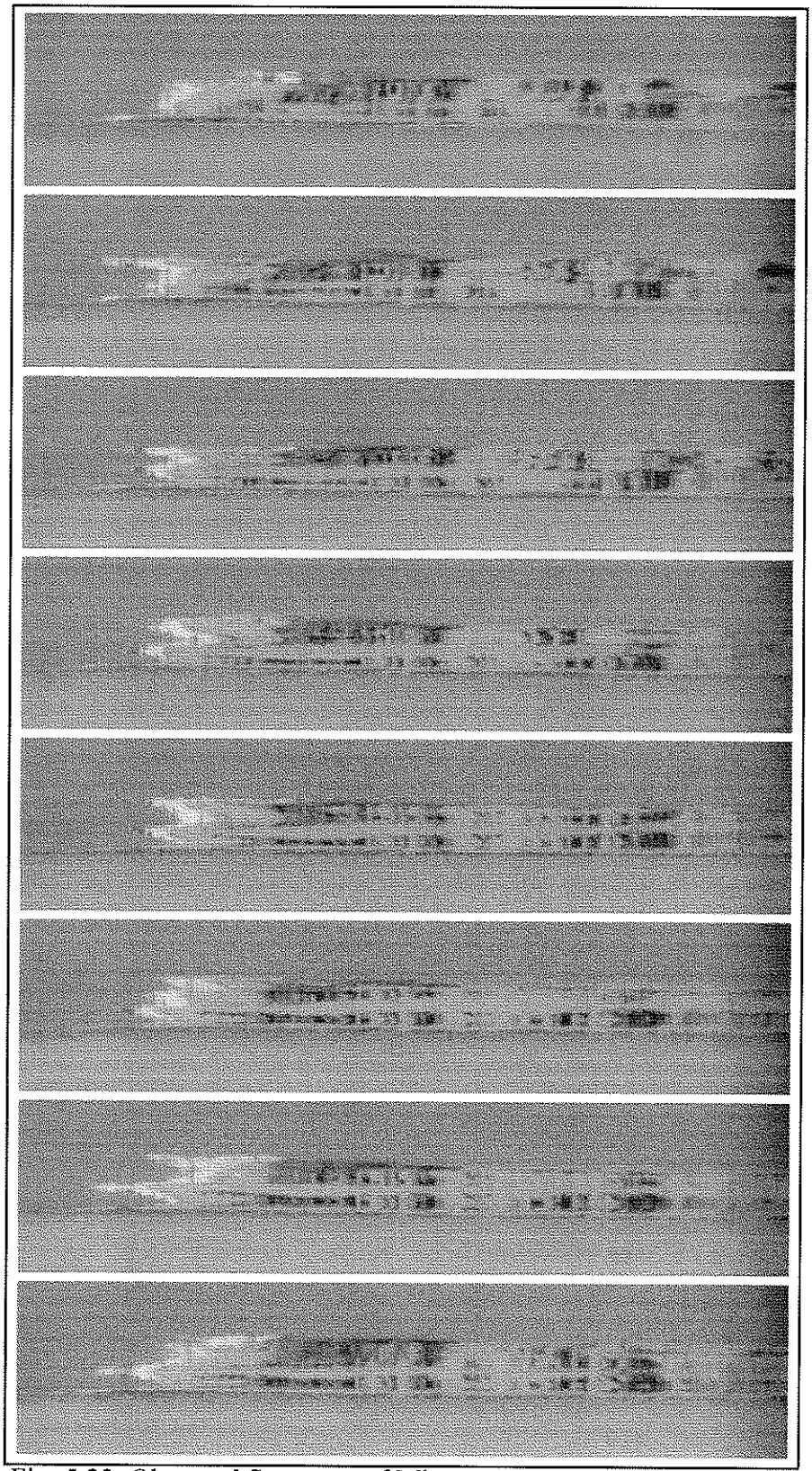


Fig. 5.23 Observed Sequence of Mirages

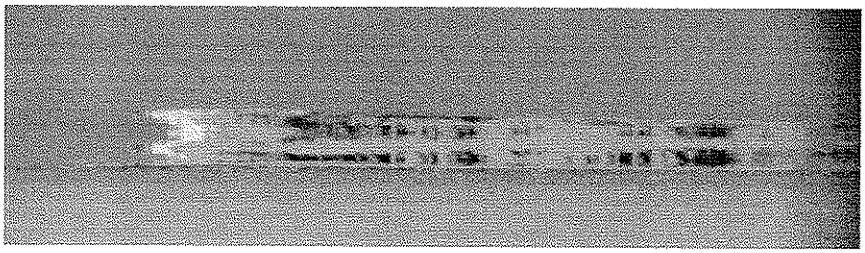


Fig. 5.24 Average Mirage of Lowther Island Sequence

Figure 5.25 is an undistorted image of Lowther Island taken from about the same location as the sequence. Using the *maketc* and *edittc* programs, the transfer characteristic

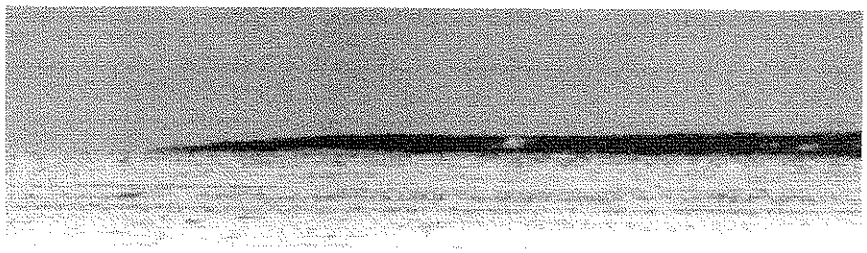


Fig. 5.25 Lowther Island without a Mirage

shown in Fig. 5.26 was generated. Using this transfer characteristic and the original image shown in Fig. 5.25, a simulation was made using the mirage simulator (*rgbmir*). The

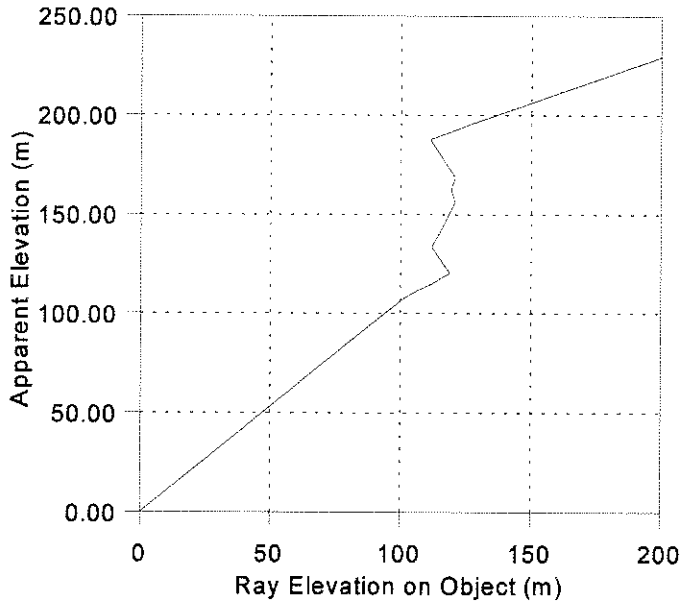


Fig. 5.26 Transfer Characteristic for Lowther Mirage

resulting simulation (Fig. 5.27) is much darker than the mirage image (Fig. 5.24) was taken when the island had much less snow cover. The grey-white portions of the mirage are caused by snow on the ground.

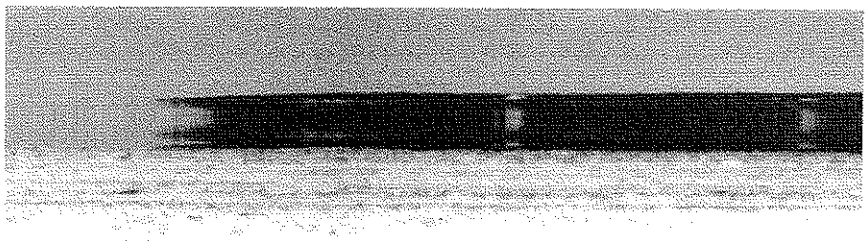
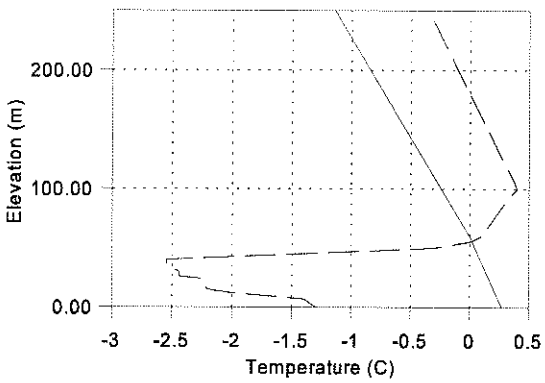


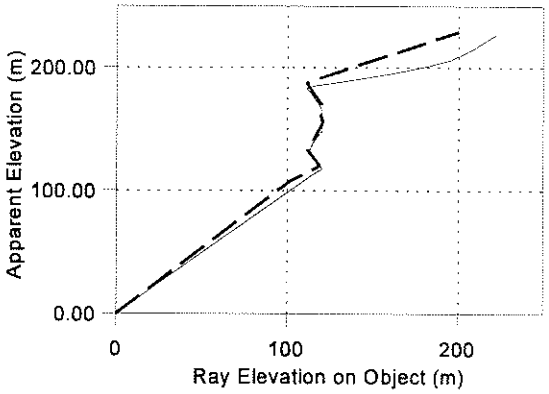
Fig. 5.27 Simulated Mirage of Lowther Island

5.4.2 Simulating an Average Mirage in the Sequence

Modelling an atmosphere with a similar transfer characteristic requires the selection of appropriate ray angles. In order to capture all of the transfer characteristics, the lowest ray should be chosen carefully with subsequent rays approximately equally spaced in elevation. Since the observed horizon elevation was minus 13 arc minutes, an angle slightly higher (-12.9 arc minutes) was selected as the lowest ray angle. The ray tracing was done in steps of 500 metres, the distance to the mirage was 71.5 kilometres, and the observer's eye was at a height of 57 metres.

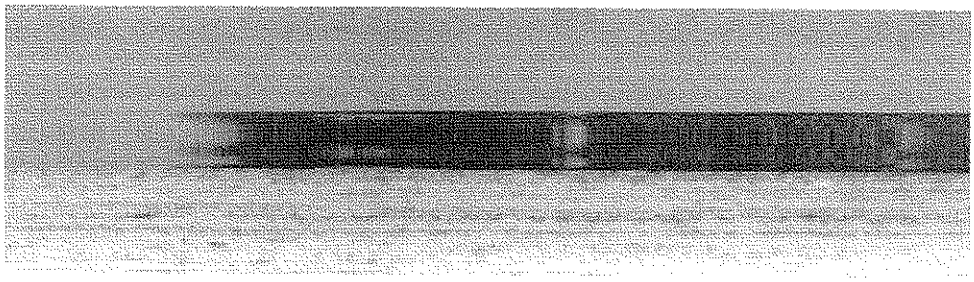
Simulations A, B, C, and D are shown in Figs. 5.28, 5.29, 5.30 and 5.31 respectively. The atmosphere in the four simulations consisted of two regions of equal width. The first region was defined as flat and the second was sloped. The slopes selected were 0, 5, 6, and 8 arc minutes respectively.





Atmosphere One (35.5 km, 0')
Atmosphere Two (36.0 km, 0')

Simulated Transfer Characteristic
Original Transfer Characteristic



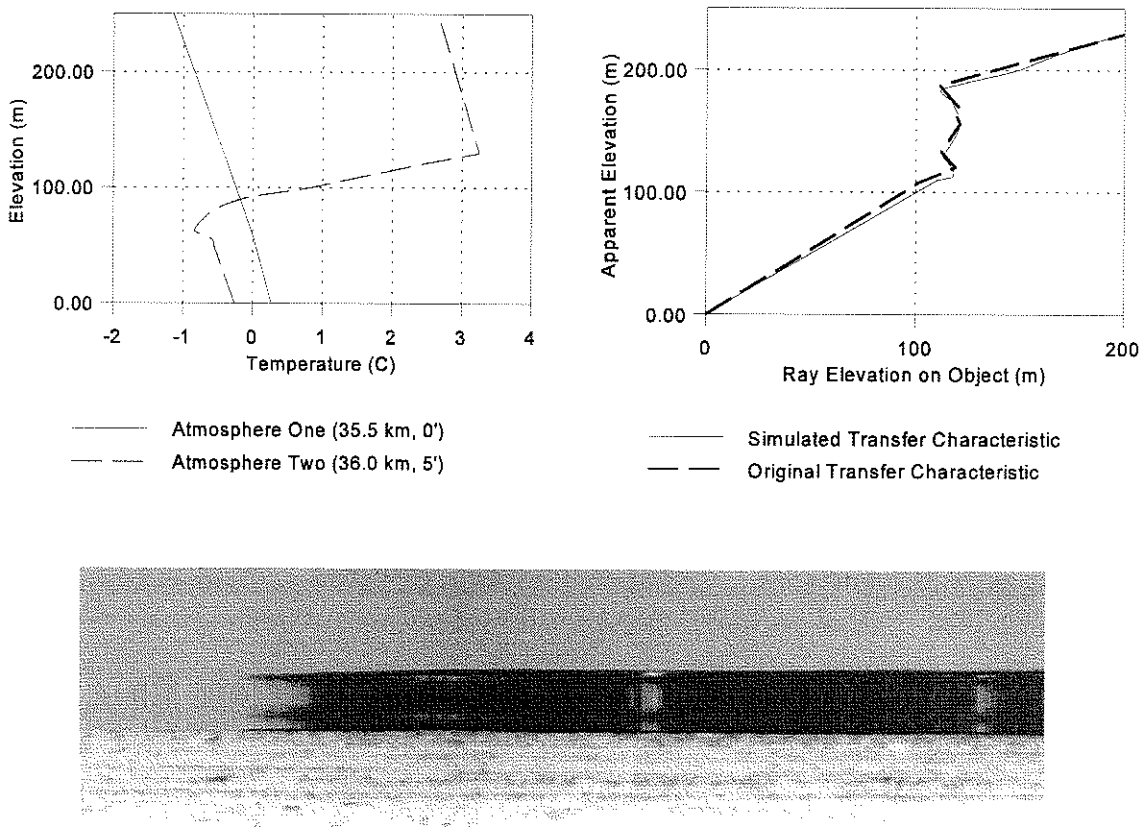
Temperature Profile (b5a.tp b5b.tp)

Atmosphere 1: Width: 35.5 kilometres Slope: 0 arc minutes
Atmosphere 2: Width: 36.0 kilometres Slope: 0 arc minutes

Transfer Characteristic (b5.tc)

Ray Angles: -12.982, -12.75, -12.6, -12.5, -12.25, -12.0, -11.90, -11.75, -11.5, -11.25, -11.0, -10.75, -10.5, -10.25, -10.0, -9.75, -9.5, -9.35, -9.25, -9.0, -8.75, -8.5, -8.25, -8.0, -7.75, -7.5, -7.25

Fig. 5.28 Simulation A - Slope Simulation (0 arc minutes) of Region Two



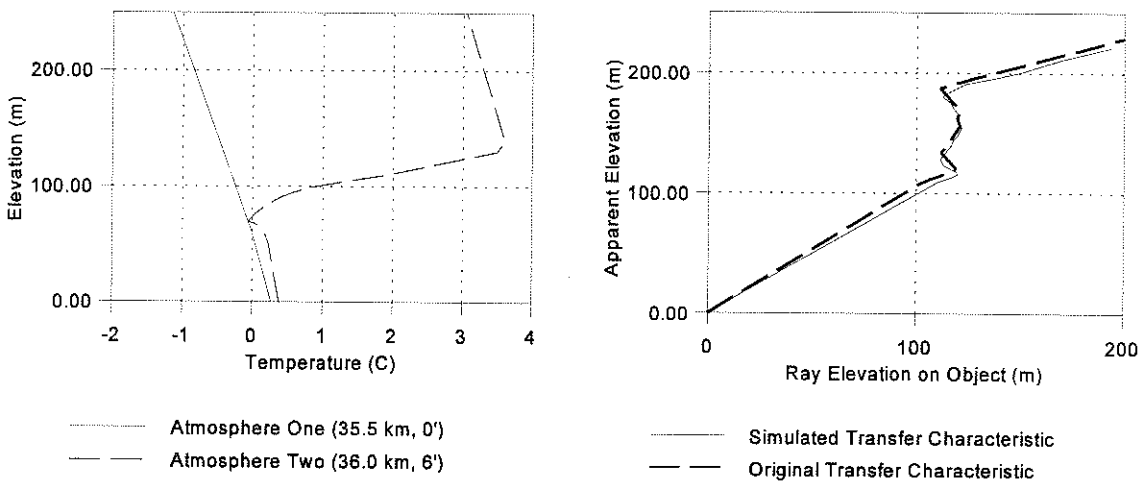
Temperature Profile (b3a.tp, b3b.tp)

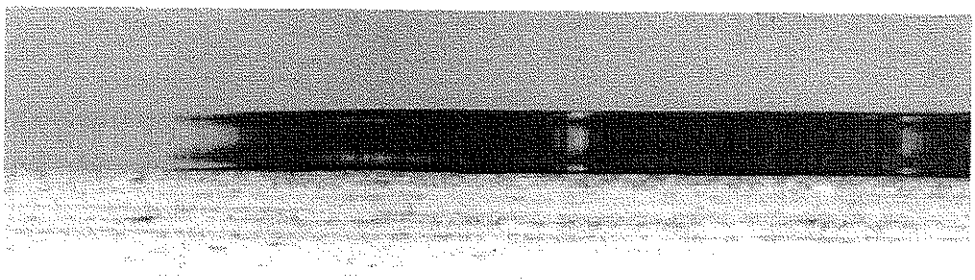
Atmosphere 1: Width: 35.5 kilometres Slope: 0 arc minutes
Atmosphere 2: Width: 36.0 kilometres Slope: 5 arc minutes

Transfer Characteristic (b3.tc)

Ray Angles: -12.982, -12.75, -12.6, -12.5, -12.25, -12.0, -11.90, -11.75, -11.5, -11.25, -11.0, -10.75, -10.5, -10.25, -10.0, -9.75, -9.5, -9.35, -9.25, -9.0, -8.75, -8.5, -8.25, -8.0, -7.75, -7.5, -7.25

Fig. 5.29 Simulation B - Slope Simulation (5 arc minutes) of Region Two





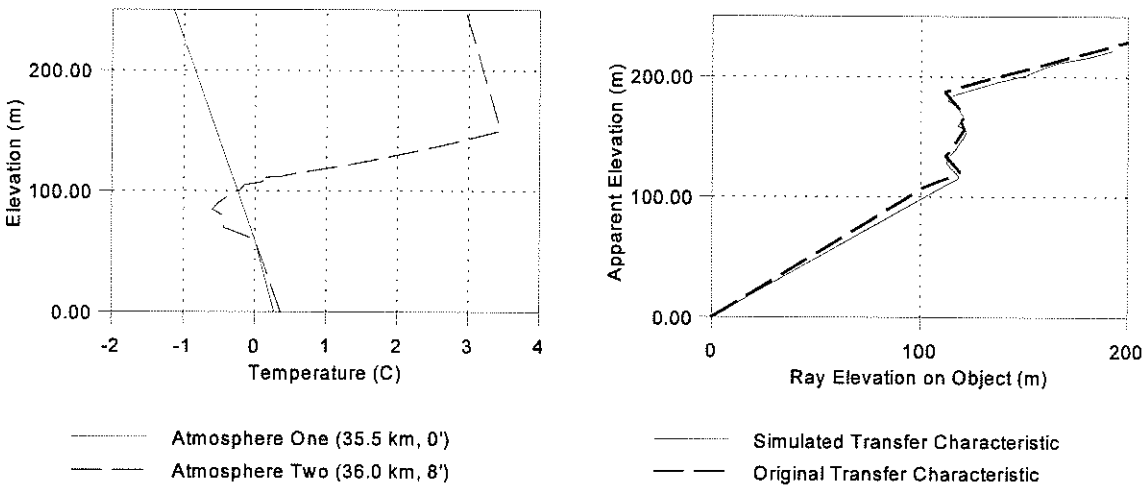
Temperature Profile (b4a.tp, b4b.tp)

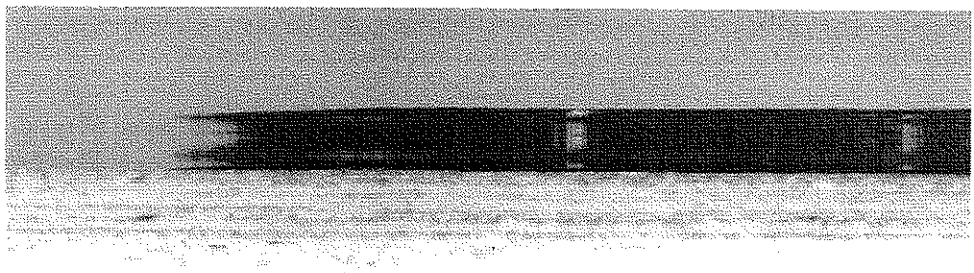
Atmosphere 1:	Width: 35.5 kilometres	Slope: 0 arc minutes
Atmosphere 2:	Width: 36.0 kilometres	Slope: 6 arc minutes

Transfer Characteristic (b4.tc)

Ray Angles: -12.982, -12.75, -12.6, -12.5, -12.25, -12.0, -11.90, -11.75, -11.5, -11.25, -11.0, -10.75, -10.5, -10.25, -10.0, -9.75, -9.5, -9.35, -9.25, -9.0, -8.75, -8.5, -8.25, -8.0, -7.75, -7.5, -7.25

Fig. 5.30 Simulation C - Slope Simulation (6 arc minutes) of Region Two





Temperature Profile (b6a.tp, b6b.tp)

Atmosphere 1: Width: 35.5 kilometres Slope: 0 arc minutes
Atmosphere 2: Width: 36.0 kilometres Slope: 8 arc minutes

Transfer Characteristic (b6.tc)

Ray Angles: -12.982, -12.75, -12.6, -12.5, -12.25, -12.0, -11.90, -11.75, -11.5, -11.25, -11.0, -10.75, -10.5, -10.25, -10.0, -9.75, -9.5, -9.35, -9.25, -9.0, -8.75, -8.5, -8.25, -8.0, -7.75, -7.5, -7.25

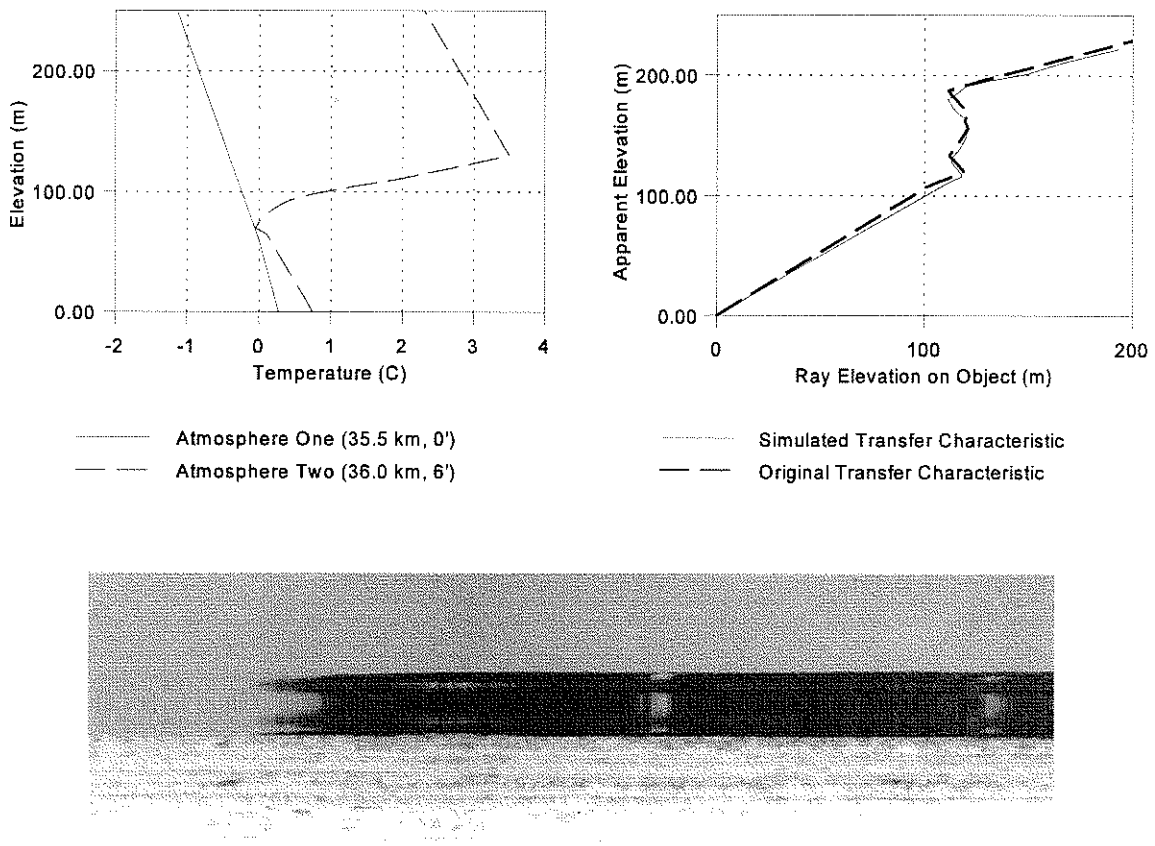
Fig. 5.31 Simulation D - Slope Simulation (8 arc minutes) of Region Two

In each of these simulations, region one of the two-region atmosphere was assumed to have a lapse rate of 6°C per kilometre, with a slightly lesser rate for the lower 50 metres near the ground. Each simulation produced an acceptable transfer characteristic, but the temperature profile for Simulation A does not look reasonable. The super-adiabatic lapse rate in the lower 40 metres could not occur over an ice surface. The temperature profile of Simulation D is possible, but the structure just below the inversion is not likely to be long lasting or constant over a long distance. Since an average profile is desired, this one is eliminated from consideration. Of the remaining two simulations, both profiles appear to be reasonable. The profile for Simulation C was selected as the best because the inversion overall was smaller.

Having chosen Simulation C as the best average mirage for the sequence, a representative mirage is generated from this simulation before gravity waves are introduced.

5.4.3 Creating a Representative Mirage

Section 5.4.2 simulated an average mirage of the sequence of mirages shown in Fig. 5.23. An average mirage is the simulation of one of the mirages in the sequence. A representative mirage is used to produce a sequence of mirages with gravity waves. To create the representative mirage, the average mirage transfer characteristic and temperature profile are smoothed and adjusted slightly. The resulting representative mirage is shown in Fig. 5.32 (Simulation E). It is interesting to note that the lapse rates above and below the inversion in the sloped atmosphere are nearly the same. To help to understand this simulation, a ray trace of the atmosphere is illustrated in Fig. 5.33.



Temperature Profile (b11a.tp, b11b.tp)

Atmosphere 1: Width: 35.5 kilometres Slope: 0 arc minutes
Atmosphere 2: Width: 36.0 kilometres Slope: 6 arc minutes

Transfer Characteristic (b11.tc)

Ray Angles: -12.982, -12.75, -12.6, -12.5, -12.25, -12.0, -11.90, -11.75, -11.5, -11.25, -11.0, -10.75, -10.5, -10.25, -10.0, -9.75, -9.5, -9.35, -9.25, -9.0, -8.75, -8.5, -8.25, -8.0, -7.75, -7.5, -7.25

Fig. 5.32 Simulation E - Representative Mirage

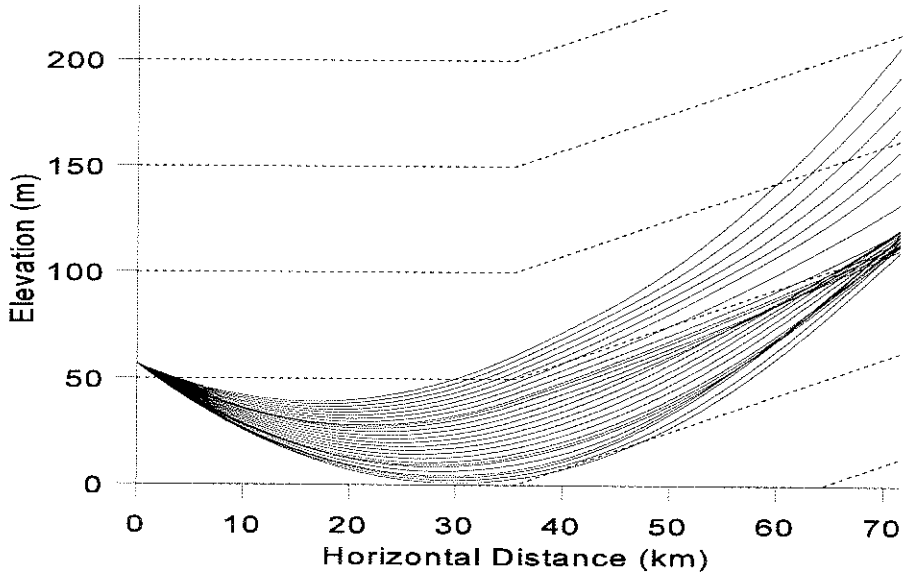


Fig. 5.33 Ray Trace through Representative Atmosphere

5.4.4 Simulating a Gravity Wave Sequence

Once a representative mirage is generated, the simulation of a gravity wave sequence begins by generating a four layer model of the temperature profile. Since the atmosphere represented in Fig. 5.32 has two regions, normally a four layer model would be required for each region. However, region one has a nearly constant lapse rate and any small vertical

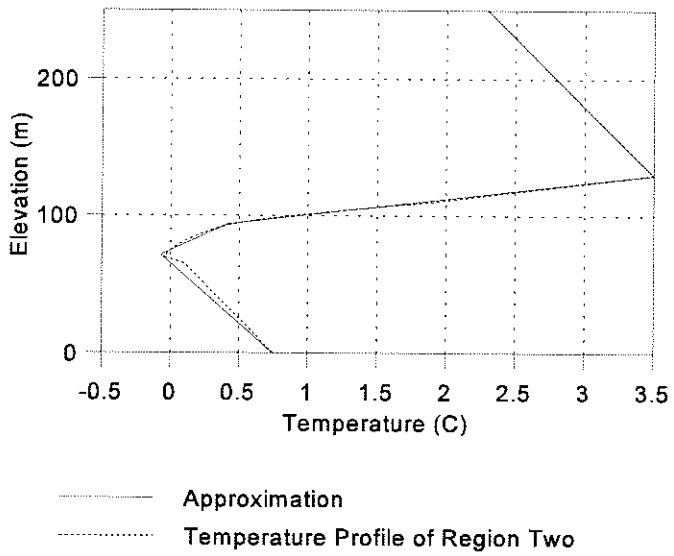


Fig. 5.34 Four-Layer Approximation

displacement would not alter the temperature profile. Therefore, gravity waves would have no effect on this region, and only the second region has to be approximated. As shown in Fig. 5.34, a four layer model, can be used as a good approximation of the profile. Once the four layer model is defined, the wave equations can be solved and values for k and ω for Eqn. 4.22 can be determined. The following italicized text is the *MathCAD* script that was used to calculate these values:

The changes in the slope of the temperature profile are at heights 0, h1, h2, h3; the corresponding temperatures are T0, T1, T2, T3. The respective layer thicknesses are t1=h1; t2=h2-h1;t3=h3-h2. Layer 4 is of infinite thickness.

Enter values of these parameters here.

$h1:=71.390 \quad h2:=93.846 \quad h3:=130 \quad t1:=h1 \quad t2:=h2-h1 \quad t3:=h3-h2$
$T0:=273+0.75 \quad T1:=273-0.069 \quad T2:=273+0.421 \quad T3:=273+3.5$

Temperature gradients:

$$grad1:=\frac{(T1-T0)}{h1} \qquad grad2:=\frac{(T2-T1)}{(h2-h1)} \qquad grad3:=\frac{(T3-T2)}{(h3-h2)}$$

Except for the top layer, the average temperature of the layer is used for T

$$Tav1:=\frac{(T1+T0)}{2} \qquad Tav2:=\frac{(T2+T1)}{2} \qquad Tav3:=\frac{(T3+T2)}{2}$$

$g:=9.8 \quad cp:=1004 \quad \beta:=3.48\times10^{-3} \quad \gamma:=1.4 \quad grad4:=-0.01$

$\frac{g}{cp}=0.009761$ *adiabatic lapse rate*

$$cs:=\sqrt{(\gamma\cdot\frac{Tav1}{\beta})} \qquad cs=331.60906 \quad \textit{speed of sound in bottom layer}$$

Speed of sound is assumed to be roughly the same in all layers.

The VB frequency N is determined by temperature and its gradient. The following equation gives the square of N.

$$Nsq(T,grad) := \frac{g}{T}\cdot\left(\frac{g}{cp} + grad\right)$$

The squares of the VB frequencies in the respective layers are

$N1sq := Nsq(Tav1,grad1)$ $\quad N1sq = -6.135256\cdot10^{-5}$ $\quad \sqrt{N1sq} = 0.007833i$

$N2sq := Nsq(Tav2,grad2)$ $\quad N2sq = 0.00114$ $\quad \sqrt{N2sq} = 0.0033763$

$N3sq := Nsq(Tav3,grad3)$ $\quad N3sq = 0.003367$ $\quad \sqrt{N3sq} = 0.058023$

$N4sq := Nsq(Tav4,grad4)$ $\quad N4sq = -8.472439\cdot10^{-6}$ $\quad \sqrt{N4sq} = 0.002911i$

The function Γ *can be expressed in terms of temperature and gradient, or it can also be expressed in terms of N:*

$$\Gamma1 := \frac{-N1sq}{(2\cdot g)} + \frac{g}{(2\cdot cs^2)} \qquad \Gamma1 = 4.769004\cdot10^{-5}$$

$$\Gamma2 := \frac{-N2sq}{(2\cdot g)} + \frac{g}{(2\cdot cs^2)} \qquad \Gamma2 = -1.359961\cdot10^{-5}$$

$$\Gamma3 := \frac{-N3sq}{(2\cdot g)} + \frac{g}{(2\cdot cs^2)} \qquad \Gamma3 = -1.272113\cdot10^{-4}$$

$$\Gamma4 := \frac{-N4sq}{(2\cdot g)} + \frac{g}{(2\cdot cs^2)} \qquad \Gamma4 = 4.499207\cdot10^{-5}$$

The 'natural' frequencies for the layers can now be defined.

$$\gamma1(k,\omega) := \sqrt{\left[\frac{-k^2}{\omega^2}\cdot(N1sq - \omega^2) + \Gamma1^2\right]}$$

$$n2(k,\omega) := \sqrt{\left[\frac{k^2}{\omega^2}\cdot(N2sq - \omega^2) + \Gamma2^2\right]}$$

$$n3(k,\omega) := \sqrt{\left[\frac{k^2}{\omega^2}\cdot(N3sq - \omega^2) + \Gamma3^2\right]}$$

$$\gamma4(k,\omega) := \sqrt{\left[\frac{-k^2}{\omega^2}\cdot(N4sq - \omega^2) + \Gamma4^2\right]}$$

Now, consider each side of the dispersion equation separately.

$$lhs(k,\omega) := \frac{\left[\tanh(\gamma 1(k,\omega)\cdot h1) + \frac{\gamma 1(k,\omega)}{n2(k,\omega)}\cdot\tan(n2(k,\omega)\cdot t2)\right]}{\left[\frac{-\gamma 1(k,\omega)}{n2(k,\omega)} + \tanh(\gamma 1(k,\omega)\cdot h1)\cdot\tan(n2(k,\omega)\cdot t2)\right]}$$

$$rhs(k,\omega) := \frac{\left[1 + \frac{\gamma 4(k,\omega)}{n3(k,\omega)}\cdot\tan(n3(k,\omega)\cdot t3)\right]}{\frac{n3(k,\omega)}{n2(k,\omega)}\cdot\left[\frac{\gamma 4(k,\omega)}{n3(k,\omega)} - \tan(n3(k,\omega)\cdot t3)\right]}$$

A value of ω *is chosen as follows and a start value of k is chosen (can be any value)*
$\omega := 0.001 \qquad k := 0.0002$

MathCAD can now find the exact value of k closest to the one selected that solves the following equation for the given ω *value.*
$lhs(k,\omega) = rhs(k,\omega)$
$ans := find(k)$
$ans = 2.745529 \times 10^{-4}$

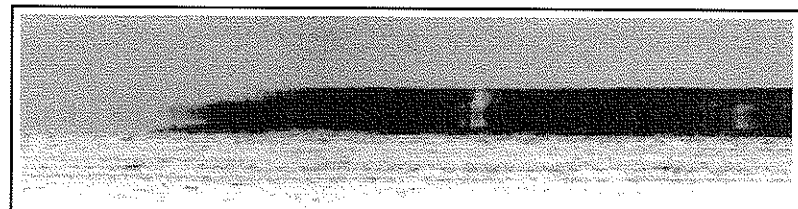
The script was used to calculate a number of k values for selected ω values. A table of the results is given as follows:

ω	k	Period (s)	Wavelength (m)
0.001	2.744529×10^{-4}	6283.2	22893
0.001	0.001322	6283.2	4753
0.001	0.002329	6283.2	2698
0.001	0.003304	6283.2	1902
0.002	5.530255×10^{-4}	3141.6	11361
0.002	0.002652	3141.6	2369
0.002	0.00467	3141.6	1345
0.004	0.001133	1570.8	5546
0.004	0.005366	1570.8	1171

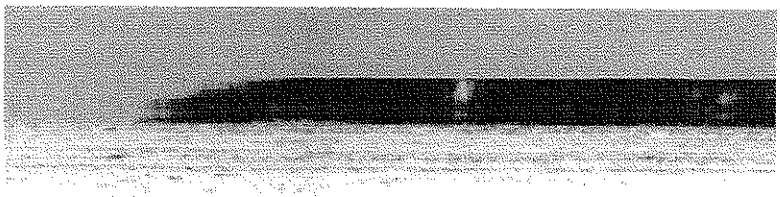
This table shows the large variety of possible combinations of ω and k values. The value of ω determines the period of the gravity wave, and k determines its wavelength. A number of simulations were performed to determine the best k values to create the desired sequence of mirages. Wavelengths of less than 10 kilometres had no effect on the representative mirage and no sequence was created. As the wavelength increased, the mirage sequence became more apparent. As indicated in the table, the period required for long wavelength is very large. However, the period of the sequence shown in Fig 5.23 appears to be fairly short as evidenced by the fairly rapid changes over the 30 minute observation. One possible explanation is that the waves are propagating at some angle to the line of sight.

Wave Sequence #1 to #4 (Figs. 5.35, 5.36, 5.37, and 5.38) show four mirage sequences produced by this process. The value of k (k=0.002744529) was held constant for the first three simulations. Wave amplitudes of 7.5, 5 and 2.5 metres respectively were used. The fourth simulation was produced with a value for k of 0.001322 and an amplitude of 5 metres.

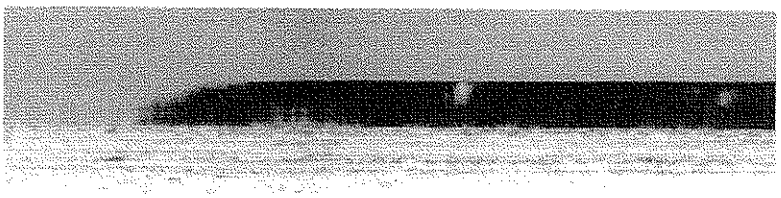
The last simulation (Fig. 5.38) shows that a large k value (shorter wavelength) results in an image sequence that changes little from step to step. Choosing other times in the sequence would not be expected to changes the results. Wave Sequence #3 (Fig. 5.37) with a longer wavelength is still unable to produce any apparent changes over the sequence period because of its relatively small wave amplitude (2.5 metres). Wave Sequence #1 and Wave Sequence #2 (Fig. 5.35 and 5.36) show sequences that have many characteristics similar to the original (Fig. 5.23). Of the two simulations, the one with a five metre amplitude (Fig. 5.36) looks the most like the original sequence.



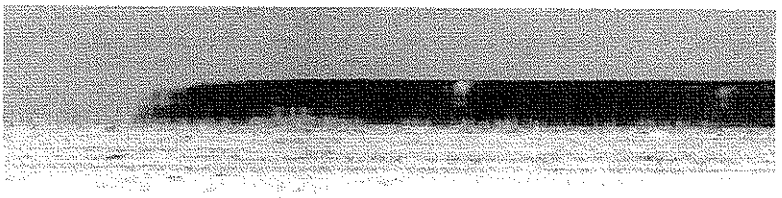
w: 0.0010
k: 0.0002744529
Ah: 7.5 metres
Time: 0 seconds



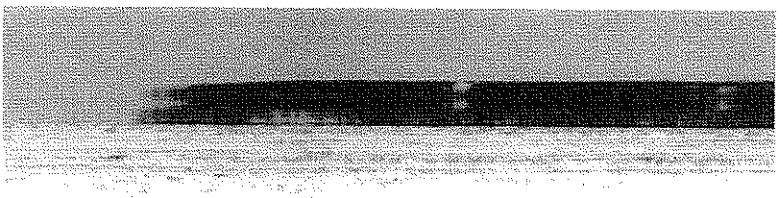
Time: 785.4 seconds



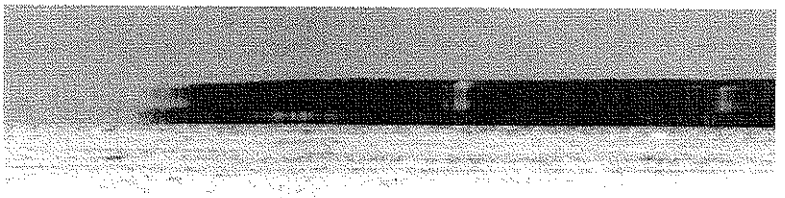
Time: 1570.8 seconds



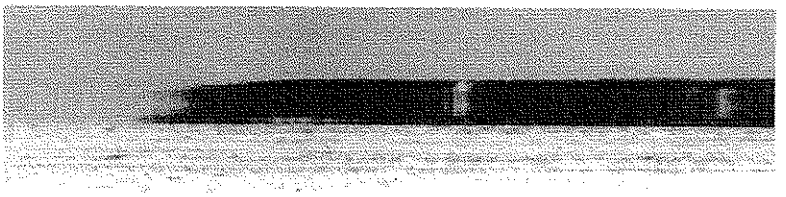
Time: 2356.2 seconds



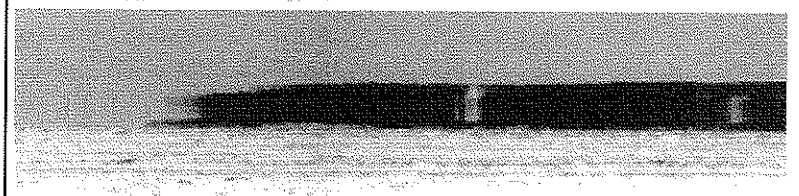
Time: 3141.6 seconds



Time: 3927.0 seconds



Time: 4712.4 seconds



Time: 5497.8 seconds

Fig. 5.35 Wave Sequence #1

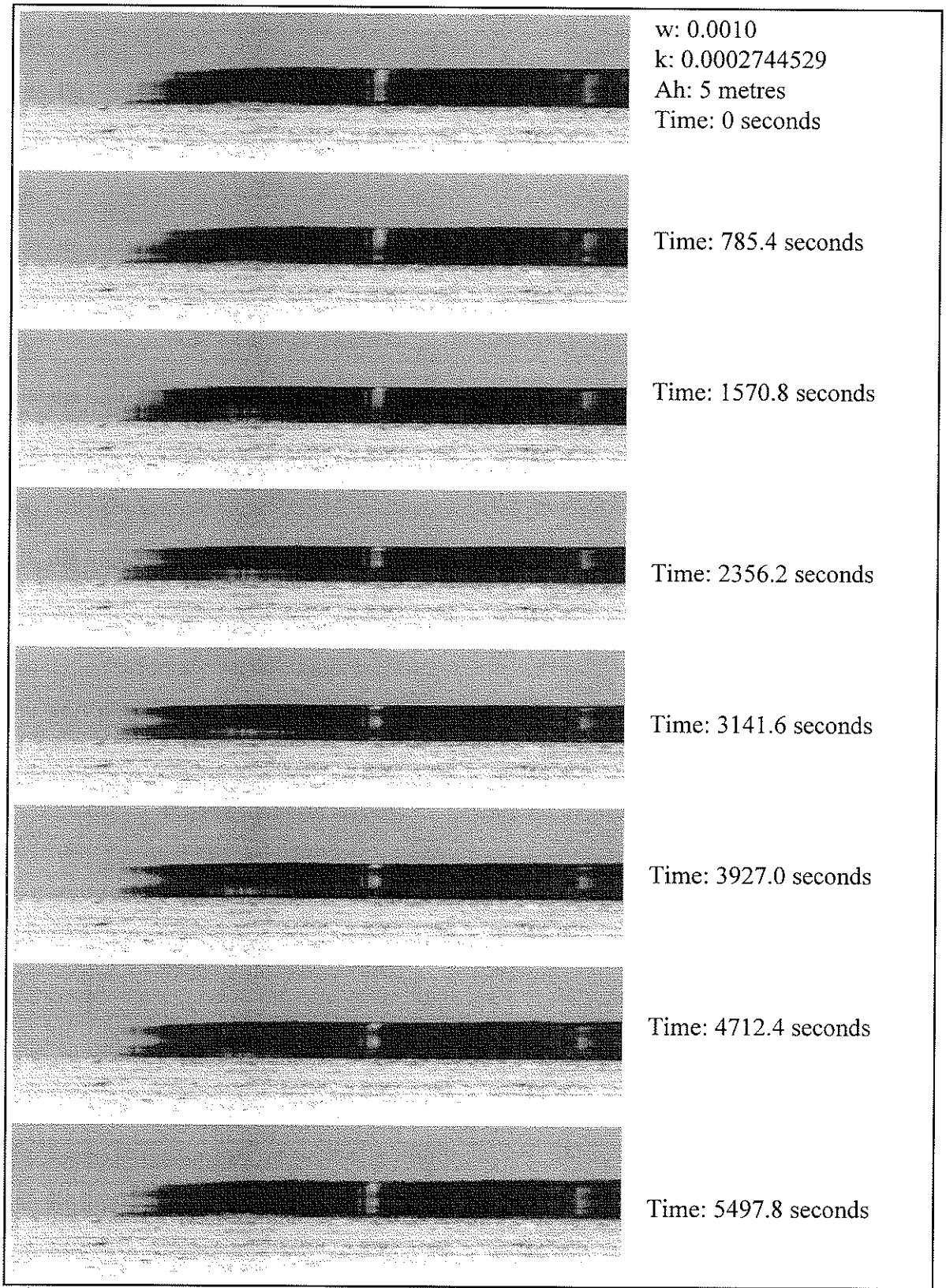


Fig. 5.36 Wave Sequence #2

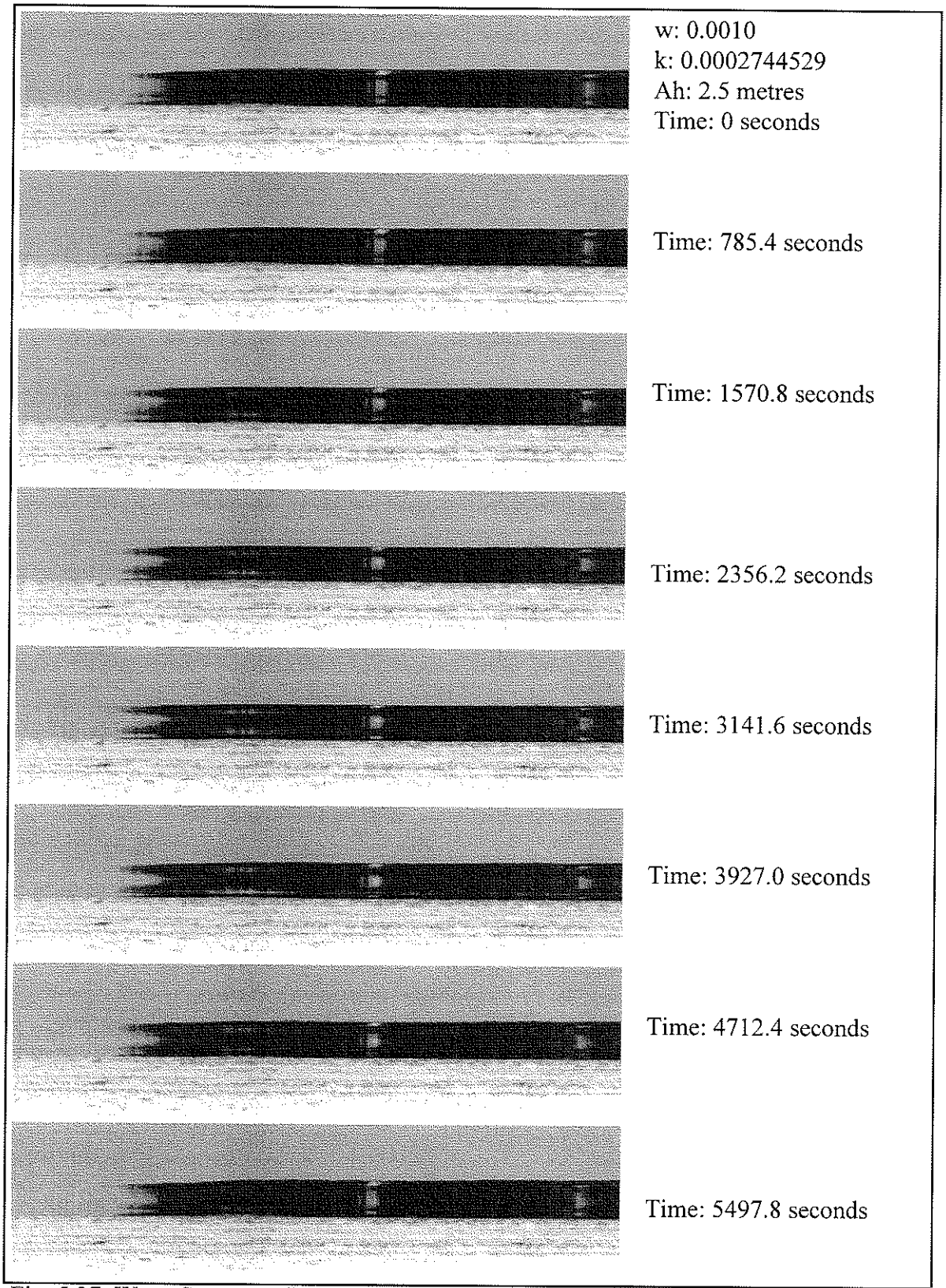


Fig. 5.37 Wave Sequence #3

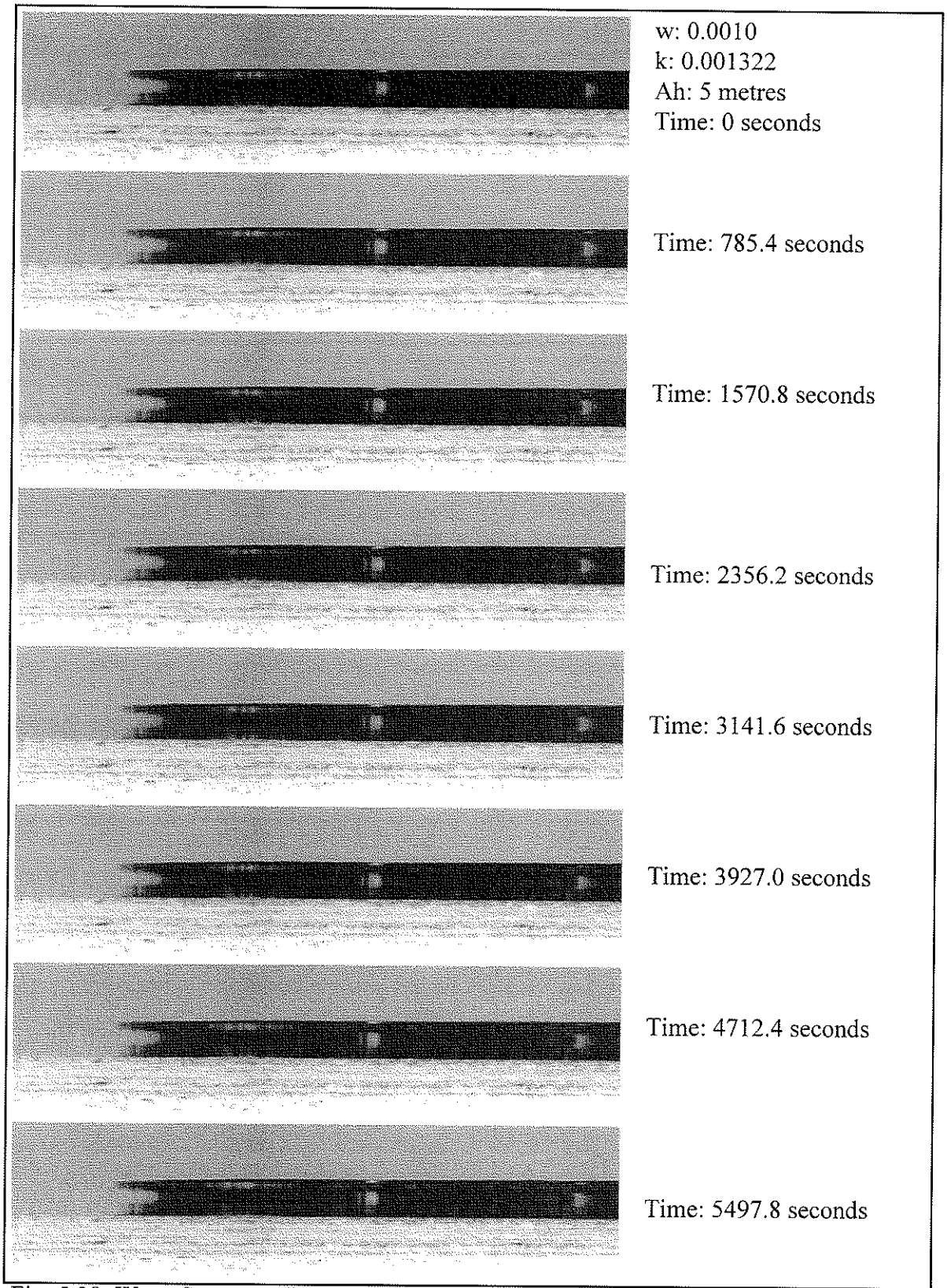


Fig. 5.38 Wave Sequence #4

5.4.5 Summary of Results

A sequence of mirages of Lowther Island displaying some periodic changes in appearance was photographed from Resolute Bay, NWT in 1987. Lowther Island is located 71.5 kilometres west of Resolute. It was expected that this sequence could be simulated with gravity waves and a sloped atmosphere. Since this mirage was observed over a long distance and there was no evidence of any short range distortions, a two atmosphere model was chosen for simulating the mirage. This model allowed the simulation of an average mirage in the sequence to be generated relatively easily. This average mirage and the transfer characteristic was smoothed of fine detail to create a representative mirage of the sequence. The atmosphere of this model was then represented by a four-layer temperature profile and solutions to the dispersion equations were found. Mirage sequences were simulated using these solutions for gravity waves. The results showed that a large wavelength was required to create an acceptable sequence, but the period of the same wave was much larger than expected. It was postulated that the apparent longer wavelengths observed were caused by viewing the sequence obliquely to the direction of propagations of the waves. Given this assumption, an acceptable solution to the sequence of mirage of Lowther Island was found for a wave number k=0.0002745 with an amplitude of 5 metres. The simulation required a sloped atmosphere in the second region away from the observer.

CHAPTER 6

Summary, Conclusions, and Recommendations

6.1 Conclusions

In the introduction to this thesis, questions were asked as to how mirages are simulated when the temperature profile of an atmosphere is not constant over the observing distance or when the ground or atmosphere has a slope. A solution to these problems was proposed by introducing a sloping atmosphere model to replace the flat atmosphere model used in previous research. This proposal also presented the possibility of multiple sloping atmospheres for simulating mirage sequences caused by atmospheric gravity waves.

The sloping atmosphere model was developed by first introducing the meteorology that affects the formation of mirages. Horizontal and vertical temperature variations in the atmosphere and the processes that cause these variations were thoroughly explored. These temperature variations cause changes in the refractive index of the atmosphere which in turn cause light rays to be bent. The types of mirages caused by different atmospheric temperature regimes were defined and classified.

Following this background discussion, the thesis presents the necessary mathematical theory for simulating mirages. It reviews the original flat atmosphere model and the multiple atmosphere model developed by earlier researchers. It also details how gravity waves sometimes produce periodic sequences of mirages and the models that have been developed to simulate this phenomena. The thesis outlines some of the limitations and problems that are sometimes encountered with these models. An atmosphere with sloping temperature layers, likely more common than horizontal stratifications, forms the basis for a model to

overcome these shortcomings.

The thesis develops the logic and theory for the new sloping model and integrates it with the flat and multi-region models. The remainder of the research examines its ability to simulate long range complex mirages. Two different long range mirages were chosen to test the sloping model. The first was a mirage of Somerset Island, an island south of Resolute Bay, NWT. This was a classic superior mirage in which peaks on the island were inverted, appearing above itself in a type of mirror image. The standard flat atmosphere models successfully simulated this mirage, but the temperature profiles required were unrealistic. The sloping model produced a successful simulation with a realistic temperature profile. A number of simulations were made by altering the widths and slopes of the multiple region atmosphere model. These simulations indicate that the real atmosphere was fairly flat for nearly half the distance from the observer to the island. The thermal layers then began to slope upward above the island. Heating of the relatively snow free surface on the island compared to very little warming over the ice explains the natural occurrence of such an atmosphere.

The second simulation used to test the sloping atmosphere model was a gravity wave simulation of a sequence of mirages of Lowther Island, an island west of Resolute Bay, NWT. This sequence, like the Somerset Island mirage, was observed over a long distance and the flat atmosphere model failed to provide a simulation with a realistic temperature profile. When a sloping atmosphere was introduced with a gravity wave simulation, a successful simulation with realistic atmospheric temperature levels was produced.

These two mirages demonstrate that the sloping atmosphere model successfully simulates mirages that cannot be simulated realistically with the flat atmosphere model. The atmospheric profiles defined by the model were quite reasonable and easily explained through natural meteorological processes.

6.2 Recommendations for Future Research

In the process of researching and writing this thesis a number of areas requiring further exploration became apparent:

- The process of simulating a mirage was lengthy and at times quite tedious. Automating the process of determining a temperature profile from a given transfer characteristic should be possible. An expert system or other type of "intelligent" software could help to automate or streamline significant portions of the whole simulation process.

- The sloping atmosphere model produced realistic simulations of mirages, but actual verification was not possible. Testing simulations of new mirages against actual temperature measurements in the field would be interesting and useful.

- Some simulations were especially sensitive to small changes in temperature. Small changes often produced drastic bending of the light rays. A method to measure the sensitivity rather than conducting frequent trial runs would help in determining the quality of a simulation.

- As mentioned in Chapter 4, the actual equations to define the sloping temperature surfaces could be developed. Simulations using surface equations rather than elevating the observer (the approach taken in this thesis) would be useful further research.

Appendix A

Experimental Data

This appendix contains the data from which all charts of the thesis are made.

Type: Transfer Characteristic **Name**: original.tc	
Apparent Elevation (m)	Actual Elevation (m)
-233.60826	117.60
-183.65719	162.71
-133.70632	212.38
-83.75561	262.02
-33.80503	304.02
6.15537	334.73
16.14547	340.00
26.13556	337.08
66.09592	316.44
116.04637	297.79
141.02161	309.78
156.00676	336.13
165.99687	353.70
175.98697	371.26
215.94744	441.53

Type: Transfer Characteristic **Name**: lowther.tc	
Apparent Elevation (m)	Actual Elevation (m)
0.00	0.00
107.82	101.50
111.83	107.00
115.27	112.50
120.44	119.00
126.17	116.00
133.63	112.00
142.80	116.00
155.99	121.00

162.30	119.50
168.61	121.00
187.61	114.45
190.63	127.48
229.40	200.00

Type: Temperature Profile
Name: a5.tp, a19.tp, a20.tp, a21.tp

Elevation (m)	Temperature (°C)
0.0	0.3000
50.0	0.7167
60.0	0.8000
100.0	0.6909
150.0	0.5545
200.0	0.4182
250.0	0.2818
280.0	0.2000
300.0	0.3884
350.0	0.8594
400.0	1.3304
450.0	1.8014
500.0	2.2724
556.0	2.800

Type: Temperature Profile
Name: a7.tp

Elevation (m)	Temperature (°C)
0.00	0.3000
50.00	0.7167
60.00	0.8000
100.00	0.6909
150.00	0.5545
200.00	0.6500
250.00	1.0500
280.00	2.0900
300.00	2.6100
320.00	5.3300
330.00	7.5500
340.00	10.6987
600.00	8.2987

Type: Transfer Characteristic **Name**: a7.tc	
Apparent Elevation (m)	Actual Elevation (m)
-233.60826	116.11
-183.65719	164.24
-133.70632	212.58
-83.75561	260.90
-33.80503	304.25
6.15537	331.82
16.14547	333.38
26.13556	332.41
66.09592	316.73
116.04637	296.49
141.02161	427.87
156.00676	553.15
165.99687	565.20
175.98697	575.19
215.94744	601.00

Type: Temperature Profile **Name**: a9.tp	
Elevation (m)	Temperature (°C)
0.00	0.3000
50.00	0.7167
60.00	0.8000
100.00	0.6909
150.00	0.5545
200.00	0.6500
250.00	0.9400
280.00	2.0000
290.00	2.3000
300.00	2.3000
340.00	7.6800
360.00	12.0000
380.00	10.0000
500.00	8.800

Type: Transfer Characteristic **Name**: a9.tc	
Apparent Elevation (m)	Actual Elevation (m)
-233.60826	115.54
-183.65719	163.37
-133.70632	211.96
-83.75561	262.84
-33.80503	304.04
6.15537	334.85
16.14547	337.36
26.13556	337.00
66.09592	316.44
116.04637	326.96
141.02161	327.21
156.00676	501.44
165.99687	545.70
175.98697	576.46
215.94744	592.19

Type: Temperature Profile **Name**: a10.tp	
Elevation (m)	Temperature (°C)
0.00	0.3000
50.00	0.7167
60.00	0.8000
100.00	0.6909
150.00	0.5545
200.00	0.5000
250.00	0.8000
280.00	1.5500
290.00	1.9500
300.00	2.2600
310.00	4.0300
320.00	6.3897
340.00	8.2500
350.00	12.1000
355.00	13.0560
600.00	10.606

Type: Transfer Characteristic **Name**: a10.tc	
Apparent Elevation (m)	Actual Elevation (m)
-233.60826	116.58
-183.65719	166.36
-133.70632	215.79
-83.75561	262.78
-33.80503	319.93
6.15537	333.84
16.14547	336.91
26.13556	338.75
66.09592	309.79
116.04637	286.74
141.02161	309.92
156.00676	336.22
165.99687	407.36
175.98697	440.05
215.94744	615.16

Type: Temperature Profile **Name**: a11.tp	
Elevation (m)	Temperature (°C)
0.00	0.3000
50.00	0.7167
60.00	0.8000
100.00	0.6909
150.00	0.5545
200.00	0.5000
250.00	0.8000
280.00	1.7800
290.00	1.9500
300.00	2.2590
310.00	4.5820
320.00	6.6290
325.00	6.8200
330.00	5.4600
335.00	8.7000
340.00	10.0000
342.00	11.3000
343.50	11.9500
344.50	12.9050

345.00	13.5500
347.50	14.4000
350.00	14.8120
600.000	12.3120

Type: Transfer Characteristic
Name: a11.tc

Apparent Elevation (m)	Actual Elevation (m)
-233.60826	114.27
-183.65719	164.42
-133.70632	213.88
-83.75561	264.48
-33.80503	319.49
6.15537	336.38
16.14547	338.98
26.13556	340.16
66.09592	314.59
116.04637	294.71
141.02161	304.66
156.00676	336.15
165.99687	516.29
175.98697	274.06
215.94744	619.50

Type: Temperature Profile
Name: a12.tp

Elevation (m)	Temperature (°C)
0.00	0.3000
50.00	0.7167
60.00	0.8000
100.00	0.6909
150.00	0.5545
200.00	0.4182
250.00	1.0800
265.00	1.2000
280.00	1.5000
290.00	1.9500
300.00	2.2600
310.00	2.8300
320.00	6.1950
330.00	7.4900

340.00	8.2800
350.00	10.7600
600.000	8.2600

Type: Transfer Characteristic
Name: a12.tc

Apparent Elevation (m)	Actual Elevation (m)
-233.60826	131.55
-183.65719	183.32
-133.70632	233.13
-83.75561	270.52
-33.80503	317.99
6.15537	338.58
16.14547	340.07
26.13556	339.27
66.09592	316.90
116.04637	298.17
141.02161	305.60
156.00676	336.14
165.99687	379.73
175.98697	390.97
215.94744	421.99

Type: Temperature Profile
Name: a13.tp

Elevation (m)	Temperature (°C)
0.000	0.300
50.000	6.990
60.000	5.450
100.000	4.900
150.000	4.410
200.000	3.530
210.000	3.560
225.000	2.690
227.500	2.520
230.000	1.980
240.000	1.600
250.000	1.300
260.000	-0.680
265.000	0.850
270.000	1.250

272.500	2.060
275.000	3.935
280.000	4.099
282.000	4.543
290.000	5.070
300.000	5.712
310.000	6.000
320.000	6.330
330.000	7.780
340.000	8.240
350.000	7.560
650.000	2.000

Type: Transfer Characteristic
Name: a13.tc

Apparent Elevation (m)	Actual Elevation (m)
-233.60826	117.77
-183.65719	162.77
-133.70632	212.64
-83.75561	256.96
-33.80503	304.25
6.15537	334.72
16.14547	337.05
26.13556	341.25
66.09592	315.42
116.04637	299.69
141.02161	311.17
156.00676	336.03
165.99687	352.98
175.98697	374.35
215.94744	441.48

Type: Temperature Profile
Name: a14.tp

Elevation (m)	Temperature (°C)
0.000	2.000
50.000	2.200
67.000	2.360
100.000	2.201
150.000	3.300
200.000	4.099

225.000	4.356
250.000	4.670
275.000	6.185
300.000	7.700
334.730	28.000
338.419	28.840
339.000	37.000
339.800	41.400
339.900	41.850
340.000	42.000
340.100	42.100
340.400	42.310
341.000	42.790
342.000	42.977
344.000	43.617
345.000	43.690
355.000	43.763
365.000	31.000
400.000	14.000

Type: Transfer Characteristic
Name: a14.tc

Apparent Elevation (m)	Actual Elevation (m)
-233.60826	117.48
-183.65719	165.99
-133.70632	212.86
-83.75561	254.33
-33.80503	304.02
6.15537	335.09
16.14547	339.88
26.13556	337.46
66.09592	313.61
116.04637	295.35
141.02161	310.01
156.00676	336.05
165.99687	353.78
175.98697	371.55
215.94744	441.27

Type: Temperature Profile **Name**: a17.tp	
Elevation (m)	Temperature (°C)
0.00	0.3000
50.00	0.7167
60.00	0.8000
100.00	0.6909
125.00	0.9000
150.00	1.5000
175.00	2.0000
200.00	2.5500
250.00	4.1000
280.00	4.1500
285.00	5.0500
290.00	6.7200
295.00	6.8000
300.00	6.8600
305.00	6.9000
310.00	6.9500
312.50	7.0100
315.00	7.1100
315.50	9.3000
316.00	9.0000
317.50	9.8500
320.00	10.7000
322.50	11.0800
325.00	11.4000
327.50	12.0000
330.00	12.9800
335.00	14.0000
340.00	15.0000
350.00	15.5000
355.00	17.0000
360.00	17.7000
600.00	15.3000

Type: Transfer Characteristic **Name**: a17.tc	
Apparent Elevation (m)	Actual Elevation (m)
-233.60826	117.13
-183.65719	158.56

-133.70632	214.28
-83.75561	258.46
-33.80503	308.69
6.15537	333.39
16.14547	335.91
26.13556	334.78
66.09592	320.44
116.04637	299.28
141.02161	296.74
156.00676	308.81
165.99687	418.47
175.98697	467.24
215.94744	595.76

Type: Temperature Profile
Name: a19a.tp

Elevation (m)	Temperature (°C)
0.00	0.3000
50.00	0.7167
60.00	0.8000
100.00	0.6909
150.00	1.2545
175.00	1.4864
200.00	1.4182
225.00	1.4496
250.00	1.8809
265.00	2.0955
280.00	1.0000
290.00	1.2221
300.00	1.2242
310.00	2.6563
320.00	3.8859
330.00	4.5805
340.00	5.0226
350.00	6.8097
360.00	11.5036
370.00	15.6478
400.00	31.8304
425.00	38.0659
450.00	34.3014
600.00	32.8014

Type: Temperature Profile
Name: a19b.tp

Elevation (m)	Temperature (°C)
0.00	0.3000
50.00	0.7167
60.00	0.8000
100.00	0.6909
150.00	1.9545
175.00	2.4864
200.00	2.4182
225.00	2.5491
250.00	3.4800
265.00	3.9500
280.00	2.2000
290.00	2.1500
300.00	2.0600
310.00	4.8300
320.00	7.1950
330.00	8.4900
340.00	9.2800
350.00	12.7600
360.00	22.0536
370.00	30.2478
400.00	60.3304
425.00	74.5659
450.00	68.8014
600.00	64.8014

Type: Transfer Characteristic
Name: a19.tc

Apparent Elevation (m)	Actual Elevation (m)
-233.60826	117.54
-183.65719	174.72
-133.70632	223.38
-83.75561	269.64
-33.80503	327.13
6.15537	344.41
16.14547	344.68
26.13556	343.28
66.09592	325.49
116.04637	301.92

141.02161	328.14
156.00676	396.79
165.99687	419.99
175.98697	517.41
215.94744	651.00

Type: Temperature Profile
Name: a20a.tp

Elevation (m)	Temperature (°C)
0.00	0.3000
50.00	0.7167
60.00	0.8000
100.00	0.6909
150.00	0.5545
200.00	0.4591
250.00	0.7409
280.00	0.9900
290.00	1.4069
300.00	1.8237
310.00	2.5323
315.00	2.9176
320.00	3.3029
325.00	3.4120
330.00	3.5855
332.50	3.7373
335.00	3.9841
337.50	4.3459
340.00	4.8826
342.00	5.2920
343.50	5.6241
344.50	5.8563
345.00	6.1812
347.50	6.1680
350.00	6.0857
355.00	5.7033
360.00	5.4768
400.00	4.4152
600.00	2.4152

Type: Temperature Profile
Name: a20b.tp

Elevation (m)	Temperature (°C)
0.00	0.3000
50.00	0.7167
60.00	0.8000
100.00	0.6909
150.00	0.5545
200.00	0.5000
250.00	1.2000
280.00	1.7800
290.00	2.5195
300.00	3.2590
310.00	4.5820
315.00	5.3055
320.00	6.0290
325.00	6.2000
330.00	6.5000
332.50	6.7800
335.00	7.2500
337.50	7.9500
340.00	9.0000
342.00	9.8000
343.50	10.4500
344.50	10.9050
345.00	11.5500
347.50	11.5000
350.00	11.3120
355.00	10.5000
360.00	10.0000
400.00	8.6000
600.00	6.6000

Type: Transfer Characteristic
Name: a20.tc

Apparent Elevation (m)	Actual Elevation (m)
-233.60826	129.00
-183.65719	180.45
-133.70632	234.60
-83.75561	276.66

-33.80503	319.18
6.15537	339.70
16.14547	342.09
26.13556	343.08
66.09592	337.91
116.04637	293.56
141.02161	436.55
156.00676	503.44
165.99687	554.10
175.98697	585.62
215.94744	612.73

Type: Temperature Profile
Name: a21a.tp

Elevation (m)	Temperature (°C)
0.00	0.3000
50.00	0.7167
60.00	0.8000
100.00	0.6909
150.00	0.7273
200.00	0.7591
250.00	0.7909
265.00	0.8205
280.00	0.8500
290.00	1.1471
295.00	1.4207
300.00	1.5942
305.00	1.8178
310.00	2.0913
315.00	2.4153
320.00	2.8884
325.00	3.5120
330.00	4.2355
340.00	5.3826
350.00	6.6297
400.00	6.1652
450.00	5.4007
500.00	4.6362
556.00	4.3400
600.00	3.7128

Type: Temperature Profile
Name: a21b.tp

Elevation (m)	Temperature (°C)
0.00	0.3000
50.00	0.7167
60.00	0.8000
100.00	0.6909
150.00	0.9000
200.00	1.1000
250.00	1.3000
265.00	1.4000
280.00	1.5000
290.00	2.0000
295.00	2.5000
300.00	2.7000
305.00	3.2000
310.00	3.7000
315.00	4.3000
320.00	5.2000
325.00	6.4000
330.00	7.8000
340.00	10.0000
350.00	12.4000
400.00	11.0000
450.00	10.5000
500.00	10.0000
556.00	9.4600
600.00	9.0000

Type: Transfer Characteristic
Name: a21.tc

Apparent Elevation (m)	Actual Elevation (m)
-233.60826	112.61
-183.65719	174.61
-133.70632	227.97
-83.75561	283.11
-33.80503	325.05
6.15537	339.37
16.14547	339.94
26.13556	339.14
66.09592	322.51

116.04637	296.58
141.02161	307.90
156.00676	324.53
165.99687	469.40
175.98697	510.42
215.94744	615.88

Type: Temperature Profile
Name: b3a.tp, b4a.tp, b5a.tp, b6a.tp, b11a.tp

Elevation (m)	Temperature (°C)
0.0	0.2720
60.0	0.0000
100.0	-0.2400
125.0	-0.3900
150.0	-0.5400
175.0	-0.6900
200.0	-0.8400
225.0	-0.9900
250.0	-1.1400

Type: Temperature Profile
Name: b3b.tp

Elevation (m)	Temperature (°C)
0.0	-0.2600
10.0	-0.3200
20.0	-0.3800
30.0	-0.4400
40.0	-0.5000
50.0	-0.5600
55.0	-0.5700
60.0	-0.7700
62.5	-0.8350
65.0	-0.8300
70.0	-0.7700
75.0	-0.6800
80.0	-0.5900
85.0	-0.4200
90.0	-0.1700
92.8	0.0500

100.0	0.8500
111.0	1.6000
120.0	2.4000
130.0	3.2500
250.0	2.6500

Type: Transfer Characteristic
Name: b3.tc

Apparent Elevation (m)	Actual Elevation (m)
-211.14	109.94
-208.18	117.68
-205.06	118.11
-202.98	118.13
-197.78	116.19
-192.58	113.34
-190.50	112.54
-187.38	114.23
-182.18	116.37
-176.98	118.05
-171.78	119.88
-166.58	120.78
-161.38	119.45
-156.19	119.85
-150.98	118.22
-145.79	117.06
-140.59	112.93
-137.47	111.39
-135.39	113.44
-130.19	125.97
-124.99	139.21
-119.79	150.54
-114.59	157.90
-109.39	165.76
-104.19	174.08
-98.99	183.42
-93.79	193.87

Type: Temperature Profile
Name: b4b.tp

Elevation (m)	Temperature (°C)
0.0	0.3800
10.0	0.3500
20.0	0.3200
30.0	0.2900
40.0	0.2600
50.0	0.2300
55.0	0.2000
60.0	0.1700
62.5	0.1500
65.0	0.1400
70.0	-0.0600
75.0	0.0150
80.0	0.1100
85.0	0.2400
90.0	0.3750
95.0	0.5850
99.5	0.8550
105.0	1.3800
110.0	1.9000
120.0	2.7000
130.0	3.5000
140.0	3.6000
250.0	3.0500

Type: Transfer Characteristic
Name: b4.tc

Apparent Elevation (m)	Actual Elevation (m)
-211.14	110.47
-208.18	116.34
-205.06	119.73
-202.98	119.26
-197.78	113.29
-192.58	111.54
-190.50	112.60
-187.38	113.77
-182.18	116.42
-176.98	117.86

-171.78	120.23
-166.58	121.71
-161.38	119.99
-156.19	120.83
-150.98	118.14
-145.79	116.38
-140.59	112.60
-137.47	115.44
-135.39	117.04
-130.19	123.48
-124.99	138.19
-119.79	150.64
-114.59	159.88
-109.39	170.27
-104.19	181.73
-98.99	194.23
-93.79	208.55

Type: Temperature Profile
Name: b5b.tp

Elevation (m)	Temperature (°C)
0.0	-1.30
7.0	-1.40
12.0	-2.00
15.0	-2.20
18.0	-2.24
22.0	-2.26
24.0	-2.26
26.0	-2.44
30.0	-2.44
35.0	-2.55
40.0	-2.55
50.0	-0.30
55.0	0.00
60.0	0.10
70.0	0.17
80.0	0.24
90.0	0.31
100.0	0.40
110.0	0.35
130.0	0.25

150.0	0.15
170.0	0.05
190.0	-0.05
210.0	-0.15
230.0	-0.25
250.0	-0.35

Type: Transfer Characteristic
Name: b5.tc

Apparent Elevation (m)	Actual Elevation (m)
-211.14	111.14
-208.18	114.21
-205.06	117.73
-202.98	120.34
-197.78	118.28
-192.58	114.44
-190.50	114.10
-187.38	113.23
-182.18	115.02
-176.98	117.65
-171.78	120.36
-166.58	120.48
-161.38	119.36
-156.19	120.53
-150.98	118.95
-145.79	116.11
-140.59	113.70
-137.47	111.21
-135.39	115.57
-130.19	141.43
-124.99	162.49
-119.79	181.00
-114.59	195.24
-109.39	203.13
-104.19	210.33
-98.99	216.35
-93.79	222.16

Type: Temperature Profile **Name**: b6b.tp	
Elevation (m)	Temperature (°C)
0.0	0.36
10.0	0.30
20.0	0.24
30.0	0.18
40.0	0.12
50.0	0.06
60.0	0.00
70.0	-0.44
80.0	-0.45
85.0	-0.60
90.0	-0.52
94.0	-0.43
98.0	-0.33
100.0	-0.22
105.0	-0.15
107.0	0.065
109.0	0.15
111.0	0.10
112.0	0.37
115.0	0.63
120.0	1.15
125.0	1.62
130.0	2.05
140.0	2.80
150.0	3.45
250.0	2.95

Type: Transfer Characteristic **Name**: b6.tc	
Apparent Elevation (m)	Actual Elevation (m)
-211.14	111.86
-208.18	115.42
-205.06	118.48
-202.98	118.02
-197.78	114.87
-192.58	112.14
-190.50	113.30
-187.38	113.48

-182.18	116.51
-176.98	118.52
-171.78	120.71
-166.58	122.08
-161.38	117.88
-156.19	121.11
-150.98	119.45
-145.79	117.29
-140.59	113.00
-137.47	114.57
-135.39	117.07
-130.19	127.75
-124.99	137.42
-119.79	149.88
-114.59	156.74
-109.39	165.91
-104.19	181.33
-98.99	192.45
-93.79	205.64

Type: Temperature Profile
Name: b11b.tp

Elevation (m)	Temperature (°C)
0.0	0.75
10.0	0.65
20.0	0.55
30.0	0.45
40.0	0.35
50.0	0.25
55.0	0.20
60.0	0.15
62.5	0.125
65.0	0.10
70.0	-0.05
75.0	0.00
80.0	0.08
85.0	0.18
90.0	0.32
93.5	0.44
96.0	0.585
99.5	0.855

105.0	1.38
110.0	1.90
120.0	2.70
130.0	3.50
250.0	2.30

Type: Transfer Characteristic
Name: b11.tc

Apparent Elevation (m)	Actual Elevation (m)
-211.14	109.98
-208.18	113.30
-205.06	117.67
-202.98	118.01
-197.78	115.36
-192.58	112.80
-190.50	113.74
-187.38	114.18
-182.18	117.01
-176.98	118.56
-171.78	120.23
-166.58	120.48
-161.38	120.70
-156.19	119.21
-150.98	115.19
-145.79	113.10
-140.59	111.12
-137.47	113.95
-135.39	115.75
-130.19	119.90
-124.99	132.63
-119.79	148.61
-114.59	157.99
-109.39	168.55
-104.19	180.26
-98.99	193.03
-93.79	206.95

Appendix B

Ray Tracing and Graphing Program

This ray tracing and graphing program developed for research on this thesis consists of three parts: *mgraph.h, mgraph.c, makeray.c.* They were compiled using Borland C++ version 4.0 and are included in this appendix for reference.

This is a DOS program requiring version 3.0 or later to execute. A VGA graphics card is also needed. To run this program the name *multiray* must be entered at the command prompt.

Upon execution the user is presented with the following menu:

Please select one of the following atmosphere types for ray tracing.
1: Flat Atmosphere.
2: Wavy Atmosphere.
3: Logistic Atmosphere.
4: Multiple Region Atmosphere.
5: Modify Previous Atmosphere.
6: Quit.
Option:

A menu item is selected by hitting the appropriate number key (1-6). The *Flat Atmosphere* selection will prompt the user to enter a single atmosphere temperature profile file (*.tp*). The *Wavy Atmosphere* prompts the user for a single wavy atmosphere file (*.atm*). The next selection prompts the user for a logistic atmosphere file (*.log*), this option was not used in this thesis. The *Multiple Region Atmosphere* selection prompts the user for an atmosphere region file (*.arf*). The *Modify* option causes the previous simulation to execute.

Once a file name has been entered, the user is prompted for the slope of the atmosphere or atmospheres represented by each file. The slope is entered in arc minutes and is zero for the flat atmosphere. After the slope is entered, the rays defined by the loaded file

are traced through the atmosphere and automatically graphed.

The graphing routine also allows the user to view any transfer characteristic within the range covered by the rays. Both the rays and transfer characteristic data can be saved to files. Another convenient function allows the user to scale the x and y axis to examine the rays in greater detail.

```
// MM     MM  GGGGGGG RRRRRRR     AAAA    PPPPPPP  HH     HH       HH     HH
// MMM   MMM GGGGGGGG RRRRRRRR   AA  AA   PPPPPPPP HH     HH       HH     HH
// MMMMMMMMM GG       RR     RR AA    AA  PP     PP HH     HH       HH     HH
// MMMMMMMMM GG  GGGG RRRRRRRR AAAAAAAA PPPPPPPP HHHHHHHHH       HHHHHHHHH
// MM MM MM GG  GGGG RRRRRRR  AAAAAAAA PPPPPPP  HHHHHHHHH       HHHHHHHHH
// MM     MM GG    GG RR  RR   AA     AA PP       HH     HH       HH     HH
// MM     MM GGGGGGGG RR   RR  AA     AA PP       HH     HH  ..   HH     HH
// MM     MM  GGGGGGG RR    RR AA     AA PP       HH     HH  ..   HH     HH

// Graphing program based on a
// Modification of EDITTC Program...  December 29, 1994

/*************************************************************************/
/************************** TYPE DEFINITIONS *****************************/
/*************************************************************************/
typedef struct {
   char name[80];       // Name of primary ray file
   double time;         // Time for atmospheric waves (seconds)
   double slope;        // Slope of atmosphere (arcmin)
   double eyeLevel;     // Eye level of observer (metres)
   double xstep;        // Horizontal step size (metres)
   double xmax;         // Maximum horizontal value (metres)
   double ymax;         // Maximum vertical value (metres)
   int     numberRays; // Number of Rays
   double *angles;      // Angles of rays (arcminutes) <- numberRays
   double **ray;        // An array of ray elevations (metres) <-
                       // numberRays x (xmax/xstep+1)
   double graphxmin;    // Minimum value of x used to graph rays
   double graphxmax;    // Maximum value of x used to graph rays
   double graphymin;    // Minimum value of y used to graph rays
   double graphymax;    // Maximum value of y used to graph rays
} rayType;

/*************************************************************************/
/************************** MAIN INCLUDE FILE ****************************/
/*************************************************************************/
void GraphData(rayType *rayList);
int ValidFileName(char *file,char *extension,char readWrite);
void GetLine(FILE *IN,char *fileline);
```

```
// MM     MM   AAAA   KK    KK EEEEEEEE RRRRRRR    AAAA   YY    YY     CCCCCC
// MMM   MMM  AA  AA  KK   KK  EEEEEEEE RRRRRRRR  AA  AA   YY  YY     CCCCCCCC
// MMMMMMMM AA    AA KK  KK   EE       RR    RR AA    AA   YYYY      CC
// MMMMMMMM AAAAAAAA KKKKK    EEEEEE   RRRRRRRR AAAAAAAA    YY       CC
// MM MM MM AAAAAAAA KKKKK    EEEEEE   RRRRRRR  AAAAAAAA    YY       CC
// MM     MM AA    AA KK  KK   EE       RR  RR   AA    AA    YY       CC
// MM     MM AA    AA KK   KK  EEEEEEEE RR   RR  AA    AA    YY    .. CCCCCCCC
// MM     MM AA    AA KK    KK EEEEEEEE RR    RR AA    AA    YY    ..  CCCCCC

/*************************************************************************/
/* Modification of the hexray.c program written by Wayne Silvester       */
/*************************************************************************/
/*                                                                       */
/* Date of Changes:  January 4, 1995  (by Thomas L. Legal)               */
/* Date of Changes:  March 15, 1995 (by Thomas L. Legal)                 */
/*                                                                       */
/*************************************************************************/

/*************************************************************************/
/********************************* Includes ******************************/
/*************************************************************************/
#pragma hdrfile "makeray.sym"
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <dos.h>
#include <dir.h>
#include <ctype.h>

#include "mgraph.h"


/*************************************************************************/
/********************************** Defines ******************************/
/*************************************************************************/
#define MAXRAYS  50
#define MAXTP       50
#define ERR         0.001         //error for Newton method for layer interception
#define BETA     0.00348
#define GRAV     9.80665          // Acceleration due to gravity
#define EPSILON  226e-06
#define K           273.15        // to convert Celsius to Kelvin
#define KEARTH   1.5699e-07       // curvature of earth
#define PI          3.141592654

#define boolean int      // Set up the boolean variable type
#define TRUE 1           // Set true to 1
#define FALSE 0          // Set false to 0


/*************************************************************************/
/***************************** Type Definitions **************************/
/*************************************************************************/
typedef struct {
   double x;
   double z;
} xz;

typedef struct {
   double z;         // Current Elevation
   double angle;     // Current Angle
   int grounded; // flag =1 if ray grounded in previous region(s)
   int spaced;   // flag =1 if ray exceeded max. z in previous region(s)
```

```
} s;

typedef struct {
   int numsteps;
   int flat;
   int wavy;
   char filename[80];
} ar;


/*************************************************************************/
/************************ Function Prototypes ****************************/
/*************************************************************************/
// Data acquisition functions
boolean GetFileName(char *fileName,char *fileType);
void LoadTempProfile(char *filename);
void LoadAtm(char *filename, int logistic);
void LoadArf(char *filename);

// Function for adjusting global variables
void EditGlobalVariables(void);
void RayAngleMenu(void);
void tpMenu(void);
void atmMenu(void);
void arfMenu(void);

// Functions for a sloped atmosphere
void SetRays(double slope, int startPos, int steps);
void ResetRays(double slope, int startPos, int steps);

// Functions called by the main program
void CalcPressDen(void);
void RayTrace(int numsteps);
rayType *makeRayType(char *fileName);

// Other functions
xz Intersection(double xo,double x,double a,double b,double c,int layer, int up);
double Curvature(double x,double z,double angle);
int Locate(double x, double z);
double Boundary(int layer, double x);
double Exp(double x);
void GetLine(FILE *IN,char *fileline);
int Input(int *intptr,double *fltptr,char *charptr,char *ext);


/*************************************************************************/
/************************** Global Variables *****************************/
/*************************************************************************/
double zeye,dx,xmax,zmax;          // TP Data
int nrays;                         // TP Data
double angle[MAXRAYS];             // Ray Angles in radians
int numLevels;                     // TP Data
double tpElev[MAXTP];              // TP Data
double tpTemp[MAXTP];              // TP Data

double Ah,w,k;                     // ATM/LOG Data
double A,B;                        // LOG Data
double mag[MAXTP];                 // ATM/LOG Data

int numregion;                     // ARF Data
ar atmregion[20];                  // ARF Data

double slope;                      // Slope of flat atm in minutes

int numpoints;                     // Number of points in profile
```

```
double den[MAXTP];                  // Densities at various levels
double *ray[MAXRAYS];               // The actual rays calculated
double waveTime;                    // Time of wave... user input

s start[MAXRAYS];                   // to hold position/angle data for rays
int prevsteps;                      // number of steps done so far

int flat;   // flag: =1 if flat atmo
int wavy;   // flag: =1 if wavy atmo; if flat=0 and wavy=0 then logistic atmo
int combo;  // flag: =1 if combo atmo; =0 otherwise


//*************************************************************************//
//*                                                                       *//
//*                              Main Program                             *//
//*                                                                       *//
//*************************************************************************//
void main(void)
{
   char ch;                     // Character read in for a menu selection
   char filename[80];           // The name of a file to be read in
   boolean newSelection;        // TRUE if a new selection is requested
   boolean quit;                // Value if quit is selected
   boolean previous;            // If this is true a previous file is loaded
   int i;                       // Counter variable

   // Initialize Varaibles
   newSelection = TRUE;
   quit = FALSE;
   previous = FALSE;

   // Initialize Global Variables
   zeye = 0;                                    // Horizontal eye elevation (metres)
   dx = 0;                                      // Step size in meters
   xmax = 0;                                    // Maximum horizontal range
   zmax = 0;                                    // Maximum ray elevation
   nrays = 0;                                   // Number of rays in ray tracing
   for (i=0; i<MAXRAYS; i++) angle[i]=0;        // The inital angles of all rays
   numLevels = 0;                               // Number of levels in temp. profile
   for (i=0; i<MAXTP; i++) tpElev[i]=0;         // Temp. Profile Elevation Data
   for (i=0; i<MAXTP; i++) tpTemp[i]=0;         // Temp. Profile Temperature Data
   Ah = 0;                                      // Magnitude multiplication factor
   w = 0;                                       // Angular Frequency (rad/m)
   k = 0;                                       // Horizontal Wave Number (m)
   A = 0;                                       // Centre of Log function (m)
   B = 0;                                       // Width of Log funciton (m)
   for (i=0; i<MAXTP; i++) mag[i]=0;            // Wave Magnitude at levels
   numregion = 0;                               // Number of Regions (in multi atmos)
   for (i=0; i<20; i++) {
      atmregion[i].numsteps = 0;                // Step in each region
      atmregion[i].flat = 0;                    // Is current region flat
      atmregion[i].wavy = 0;                    // Is current region wavy
      strcpy(atmregion[i].filename,"");         // Name of region file
   }

   // Keep looping through menu
   do {
      // Clear screen and print menu
      clrscr();
      printf("\r\n");
      printf(" Please select one of the following atmosphere types for\
               raytracing.\r\n\r\n");
      printf("  1: Flat Atmosphere.\r\n");
      printf("  2: Wavy Atmosphere.\r\n");
      printf("  3: Logistic Atmosphere.\r\n");
```

```
    printf("  4: Multiple Region Atmosphere.\r\n");
    printf("  5: Modify Previous Atmosphere.\r\n");
    printf("  6: Quit.\r\n\r\n");
    printf(" Option: ");

    // Get a selection from the user
    do {
       ch=getch();
    } while (((ch<'1')||(ch>'6'))&&(ch!='q'));

    // Set variables depending on menu selection
    switch (ch) {
       case '1':
          printf("Flat.\r\n\r\n");
          if ((newSelection = GetFileName(filename,"tp"))==FALSE) {
             flat=1; wavy=0; combo=0;
             LoadTempProfile(filename);
          }
          break;
       case '2':
          printf("Wavy.\r\n\r\n");
          if ((newSelection = GetFileName(filename,"atm"))==FALSE) {
             flat=0; wavy=1; combo=0;
             LoadAtm(filename,0);
          }
          break;
       case '3':
          printf("Logistic.\r\n\r\n");
          if ((newSelection = GetFileName(filename,"log"))==FALSE) {
             flat=0; wavy=0; combo=0;
             LoadAtm(filename,1);
          }
          break;
       case '4':
          printf("Combo.\r\n\r\n");
          if ((newSelection = GetFileName(filename,"arf"))==FALSE) {
             flat=0; wavy=0; combo=1;
             LoadArf(filename);
          }
          break;
       case '5':
          if (previous == TRUE) {
             newSelection = FALSE;
             EditGlobalVariables();
          }
          else {
             printf("No previous file exists.\r\n");
             newSelection = TRUE;
             sleep(2);
          }
          break;
       case '6':
          quit = TRUE;
          break;
       default:
          quit = TRUE;
          break;
    }

    // If we are ready to calculate the rays
    if ((newSelection == FALSE) && (quit != TRUE)) {
       // Calculate number of points in set of rays
       numpoints = xmax/dx+1;

       // Change zmax to highest tp Elevation
```

```
 if (combo==0) if (zmax<tpElev[numLevels-1]) zmax=tpElev[numLevels-1];


 /*************************************************************************/
 /********** Allocate Ray Memory and Set Starting Conditions ***********/
 /*************************************************************************/
 for (i=0; i<nrays; i++) {
    if (ray != NULL) {
       free(ray[i]);
       ray[i] = NULL;
    }
    ray[i] = (double *) calloc(numpoints,sizeof(double));
    ray[i][0] = zeye;
    start[i].z = zeye;
    start[i].angle = angle[i];
    start[i].grounded = 0;
    start[i].spaced = 0;
 } // end for

 /*************************************************************************/
 /********** Do Ray Trace for the type of atmosphere selected **********/
 /*************************************************************************/
 //################## flat atmosphere ##################//
 if ((combo==0)&&(flat==1)) {
    prevsteps=0;
    printf("\r\n");
    printf(" Please enter slope of atmosphere in arc minutes: ");
    Input(NULL,&slope,NULL,NULL);
    printf("\r\n\r\n");
    SetRays(slope,prevsteps,numpoints-1);
    CalcPressDen();
    RayTrace(numpoints-1);
    ResetRays(slope,prevsteps,numpoints-1);
    gotoxy(1,wherey());
    clreol();
    GraphData(makeRayType(filename));
 }

 //############ wavy or logistic atmosphere ############//
 if ((combo==0)&&(flat==0)) {
    prevsteps=0;
    printf("\r\n");
    printf(" Please enter slope of atmosphere in arc minutes: ");
    Input(NULL,&slope,NULL,NULL);
    printf("\r\n\r\n");
    double period=2*PI/w;
    printf(" Period of waves is: ");
    sprintf(filename,"%-8.2f",period);
    for (i=strlen(filename);i>0;i--)
       if (filename[i]==' ') filename[i]=NULL;
    printf("%s",filename);
    printf(" s. Please enter time (in seconds): ");
    Input(NULL,&waveTime,NULL,NULL);
    printf("\r\n\r\n");
    SetRays(slope,prevsteps,numpoints-1);
    CalcPressDen();
    RayTrace(numpoints-1);
    ResetRays(slope,prevsteps,numpoints-1);
    gotoxy(1,wherey());
    clreol();
    GraphData(makeRayType(filename));
 }

 //############ multiple region atmosphere ############//
 if (combo==1) {
```

```
                prevsteps=0;
                // Check to see if there is any wavy regions and if so get the time
                // that we are calculating the rays from (Get the Phase 0 - in phase)
                flat=1;
                for (i=0;i<numregion;i++)
                    if (atmregion[i].flat==0) flat=0;
                if (flat==0) {
                    printf("\r\n Please enter time (in seconds): ");
                    Input(NULL,&waveTime,NULL,NULL);
                    printf("\r\n");
                }

                /* perform raytrace */
                for (i=0;i<numregion;i++) {
                    // Get the sloped of each atmosphere
                    printf("\r\n\r\n");
                    printf(" Please enter slope of atmospheric region %d in arc
minutes: "
                                ,i);
                    Input(NULL,&slope,NULL,NULL);
                    printf("\r\n");
                    // Flat
                    if (atmregion[i].flat==1) {
                        flat=1;
                        wavy=0;
                        LoadTempProfile(atmregion[i].filename);
                    }
                    // Wavy
                    if (atmregion[i].wavy==1) {
                        wavy=1;
                        flat=0;
                        LoadAtm(atmregion[i].filename,0);
                    }
                    // Logistic
                    if ((atmregion[i].flat==0)&&(atmregion[i].wavy==0)) {
                        flat=0;
                        wavy=0;
                        LoadAtm(atmregion[i].filename,1);
                    }
                    // Add angle to rays if necessary
                    SetRays(slope,prevsteps,atmregion[i].numsteps);

                    // Output a progress message
                    printf(" Region ");
                    gotoxy(9,wherey());
                    clreol();
                    printf("%d",i);
                    printf(":");

                    // Trace Rays
                    CalcPressDen();
                    RayTrace(atmregion[i].numsteps);

                    // Remove slope from resulting atmosphere (to the eye the atmos
                    // is flat)
                    ResetRays(slope,prevsteps,atmregion[i].numsteps);

                    // Set the number of steps preformed so far.
                    prevsteps+=atmregion[i].numsteps;
                }
                gotoxy(1,wherey());
                clreol();

                // Call the graphing program
                GraphData(makeRayType(filename));
```

```
            }       // End if (combo atmosphere

            // Set variable to indicate that on atmosphere has been processed
            previous = TRUE;
        }
    } while (quit == FALSE);

    // Print Exit Message
    printf("Quit.\r\n\r\n");
    sleep(1);

    // Clean up the ray data
    for (i=0; i<nrays; i++) {
        if (ray != NULL) {
            free(ray[i]);
            ray[i] = NULL;
        }
    }

    // Exit on normal condition
    exit(0);
}


//*************************************************************************//
//*                                                                       *//
//*                          Data Access Functions                        *//
//*                                                                       *//
//*************************************************************************//

//****************************** GetFileName ******************************//
//*                                                                       *//
//*  This function gets a filename from the user.  The function checks    *//
//*  if the file exists and if so returns the name.                       *//
//*                                                                       *//
//*************************************************************************//
boolean GetFileName(char *fileName,char *fileType) {
    FILE *fp;                // Pointer to a file
    int x,y;                 // Location of the cursor
    boolean escape;          // Indicates TRUE when the user hits escape
    boolean validInput;      // Indicates TRUE when a valid file name is entered
    char tempFileName[80];   // Temporary variable to hold a file name

    x=wherex();   // Get the x location of the cursor
    y=wherey();   // Get the y location of the cursor

    // Loop until a valid file name is entered or the user hits the escape key
    do {
        gotoxy(x,y); // Locate the cursor correctly
        switch (fileType[1]) {
            case 'p': //Load TP
                printf(" Enter file name of temperature profile file (ESC to cancel):
");
                break;
            case 't': //Load ATM
                printf(" Enter file name of atmosphere boundary file (ESC to cancel):
");
                break;
            case 'o': //Load LOG
                printf(" Enter file name of logistic atmosphere file (ESC to cancel):
");
                break;
            case 'r': //Load ARF
                printf(" Enter file name of atmosphere region file (ESC to cancel):
");
```

```
            break;
        }
        clreol();     // Clean up input line
        escape = Input(NULL,NULL,tempFileName,fileType);  // Get name from user
        if (escape == FALSE) {
            if ((fp=fopen(tempFileName,"r"))==NULL) {      // input is not valid
                printf("\r\n File not found.  Hit a key.");
                getch();
                gotoxy(x,y+1);
                clreol();
                escape = TRUE;
            }
            else validInput = TRUE;  // Input is valid
        }
    } while ((validInput == FALSE) && (escape == FALSE));

    // Clean up screen
    gotoxy(x,y+1);
    clreol();

    // Close the file used to test if the file name entered was valid
    fclose(fp);

    // Copy user entered file name to be return to main program
    strcpy(fileName,tempFileName);

    // Return whether the escape key was hit
    return (escape);
}


//***************************** LoadTempProfile ****************************//
//*                                                                        *//
//*  Purpose: Reads data from a temperature profile (.TP) file.            *//
//*  Parameters: *filename - name of .TP file.                             *//
//*  Return: None.                                                         *//
//*                                                                        *//
//**************************************************************************//
void LoadTempProfile(char *filename)
{
   FILE *IN;               // File data is loaded from
   int i;                  // Counter Variable
   int    dummyInt;        // Dummy variable to read ints
   double dummyDouble;     // Dummy variable to read doubles
   char dummyString[80];   // Dummy variable to read strings

   // Open File described by filename
   IN=fopen(filename,"r");

   /*** Read data from temperature profile file ***/
   fscanf(IN,"%lf\n",&dummyDouble);         // Eye elevation in metres
   if (combo==0) zeye=dummyDouble;

   fscanf(IN,"%lf\n",&dummyDouble);         // Horizontal Step Size (KM)
   if (combo==0) dx=1000.0*dummyDouble;     // Convert step size to metres

   fscanf(IN,"%d\n",&dummyInt);             // Print Interval (not used)

   fscanf(IN,"%lf\n",&dummyDouble);         // Maximum horizontal range
   if (combo==0) xmax=dummyDouble;

   fscanf(IN,"%lf\n",&dummyDouble);         // Maximum ray elevation
   if (combo==0) zmax=dummyDouble;

   fscanf(IN,"%d\n",&dummyInt);             // number of rays
```

```
    if (combo==0) nrays=dummyInt;

    for ( i=0; i<dummyInt; i++ ) {          // initial ray elevations (arcmin)
       fscanf(IN,"%lf\n",&dummyDouble);
       if (combo==0) angle[i]=dummyDouble/3437.747;
    }

    fgets(dummyString,80,IN);               // Title of temperature profile

    fscanf(IN,"%d\n",&numLevels);           // Number of levels in temp. profile
    for (i=0; i<numLevels; i++)   {
       fscanf(IN,"%lf %lf\n",&tpElev[i],&tpTemp[i]);  // TP elevations, temps.
    }

    // Close the file described by filename
    fclose(IN);
}


//****************************** LoadAtm ******************************//
//*                                                                     *//
//*  Purpose: Reads data from atmosphere boundary file (.ATM or .LOG).  *//
//*  Parameters: *filename - name of .ATM file.                         *//
//*              logistic - flag: =1 if loading .LOG file.              *//
//*  Return: None.                                                      *//
//*                                                                     *//
//***********************************************************************//
void LoadAtm(char *filename, int logistic)
{
    FILE *IN;                   // Variable to hold file pointer
    int    dummyInt;            // Dummy variable to read ints
    double dummyDouble;         // Dummy variable to read doubles
    char dummyString[80];       // Dummy variable to read strings
    int i;                      // Counter Variable

    // Open File described by filename
    IN=fopen(filename,"r");

    /*** Read data from atmosphere boundary file ***/
    GetLine(IN,dummyString);
    sscanf(dummyString,"%d",&dummyInt);         // number of layers (Not used)

    GetLine(IN,dummyString);
    dummyString[strlen(dummyString)-1] = NULL;  // Remove '\n' character
    LoadTempProfile(dummyString);               // Load Temp Profile

    GetLine(IN,dummyString);
    sscanf(dummyString,"%lf",&Ah);              // Magnitude Multipication Factor

    GetLine(IN,dummyString);
    sscanf(dummyString,"%lf",&w);               // Angular Frequency

    GetLine(IN,dummyString);
    sscanf(dummyString,"%lf",&k);               // Horizontal Wave Number

    // If function is logistic, read two addition variables
    if (logistic==1) {
       GetLine(IN,dummyString);
       sscanf(dummyString,"%lf",&B);            // Centre of Log function

       GetLine(IN,dummyString);
       sscanf(dummyString,"%lf",&B);            // Width of Log function
    }

    // Read in Data into the Wave Magnitude array
```

```
    for (i=1; i<numLevels; i++) {
        GetLine(IN,dummyString);
        sscanf(dummyString,"%lf",&dummyDouble);
        mag[i]=dummyDouble;
    }

    // Close the file described by filename
    fclose(IN);
}


//****************************** LoadArf ******************************//
//*                                                                     *//
//*  Purpose: Reads data from atmosphere boundary file (.ARF)           *//
//*  Parameters: *filename - name of .ARF file.                         *//
//*  Return: None.                                                      *//
//*                                                                     *//
//*********************************************************************//
void LoadArf(char *filename) {
    FILE *IN;                   // Variable to hold file pointer
    int    dummyInt;            // Dummy variable to read ints
    double dummyDouble;         // Dummy variable to read doubles
    char dummyString[80];       // Dummy variable to read strings
    int i;                      // Counter Variable

    // Open file pointer for reading ARF
    IN=fopen(filename,"r");

    /*** Read data from Atmosphere Region File (ARF) file ***/
    GetLine(IN,dummyString);
    sscanf(dummyString,"%d\n",&numregion);      // number of regions

    GetLine(IN,dummyString);
    sscanf(dummyString,"%lf\n",&zeye);          // Eye elevation in metres

    GetLine(IN,dummyString);
    sscanf(dummyString,"%lf\n",&dummyDouble);   // Horizontal Step Size (KM)
    dx=1000.0*dummyDouble;                      // Convert step size to metres

    GetLine(IN,dummyString);
    sscanf(dummyString,"%lf\n",&xmax);          // Maximum horizontal range

    GetLine(IN,dummyString);
    sscanf(dummyString,"%lf\n",&zmax);          // Maximum ray elevation

    GetLine(IN,dummyString);
    sscanf(dummyString,"%d\n",&nrays);          // number of rays

    for (i=0; i<nrays; i++) {                   // initial ray elevs (arcmin)
        GetLine(IN,dummyString);
        sscanf(dummyString,"%lf\n",&dummyDouble);
        angle[i]=dummyDouble/3437.747;
    }

    // Read in data for each of the regions
    for (i=0;i<numregion;i++) {
        GetLine(IN,dummyString);
        sscanf(dummyString,"%d\n",&dummyInt);
        atmregion[i].numsteps = dummyInt;       // Number of Steps in Region
        GetLine(IN,dummyString);                // Read Type of Region
        if (toupper(dummyString[0])=='F') {
            atmregion[i].flat=1;
            atmregion[i].wavy=0;
        }
        if (toupper(dummyString[0])=='W') {
```

```
            atmregion[i].flat=0;
            atmregion[i].wavy=1;
        }
        if (toupper(dummyString[0])=='L') {
            atmregion[i].flat=0;
            atmregion[i].wavy=0;
        }
        GetLine(IN,dummyString);                        // Get file name
        dummyString[strlen(dummyString)-1] = NULL;  // Remove '\n' character
        strcpy(atmregion[i].filename,dummyString);
    }

    // Close the file described by filename
    fclose(IN);
}


//*************************************************************************//
//*                                                                       *//
//*                      Variable Adjustment Functions                    *//
//*                                                                       *//
//*************************************************************************//

//************************** EditGlobalVariables ************************//
//*                                                                       *//
//* Purpose: This function allows the user to change various variable   *//
//*          from a previous run of the program.  The user can then     *//
//*          execute the program again with these changes.              *//
//*                                                                       *//
//*************************************************************************//
void EditGlobalVariables(void) {
   boolean quit = FALSE;      // Equal TRUE when user wants to quit
   char ch;                   // Input character

   // Keep looping through menu
   do {
      // Clear screen and print menu
      clrscr();
      printf("\r\n");
      printf(" Please select one of the following variable to adjust.\r\n");
      printf(" (Press <enter> the return to ray tracing)\r\n\r\n");
      printf("  1: Horizontal Eye Elevation  zeye = %.2lf m\r\n",zeye);
      printf("  2: Horizontal Step Size      dx = %.2lf m\r\n",dx);
      printf("  3: Maximum Horizontal Range  xmax = %.2lf m\r\n",xmax);
      printf("  4: Maximum Ray Elevation     zmax = %.2lf m\r\n",zmax);
      printf("  5: Ray Angles\r\n");
      if (((combo == 0) && (wavy == 1)) ||
          ((combo == 0) && (wavy == 0) && (flat == 0)))
         printf("  6: Wavy Atmosphere Data\r\n");
      else if (combo == 0) printf("  6: Temperature Profile Data\r\n");
      else printf("  6: Atmospheric Region File Data\r\n");
      printf("\r\n Option: ");

      // Get a selection from the user
      do {
         ch=getch();
      } while (((ch<'1')||(ch>'6'))&&(ch!=13));

      // Print user selection
      printf("%c\n",ch);

      // Set variables depending on menu selection
      switch (ch) {
         case '1':
            printf("\r\nEnter new eye elevation: ");
```

```
            Input(NULL,&zeye,NULL,NULL);
            printf("\r\n");
            break;
         case '2':
            printf("\r\nEnter new step size: ");
            Input(NULL,&dx,NULL,NULL);
            printf("\r\n");
            break;
         case '3':
            printf("\r\nEnter maximum horizontal range: ");
            Input(NULL,&xmax,NULL,NULL);
            printf("\r\n");
            break;
         case '4':
            printf("\r\nEnter maximum ray elevation: ");
            Input(NULL,&zmax,NULL,NULL);
            printf("\r\n");
            break;
         case '5':
            RayAngleMenu();   // Goto a menu that allows changes to the ray angle
            break;
         case '6':
            // Goto a menu that allows changes to the data files (tp,atm,arf)
            if (((combo == 0) && (wavy == 1)) ||
                  ((combo == 0) && (wavy == 0) && (flat == 0))) atmMenu();
            else if (combo == 0) tpMenu();
            else arfMenu();
            break;
         default:
            printf("\r\n Start ray tracing.");
            sleep(1);
            quit = TRUE;
            break;
      }
   }  while (quit == FALSE);
}


//****************************** RayAngleMenu ****************************//
//*                                                                      *//
//* Purpose: This function allows the user to change the ray angles      *//
//*          used in the previous run of the program.                    *//
//*                                                                      *//
//************************************************************************//
void RayAngleMenu() {
   int i;                  // Counter Variable
   boolean quit = FALSE;   // Equal TRUE when user wants to quit
   char ch;                // Input character
   double dummyDouble;     // Dummy variable

   // Keep looping through menu
   do {
      // Clear screen and print menu
      clrscr();
      printf("\r\n");
      printf(" Select the ray angle you wish to change.\r\n");
      printf(" (Press <enter> the return to previous menu)\r\n\r\n");
      for (i=0; i<nrays; i++) {
         printf("  %c: %.3lf arcmin\r\n",(i+'a'),(angle[i]*3437.747));
      }
      printf("\r\n Ray (letter), Add (+), Delete (-): ");

      // Get a selection from the user
      do {
         ch=getch();
```

```
        } while (((ch<'a')||(ch>=(i+'a')))&&(ch!='+')&&(ch!='-')&&(ch!=13));

        // Print user selection
        printf("%c\n",ch);

        // Depending on the user selection change a ray angle
        if ((ch >= 'a') && (ch <= ('a' + nrays))) {
            printf("\r\nEnter new ray angle: ");
            Input(NULL,&dummyDouble,NULL,NULL);
            angle[(ch-'a')] = dummyDouble/3437.747;
        }
        // Add a new ray angle
        else if (ch == '+') {
            if (nrays == MAXRAYS) {
                printf("\r\n A maximum of %d rays are allowed.\r\n",MAXRAYS);
                sleep(1);
            }
            else {
                printf("\r\n Add... Enter new ray angle: ");
                Input(NULL,&dummyDouble,NULL,NULL);
                angle[nrays] = dummyDouble/3437.747;
                nrays++;
            }
        }
        // Remove one of the current ray angles
        else if (ch == '-') {
            if (nrays == 1) {
                printf("\r\n You only have one ray, you cannot delete it.\r\n");
                sleep(1);
            }
            else {
                printf("\r\n Delete... Enter letter of ray you wish to delete: ");
                do {
                    ch = getch();
                } while ((ch<'a')||(ch>=(nrays+'a')));
                int delPosition = (ch-'a');
                for (i=0; i<nrays; i++) {
                    if (i>delPosition) angle[i-1]=angle[i];
                }
                nrays--;
            }
        }
        // Return to main menu
        else {
            printf("\r\n Return to previous menu.");
            sleep(1);
            quit = TRUE;
        }
    }  while (quit == FALSE);
}


//********************************* tpMenu ******************************//
//*                                                                       *//
//* Purpose: This function allows the user to change the temperature    *//
//*          levels in a temperature profile file (.tp) temporarily.    *//
//*                                                                       *//
//*************************************************************************//
void tpMenu(){
    int i;                  // Counter Variable
    boolean quit = FALSE;   // Equal TRUE when user wants to quit
    char ch;                // Input character
    double dummyDouble;     // Dummy variable

    // Keep looping through menu
```

```
do {
    // Clear screen and print menu
    clrscr();
    printf("\r\n");
    printf(" Select the temperture level you wish to change.\r\n");
    printf(" (Press <enter> the return to previous menu)\r\n\r\n");
    for (i=0; i<numLevels; i++) {
        printf("  %c: %5.31f Metres %5.41f Degrees\r\n",(i+'a'),
                  tpElev[i],tpTemp[i]);
    }
    printf("\r\n Level (letter), Add (+), Delete (-): ");

    // Get a selection from the user
    do {
        ch=getch();
    } while (((ch<'a')||(ch>=(i+'a')))&&(ch!='+')&&(ch!='-')&&(ch!=13));

    // Print user selection
    printf("%c\n",ch);

    // Change the temperature of a selected level
    if ((ch >= 'a') && ((ch <= 'a' + numLevels))) {
        printf("\r\nEnter new elevation: ");
        Input(NULL,&dummyDouble,NULL,NULL);
        tpElev[(ch-'a')] = dummyDouble;
        printf("\r\nEnter new temperture: ");
        Input(NULL,&dummyDouble,NULL,NULL);
        tpTemp[(ch-'a')] = dummyDouble;
    }
    // Add a new temperature level
    else if (ch == '+') {
        if (numLevels == MAXTP) {
            printf("\r\n A maximum of %d levels are allowed.\r\n",MAXTP);
            sleep(1);
        }
        else {
            printf("\r\n Add... Enter new elevation: ");
            Input(NULL,&dummyDouble,NULL,NULL);
            int insertPos=numLevels;
            for (i=numLevels-1; i>=0; i--) {
                if (tpElev[i] > dummyDouble) {
                    insertPos = i;
                    tpElev[i+1]=tpElev[i];
                    tpTemp[i+1]=tpTemp[i];
                }
            }
            tpElev[insertPos] = dummyDouble;
            numLevels++;
            printf("\r\n Add... Enter new temperature at %lf m: ",dummyDouble);
            Input(NULL,&dummyDouble,NULL,NULL);
            tpTemp[insertPos] = dummyDouble;
        }
    }
    // Remove an old temperature level
    else if (ch == '-') {
        if (numLevels == 1) {
            printf("\r\n You only have one level, you cannot delete it.\r\n");
            sleep(1);
        }
        else {
            printf("\r\n Delete... Enter letter of level you wish to delete: ");
            do {
                ch = getch();
            } while ((ch<'a')||(ch>=(numLevels+'a')));
            int delPosition = (ch-'a');
```

```
            for (i=0; i<numLevels; i++) {
               if (i>delPosition) {
                  tpElev[i-1] = tpElev[i];
                  tpTemp[i-1] = tpTemp[i];
               }
            }
            numLevels--;
         }
      }
      // Return to the previous menu
      else {
         printf("\r\n Return to previous menu.");
         sleep(1);
         quit = TRUE;
      }
   }  while (quit == FALSE);
}


//********************************* atmMenu ******************************//
//*                                                                        *//
//* Purpose: This function allows the user to change the entries in a      *//
//*          atmosphere file (.atm) temporarily.                           *//
//*                                                                        *//
//**************************************************************************//
void atmMenu() {
   boolean quit = FALSE;   // Equal TRUE when user wants to quit
   char ch;                // Input character

   // Keep looping through menu
   do {
      // Clear screen and print menu
      clrscr();
      printf("\r\n");
      printf(" Please select one of the following variable to adjust.\r\n");
      printf(" (Press <enter> the return to ray tracing)\r\n\r\n");
      printf("  1: Magnitude Multiplication Factor  Ah = %.2lf\r\n",Ah);
      printf("  2: Angular Frequency                w = %.2lf rad/m\r\n",w);
      printf("  3: Horizontal Wave Number           k = %.2lf m\r\n",k);
      if (wavy == 0) {
         printf("  4: Centre of Log Function             A = %.2lf m\r\n",A);
         printf("  5: Width of Log Function              B = %.2lf m\r\n",B);
      }
      printf("\r\n Option: ");

      // Get a selection from the user
      do {
         ch=getch();
      } while (((ch<'1')||(ch>'5'))&&(ch!=13));

      // Print user selection
      printf("%c\n",ch);

      // Set variables depending on menu selection
      switch (ch) {
         case '1':
            printf("\r\nEnter magnitude multiplication factor: ");
            Input(NULL,&Ah,NULL,NULL);
            printf("\r\n");
            break;
         case '2':
            printf("\r\nEnter angular frequency: ");
            Input(NULL,&w,NULL,NULL);
            printf("\r\n");
            break;
```

```
            case '3':
                printf("\r\nEnter horizontal wave number: ");
                Input(NULL,&k,NULL,NULL);
                printf("\r\n");
                break;
            case '4':
                if (wavy == 0) {
                    printf("\r\nEnter centre of log function: ");
                    Input(NULL,&A,NULL,NULL);
                    printf("\r\n");
                }
                break;
            case '5':
                if (wavy == 0) {
                    printf("\r\nEnter width of log function: ");
                    Input(NULL,&B,NULL,NULL);
                    printf("\r\n");
                }
                break;
            default:
                printf("\r\n Return to previous menu.");
                sleep(1);
                quit = TRUE;
                break;
        }
    }  while (quit == FALSE);
}


//******************************** arfMenu ********************************//
//*                                                                        *//
//* Purpose: This function allows the user to change the entries in a      *//
//*          atmosphere region file (.arf) temporarily.                    *//
//*                                                                        *//
//************************************************************************//
void arfMenu() {
    int i;                  // Counter Variable
    boolean quit = FALSE;   // Equal TRUE when user wants to quit
    char ch;                // Input character
    double dummyDouble;     // Dummy variable
    char dummyString[80];   // Dummy variable
    char regionType;        // Variable to set region type

    // Keep looping through menu
    do {
        // Clear screen and print menu
        clrscr();
        printf("\r\n");
        printf(" Select the region you wish to change.\r\n");
        printf(" (Press <enter> the return to previous menu)\r\n\r\n");
        for (i=0; i<numregion; i++) {
            if (atmregion[i].flat == 1) regionType = 'F';
            else if (atmregion[i].wavy == 1) regionType = 'W';
            else regionType = 'L';
            printf("  %c: File: %s (%c)   Steps: %d (%.2lf m)\r\n",(i+'a'),
                      atmregion[i].filename,regionType,atmregion[i].numsteps,
                      (atmregion[i].numsteps*dx));
        }
        printf("\r\n Region (letter), Add (+), Delete (-): ");

        // Get a selection from the user
        do {
            ch=getch();
        } while (((ch<'a')||(ch>=(i+'a')))&&(ch!='+')&&(ch!='-')&&(ch!=13));
```

```
// Print user selection
printf("%c\n",ch);

// Change an existing region
if ((ch >= 'a') && (ch <= (numregion+'a'))) {
    int position = ch - 'a';
    printf("\r\n Enter new region file Name: ");
    Input(NULL,NULL,dummyString,NULL);
    strcpy(atmregion[position].filename,dummyString);
    printf("\r\n Enter new region type (f,w,l): ");
    do {
        ch = getch();
    } while ((ch != 'f') && (ch != 'w') && (ch != 'l'));
    printf("%c",ch);
    atmregion[position].flat = 0;
    atmregion[position].wavy = 0;
    if (ch == 'f') atmregion[position].flat = 1;
    if (ch == 'w') atmregion[position].wavy = 1;
    printf("\r\n Enter number of steps for this region: ");
    Input(NULL,&dummyDouble,NULL,NULL);
    atmregion[position].numsteps = dummyDouble;
}
// Add a new region file
else if (ch == '+') {
    if (numregion == MAXRAYS) {
        printf("\r\n A maximum of %d rays are allowed.\r\n",MAXRAYS);
        sleep(1);
    }
    else {
        printf("\r\n Enter letter to place region in: ");
        do {
            ch = getch();
        } while ((ch<'a')||(ch>(numregion+'a')));
        printf("%c",ch);
        int insertPosition = (ch-'a');
        for (i=numregion; i>=0; i--) {
            if (i > dummyDouble) atmregion[i]=atmregion[i-1];
        }
        numregion++;

        printf("\r\n Enter new region file Name: ");
        Input(NULL,NULL,dummyString,NULL);
        strcpy(atmregion[insertPosition].filename,dummyString);

        printf("\r\n Enter new region type (f,w,l): ");
        do {
            ch = getch();
        } while ((ch != 'f') && (ch != 'w') && (ch != 'l'));
        printf("%c",ch);
        atmregion[insertPosition].flat = 0;
        atmregion[insertPosition].wavy = 0;
        if (ch == 'f') atmregion[insertPosition].flat = 1;
        if (ch == 'w') atmregion[insertPosition].wavy = 1;

        printf("\r\n Enter number of steps for this region: ");
        Input(NULL,&dummyDouble,NULL,NULL);
        atmregion[insertPosition].numsteps = dummyDouble;
    }
}
// Remove a existing region file
else if (ch == '-') {
    if (numregion == 1) {
        printf("\r\n You only have one region, you cannot delete it.\r\n");
        sleep(1);
    }
```

```
            else {
                printf("\r\n Delete... Enter letter of region you wish to delete: ");
                do {
                    ch = getch();
                } while ((ch<'a')||(ch>=(numregion+'a')));
                int delPosition = (ch-'a');
                for (i=0; i<numregion; i++) {
                    if (i>delPosition) atmregion[i-1]=atmregion[i];
                }
                numregion--;
            }
        }
        // Return the previous menu
        else {
            printf("\r\n Return to previous menu.");

            sleep(1);
            quit = TRUE;
        }
    }   while (quit == FALSE);
}


//*************************************************************************//
//*                                                                       *//
//*                            Other Functions                            *//
//*                                                                       *//
//*************************************************************************//

//****************************** SetRays *********************************//
//*                                                                       *//
//* Purpose: Add a slope to a flat atmoshpere by increasing the eye       *//
//*          level at the observer.                                       *//
//*                                                                       *//
//*************************************************************************//
void SetRays(double slope, int startPos, int steps) {
    double distance = steps*dx;
    double raiseEye = distance*tan(slope/3437.747);
    for (int i=0; i<nrays; i++) {
        ray[i][startPos] = ray[i][startPos] + raiseEye;
        start[i].z = start[i].z + raiseEye;
        start[i].angle = start[i].angle - slope/3437.747;
    }
}


//****************************** ResetRays *******************************//
//*                                                                       *//
//* Purpose: Removes the increase eye level artifact from a set of rays *//
//*          in a sloped atmosphere.                                      *//
//*                                                                       *//
//*************************************************************************//
void ResetRays(double slope, int startPos, int steps) {
    double distance = steps*dx;
    for (int j=startPos; j<=startPos+steps; j++) {
        double lowerEye = (distance-(j-startPos)*dx)*tan(slope/3437.747);
        for (int i=0; i<nrays; i++) {
            ray[i][j] -=lowerEye;
        }
    }
    for (int i=0; i<nrays; i++) {
        start[i].angle = start[i].angle + slope/3437.747;
    }
}
```

```
//****************************** CalcPressDen ***************************//
//*                                                                      *//
//* Purpose: Calculates the pressure and density at points on the        *//
//*          temperature profile.                                        *//
//* Parameters: None.                                                    *//
//* Return: None.                                                        *//
//*                                                                      *//
//************************************************************************//
void CalcPressDen(void)
{
   double press[MAXTP];    // Variable holding pressures
   double Tint = 0.0;
   int i;                  // Counter variable

   press[0]=101325.0;                    // surface pressure
   den[0]=press[0]*BETA/(tpTemp[0]+K);  // surface density

   for (i=1;i<numLevels;i++) {
      Tint=Tint+fabs(0.5*(1/(tpTemp[i]+K)-1/(tpTemp[i-1]+K))*
             (tpElev[i]-tpElev[i-1]));
      press[i]=press[0]*Exp(-1*GRAV*BETA*Tint);
      den[i]=BETA*press[i]/(tpTemp[i]+K);
   }
}


//****************************** RayTrace ******************************//
//*                                                                      *//
//* Purpose: Calculates the paths of rays over a number of intervals.    *//
//* Parameters: numsteps - Number of intervals over which to trace rays.*//
//* Return: None.                                                        *//
//*                                                                      *//
//************************************************************************//
void RayTrace(int numsteps)
{
   int xx,yy;           // starting location of cursor
   int i,j,grounded,spaced,startlayer,endlayer,endstep,direction,vertexlayer;
   double phi,curv=0.0,xo,zo,z=0.0,xstep=0.0,xvertex,zvertex,a,b;
   xz intpt;            // intersection point

   /*** loop thru rays ***/
   printf(" Tracing ray ");
   xx=wherex();
   yy=wherey();
   for (i=0;i<nrays;i++) {
      gotoxy(xx,yy);
      clreol();
      //textcolor(C2);
      /*c*/printf("%d",i);

      grounded=0;    // flag if ray hits ground
      spaced=0;      // flag if ray exceeds zmax

      for (j=1;j<=numsteps;j++) {
         if (j==1) { xo=0; zo=start[i].z; phi=start[i].angle; }
         else { xo=dx*(j-1); zo=z; phi=atan(-1*curv*xstep+tan(phi)); }
         xstep=dx;
         startlayer=Locate(xo,zo);
         endstep=0;
         if (start[i].grounded==1) grounded=1;
         if (start[i].spaced==1) spaced=1;

         /*** do one step ***/
         do {
            if ((grounded==0)&&(spaced==0)) {
```

```
                    curv=Curvature(xo,zo,(PI/2-phi));
                    z=-0.5*curv*xstep*xstep+xstep*tan(phi)+zo;
                    ray[i][j+prevsteps]=z;

                    // Check if ray hit ground
                    if (z<0) {
                        ray[i][j+prevsteps]=-1.0;
                        grounded=1;
                        start[i].grounded=1;
                        endstep=1;
                    }

                    // Check if ray went above maximum altitude
                    if (z>zmax) {
                        ray[i][j+prevsteps]=zmax+1.0;
                        spaced=1;
                        start[i].spaced=1;
                        endstep=1;
                    }

                    /** check to see if in same layer **/
                    endlayer=Locate(xo+xstep,z);
                    if (endlayer==startlayer) {
                        a=-0.5*curv;
                        b=tan(phi);
                        if (a!=0.0) {
                            xvertex=-0.5*b/a;
                            zvertex=a*xvertex*xvertex+b*xvertex+zo;
                            if ((xvertex>0)&&(xvertex<xstep)) {
                                vertexlayer=Locate(xo+xvertex,zvertex);
                                if (vertexlayer<startlayer) direction=-1;
                                if (vertexlayer>startlayer) direction=0;
                                if (vertexlayer==startlayer) endstep=1;
                            }
                            else
                                endstep=1;
                        }
                        else
                            endstep=1;
                    }
                    if (endlayer<startlayer) direction=-1;
                    if (endlayer>startlayer) direction=0;

                    /** if different layer, find intersection **/
                    if (endstep==0) {
                        intpt=Intersection(xo,xstep,-0.5*curv,tan(phi),zo,
startlayer+direction,direction+1);
                        xstep=xstep-intpt.x;
                        xo+=intpt.x;
                        zo=intpt.z;
                        startlayer=Locate(xo,zo);
                        phi=atan(-1*curv*intpt.x+tan(phi));
                    }
                }
                else endstep=1;
            }  while (endstep==0);
            if (grounded==1) ray[i][j+prevsteps]=-1.0;
            if (spaced==1) ray[i][j+prevsteps]=zmax+1.0;
        }
        /* save start[] data for next atmosphere region */
        start[i].z=ray[i][j-1+prevsteps];
        start[i].angle=atan(-1*curv*xstep+tan(phi));
    }
    gotoxy(xx-13,yy);
```

```
    clreol();
}


//***************************** Intersection ******************************//
//*                                                                       *//
//| Purpose: Calculates the point of intersection between the ray and    *//
//*          the atmosphere boundary.                                     *//
//* Parameters: xo - x-coordinate of starting point of ray arc.          *//
//*             x - length of interval in which intersection occurs.     *//
//*             a -┐ parameters of ray arc equation:                     *//
//*             b -┼                  z=a·x²+b·x+c                      *//
//*             c -┘                                                     *//
//*             layer - layer number with which intersection occurs.     *//
//*             up - =1 if ray moving up; =0 if downwards.               *//
//* Return: The location of the interception point (x,z).                *//
//*                                                                       *//
//***************************************************************************//
xz Intersection(double xo,double x,double a,double b,double c,int layer,int up)
{
 double xstep,guess,slope,error,intercept,oldguess,zguess;
 double temp;
 int iter;
 xz ret;

 xstep=x;
 guess=0;

 /*** perform Newton method ***/
 iter=0;
 do
   {
    do
      {
       oldguess=guess;
       zguess=a*guess*guess+b*guess+c-Boundary(layer,(guess+xo));
       if ((flat==1)&&(wavy==0))
          slope=2*a*guess+b;
       if ((flat==0)&&(wavy==1))
          slope=2*a*guess+b-Ah*mag[layer]*k*cos(k*(guess+xo)-w*waveTime);
       if ((flat==0)&&(wavy==0))
          {
           temp=k*cos(k*(guess+xo)-w*waveTime)*(1+Exp(-1.0*B*(guess+xo-A))) +
                    B*sin(k*(guess+xo)-w*waveTime)*Exp(-1.0*B*(guess+xo-A));
           slope=2*a*guess+b-temp*Ah*mag[layer]/pow( (1+Exp(-1.0*B*(guess+xo-A)))
,2);
          }
       if (slope==0.0)
          guess+=1.0;
      }
    while(slope==0.0);
    intercept=zguess-slope*guess;
    guess=-1.0*intercept/slope;

    if ((guess<0)||(guess>xstep))
      {
       iter++;
       guess=0.1*xstep*iter;
       oldguess=0;
      }

    error=fabs(guess-oldguess);
   }
while (error>ERR);
```

```
  ret.x=guess;
  if (up==1)
    ret.z=Boundary(layer,(guess+xo))+ ERR;
  else
    ret.z=Boundary(layer,(guess+xo))- ERR;

  return(ret);
 }


 //****************************** Curvature ******************************//
 //*                                                                       *//
 //* Purpose: Calculates the curvature for a given (x,z) point.            *//
 //* Parameters: x - the x-coordinate in meters.                           *//
 //*             z - the z-coordinate in meters.                           *//
 //*             angle - angle ray makes with vertical in radians.         *//
 //* Return: The curvature at this point on the ray.                       *//
 //*                                                                       *//
 //*************************************************************************//
 double Curvature(double x,double z,double angle)
 {
  double ret,density,dT,dz,temp;
  int layer;

  layer=Locate(x,z);
  ret=0.0;
  if ((layer>0)&&(layer<numLevels))
     {
      dT=tpTemp[layer]-tpTemp[layer-1];
      dz=Boundary(layer,x)-Boundary((layer-1),x);
      density=den[layer]+(z-Boundary((layer-1),x))*(den[layer]-den[layer-1])/dz;
      temp=tpTemp[layer]+(z-Boundary(layer-1,x))*dT/dz+K;

     ret=(EPSILON*density)/(temp*(1+EPSILON*density))*sin(angle)*(dT/dz+BETA*GRAV);

      ret=ret-KEARTH;  // return effective curvature
     }
  return(ret);
 }


 //****************************** Locate ******************************//
 //*                                                                   *//
 //* Purpose: Given the (x,z) values for a point, it finds which layer *//
 //*          this point is located in. The lowest layer is layer 1.   *//
 //* Parameters: x - x-coordinate in meters.                           *//
 //*             z - z-coordinate in meters.                           *//
 //* Return: Layer in which (x,z) point is located.                    *//
 //*                                                                   *//
 //*********************************************************************//
 int Locate(double x, double z)
 {
  int i,ret=0;

  for (i=0;i<numLevels;i++)
     if (z>Boundary(i,x)) ret++;

  return(ret);
 }


 //****************************** Boundary ******************************//
 //*                                                                     *//
 //* Purpose: Returns the z-coordinate of a layer boundary for a given   *//
 //*          x coordinate and layer number.                             *//
```

```
//* Parameters: layer - layer number of boundary.                          *//
//*             x - value of x coordinate (meters).                        *//
//* Return: None.                                                          *//
//*                                                                        *//
//**************************************************************************//
double Boundary(int layer, double x)
{
 double ret;

 /* flat atmosphere */
 if ((flat==1)&&(wavy==0))
   ret=tpElev[layer];
 /* wavy atmosphere */
 if ((flat==0)&&(wavy==1))
   ret=tpElev[layer]+Ah*mag[layer]*sin(k*x-w*waveTime);
 /* logistic atmosphere */
 if ((flat==0)&&(wavy==0))
   ret=tpElev[layer]+(1/(1+Exp(-1.0*B*(x-A))))*Ah*mag[layer]*sin(k*x-w*waveTime);

 return(ret);
}


//********************************** Exp ***********************************//
//*                                                                        *//
//* Purpose: Calculates value of Exp(x) but prevents overflow error.       *//
//* Parameters: x - arguement of Exp().                                    *//
//* Return: None.                                                          *//
//*                                                                        *//
//**************************************************************************//
double Exp(double x)
{
 double ret;

 if (x>100.0) x=100.0;
 if (x<-100.0) x=-100.0;
 ret=exp(x);

 return(ret);
}


/******************************** GetLine *********************************/
/*                                                                        */
/* Purpose: Used to read in line-by-line an .ARF, .ATM, or .LOG file.     */
/*          It ignores lines that start with '/' (comments) and sends     */
/*          all others back to caller.                                    */
/* Parameters: *IN - pointer to input file.                               */
/*               *fileline - pointer to string returned to caller.        */
/*                                                                        */
/**************************************************************************/
void GetLine(FILE *IN,char *fileline)
{
   char temp[80];

   do {
      fgets(temp,80,IN);
   } while (temp[0]=='/');

   strcpy(fileline,temp);
}


/****************************** MakeRayType *******************************/
/*                                                                        */
```

```
/* The function makes a ray type for the graphing procedure                */
/*                                                                         */
/***************************************************************************/
rayType *makeRayType(char *fileName) {
    int i,j;      // Counter Variables

    // Create and allocate variable to be returned
    rayType   *raySet;
    raySet = (rayType *) malloc(sizeof(rayType));

    // Set basic variables
    strcpy(raySet->name,fileName);
    raySet->time = waveTime;
    raySet->slope = slope;
    raySet->eyeLevel = zeye;
    raySet->xstep = dx;
    raySet->xmax = xmax;
    raySet->ymax = zmax;
    raySet->numberRays = nrays;
    raySet->graphxmin = 0;
    raySet->graphxmax = xmax/1000.0;
    raySet->graphymin = 0;
    raySet->graphymax = zmax;

    // Allocate room for the ray angles and copy angles into array
    raySet->angles = (double *) malloc(sizeof(double)*nrays);
    for(i=0;i<nrays;i++) raySet->angles[i] = angle[i]*3437.747;

    // Allocate room for ray arrays
    raySet->ray = (double **) malloc(sizeof(double*)*nrays);
    for(i=0;i<nrays;i++)      raySet->ray[i]      =      (double
*)malloc(sizeof(double)*numpoints);

    // Copy data into these arrays
    for(i=0;i<nrays;i++) {
       for(j=0;j<numpoints;j++) raySet->ray[i][j] = ray[i][j];
    }

    // Return variable that is created by this function
    return raySet;
}


//********************************** Exp ********************************//
//*                                                                      */
//* Purpose:  Reads in input from keyboard only allowing proper          */
//*           Characters for desired input variable type.                */
//* Parameters: *intptr - pointer to integer variable.                   */
//*              *fltptr - pointer to double variable.                   */
//*              *charptr - pointer to character string variable.        */
//*             *ext - if want filename input, this variable contains    */
//*                   the extension that will be added to filename if    */
//*                   no extension is given.                             */
//* Return: 0 - if no problems.                                          */
//*         1 - if escape key is hit during input.                       */
//*                                                                      */
//************************************************************************//
int Input(int *intptr,double *fltptr,char *charptr,char *ext)
{
 char ch,text[80],name[10],type;
 int i=0,ret=0,valid,last,tempint;
 double tempflt;

 if (intptr) type='I';
 if (fltptr) type='F';
```

```
  if (charptr) type='T';
  if (ext) type='N';
  strcpy(text,"            ");
  do
     {
       ch=getch();
       valid=0;
       if (ch==27) {ch=13; ret=1;};

       /**** if looking for integer input ****/
       if ((type=='I') && (ch!=13) && ( ((ch>='0')&&(ch<='9'))
             ||(ch=='+')||(ch=='-')||(ch==8) ) )
        valid=1;

       /**** if looking for double input ****/
       if ((type=='F') && (ch!=13) && ( ((ch>='0')&&(ch<='9'))||(ch=='+')
             ||(ch=='-')||(ch=='.')||(ch=='E')||(ch=='e')||(ch==8) ) )
        valid=1;

       /**** if looking for text input ****/
       if ( (type=='T') && (ch!=13) && ( ((ch>=32)&&(ch<='~'))||(ch==8) ) )
        valid=1;

       /**** if looking for filename input ****/
       if ((type=='N') && (ch!=13) && ( ((ch>='0')&&(ch<=':'))||((ch>='@')
             &&(ch<='~'))||(ch=='-')||(ch=='.')||((ch>='#')&&(ch<='&'))||(ch==8)
) )
        valid=1;

       /**** put the character in the string or move backwards if backspace ****/
       if (valid==1)
        if ((ch==8)&&(i>0))
           {
            /*c*/printf("%c",ch);
            /*c*/printf(" %c",ch);
            i=i-2;
            text[i+1]=' ';
           }
        else
           {
            text[i]=ch;
            /*c*/printf("%c",ch);
           }
       i++;
     }
  while (ch!=13);
  text[i-1]=NULL;

  /**** if filename, add extension ****/
  if (type=='N')
    {
     last=0;
     for (i=0;i<strlen(text);i++)
        if (text[i]==92) last=i;
     if (last==0)
       {
        if ((strchr(text,'.')==NULL))
           {
            strcat(text,".");
            strcat(text,ext);
           }
       }
       else
       {
        for (i=last+1;i<strlen(text)+1;i++)
```

```
            name[i-last-1]=text[i];
          if ((strchr(name,'.')==NULL))
            {
             strcat(text,".");
             strcat(text,ext);
            }
        }
    }

    /**** fill the appropriate pointer ****/
 if (type=='I')
    {
     //tempint=strtol(text,NULL,10);
     tempint=atoi(text);
     *intptr=tempint;
    }
 if (type=='F')
    {
     tempflt=strtod(text,NULL);
     *fltptr=tempflt;
    }
 if ((type=='N')||(type=='T'))
    strcpy(charptr,text);

 return(ret);
}
```

```
// MM    MM  GGGGGGG RRRRRRR    AAAA   PPPPPPP  HH    HH       CCCCCC
// MMM  MMM GGGGGGGG RRRRRRRR  AA  AA  PPPPPPPP HH    HH      CCCCCCCC
// MMMMMMMM GG       RR    RR AA    AA PP    PP HH    HH      CC
// MMMMMMMM GG  GGGG RRRRRRRR AAAAAAAA PPPPPPPP HHHHHHHH      CC
// MM MM MM GG  GGGG RRRRRRR  AAAAAAAA PPPPPPP  HHHHHHHH      CC
// MM    MM GG    GG RR  RR   AA    AA PP       HH    HH      CC
// MM    MM GGGGGGGG RR   RR  AA    AA PP       HH    HH  ..  CCCCCCCC
// MM    MM  GGGGGGG RR    RR AA    AA PP       HH    HH  ..   CCCCCC

// Graphing program based on a
// Modification of EDITTC Program...  December 29, 1994


/*****************************************************************************/
/***************************** INCLUDE FILES *********************************/
/*****************************************************************************/
#pragma hdrfile "mgraph.sym"
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include <dir.h>
#include <math.h>
#include <ctype.h>

#include "mgraph.h"


/*****************************************************************************/
/****************************** DEFINES **************************************/
/*****************************************************************************/
#define XAxisMin 70      /* Minimum X pixel value on X-Axis */
#define XAxisMax 570     /* Maximum X pixel value on X-Axis */
#define YAxisMin 390     /* Maximum Y pixel value on Y-Axis */
#define YAxisMax 40      /* Minimum Y pixel value on Y-Axis */
#define XSpacing 50      /* Number of pixels between ticks on X-Axis */
#define YSpacing 35      /* Number of pixels between ticks on Y-Axis */

#define boolean int      // Set up the boolean variable type
#define TRUE 1           // Set true to 1
#define FALSE 0          // Set false to 0


/*****************************************************************************/
/************************** TYPE DEFINITIONS *********************************/
/*****************************************************************************/
typedef struct {
   int x;
   int y;
   int quit;
} xy;

typedef struct {
   double x[1000];
   double y[1000];
   int number;
   boolean inGraph[1000];
} lineType;

typedef struct {
   double *apparent;
   double *actual;
   double graphxmax;
```

```
    double graphxmin;
    double graphymax;
    double graphymin;
    int number;
} tcType;

/*************************************************************************/
/***************************** INCLUDE FILES *****************************/
/*************************************************************************/
void DrawAxis(void);                            // Draw the axis of the graph
void DrawText(char *graphTitle,
              char *yLabel,
              char *xLabel);                    // Draw the text info on page
void DrawGrid(int grid);                        // Draw a grid on the graph
void DrawScale(double xmin, double xmax,
              double ymin, double ymax);        // Draw scale value on axis
void DrawPoints(lineType *points,
                double xmin, double xmax,
                double ymin, double ymax,
                int colour);                    // Draw the points on a line
void DrawLine(lineType *points,
              double xmin, double xmax,
              double ymin, double ymax,
              int colour);                      // Draw a line from a set of points

tcType *DrawTransferChar(rayType *rays,
                         int grid);             // Draw a transfer characteristic
void RedrawTransferChar(tcType *tc,
                        int grid,
                        int newTC);             // Draw a tc already created
void DrawCompareTC(tcType *tc,
                   char *compareFile,
                   int grid);                   // Draw a tc on top of one displayed
void DrawRays(rayType *rays,
              int grid);                        // Draw the set of rays

void SetGraphics(void);                         // Setup the graphics routines
int Round(double number);                       // Round a double into an integer
void message(char *text);                       // Output a message
char *prompt(char *text);                       // Output prompt and get response

void SaveRay(char *fileName,
             rayType *rays,
             int addProfile);                   // Saves rays to a file
void SaveTC(char *fileName, tcType *tc);        // Saves tc to a file


/*************************************************************************/
/***************************** MAIN FUNCTION *****************************/
/*************************************************************************/
void GraphData(rayType *rays)
{
   int graphingData = 0;      // flag to output ray data in graph format
   int grid=1;                // flag if grid present (1=present 0=not)
   int isRays=1;              // flag if rays are drawn (1=rays 0=tc)
   char *inputStr;            // String of information input from user
   int count,index;           // Counter Variables
   tcType *tc=NULL;           // Current TC

   // Set up Graphics Screen and Draw X and Y axis
   SetGraphics();
   DrawAxis();
   DrawText("Rays","Metres","Kilometres");
   DrawScale(rays->graphxmin,rays->graphxmax,
             rays->graphymin,rays->graphymax);
```

```
DrawRays(rays,grid);

// Set up default write modes
setcolor(WHITE);
setfillstyle(SOLID_FILL,BLACK);

// Get Characters From User
while (1) {
    char character = getch();
        switch (character) {
    /*XX*/case 'x':   // This case scales the X-Axis
           int valid;
           double newmin;
           double newmax;
           valid = 0;
           while (valid == 0) {
               char firstStr[20];
               char secondStr[20];
               inputStr = prompt("Enter new X-Axis values (min max): ");
               // Find first number string in input
               count=0;
               index=0;
               while ((count<strlen(inputStr)) &&
                           ((inputStr[count] == '.') ||
                             (inputStr[count] == '-') ||
                             isdigit(inputStr[count]))) {
                   firstStr[index] = inputStr[count];
                   index++;
                   count++;
               }
               firstStr[index]=0;
               // Find start on next number
               while  ((count<strlen(inputStr)) &&
                           (inputStr[count] != '.') &&
                           (inputStr[count] != '-') &&
                           !isdigit(inputStr[count])) {
                   count++;
               }
               // Find second number string in input
               index=0;
               while ((count<strlen(inputStr)) &&
                           ((inputStr[count] == '.') ||
                             (inputStr[count] == '-') ||
                              isdigit(inputStr[count]))) {
                   secondStr[index] = inputStr[count];
                   index++;
                   count++;
               }
               secondStr[index]=0;
               // Check if the strings are valid numeric inputs
               if ((firstStr[0] == 0) || (secondStr[0] == 0)) {
                   valid = 0;
               }
               else {
                   sscanf(firstStr,"%lf",&newmin);
                   sscanf(secondStr,"%lf",&newmax);
                   if (newmax > newmin) valid = 1;
                   else valid = 0;
               }
               // If the strings were not valid output message
               if (valid == 0) {
                   message("Invalid input values... Try again.");
                   sleep(2);
               }
           }
```

```
            // Redraw TC or Rays
            if (isRays == 1) {
                rays->graphxmin = newmin;
                rays->graphxmax = newmax;
                DrawAxis();
                DrawRays(rays,grid);
                DrawScale(rays->graphxmin,rays->graphxmax,
                                rays->graphymin,rays->graphymax);
            }
            else {
                tc->graphxmin = newmin;
                tc->graphxmax = newmax;
                DrawAxis();
                RedrawTransferChar(tc,grid,1);

DrawScale(tc->graphxmin,tc->graphxmax,tc->graphymin,tc->graphymax);
            }
            break;
      /*YY*/case 'y':   // This case scales the Y-Axis
            valid = 0;
            while (valid == 0) {
                char firstStr[20];
                char secondStr[20];
                inputStr = prompt("Enter new Y-Axis values (min max): ");
                // Find first number string in input
                count=0;
                index=0;
                while ((count<strlen(inputStr)) &&
                            ((inputStr[count] == '.') ||
                              (inputStr[count] == '-') ||
                              isdigit(inputStr[count]))) {
                    firstStr[index] = inputStr[count];
                    index++;
                    count++;
                }
                firstStr[index]=0;
                // Find start on next number
                while  ((count<strlen(inputStr)) &&
                            (inputStr[count] != '.') &&
                            (inputStr[count] != '-') &&
                            !isdigit(inputStr[count])) {
                    count++;
                }
                // Find second number string in input
                index=0;
                while ((count<strlen(inputStr)) &&
                            ((inputStr[count] == '.') ||
                              (inputStr[count] == '-') ||
                              isdigit(inputStr[count]))) {
                    secondStr[index] = inputStr[count];
                    index++;
                    count++;
                }
                secondStr[index]=0;
                // Check if the strings are valid numeric inputs
                if ((firstStr[0] == 0) || (secondStr[0] == 0)) {
                    valid = 0;
                }
                else {
                    sscanf(firstStr,"%lf",&newmin);
                    sscanf(secondStr,"%lf",&newmax);
                    if (newmax > newmin) valid = 1;
                    else valid = 0;
                }
                // If the strings were not valid output message
```

```
                if (valid == 0) {
                    message("Invalid input values... Try again.");
                    sleep(2);
                }
            }
            // Redraw TC or Rays
            if (isRays == 1) {
                rays->graphymin = newmin;
                rays->graphymax = newmax;
                DrawAxis();
                DrawRays(rays,grid);
                DrawScale(rays->graphxmin,rays->graphxmax,
                                rays->graphymin,rays->graphymax);
            }
            else {
                tc->graphymin = newmin;
                tc->graphymax = newmax;
                DrawAxis();
                RedrawTransferChar(tc,grid,1);
DrawScale(tc->graphxmin,tc->graphxmax,tc->graphymin,tc->graphymax);
            }
            break;
        /*GG*/case 'g':   // This case toggles the drawing grid
            if (grid==0) {
                grid=1;
                DrawGrid(grid);
            }
            else {
                grid=0;
                if (isRays == 1) DrawRays(rays,grid);
                else RedrawTransferChar(tc,grid,1);
            }
            break;
        /*FF*/case 'F':     // This sets a variable to output ray graphing data
            graphingData = 1;
        /*FF*/case 'f':     // This case outputs the current data to a file (ray or
tc)
            int returnValue;
            char *inputFile;
            inputFile = prompt("Enter name of file to output: ");
            if (isRays == 1) returnValue = ValidFileName(inputFile,".ray",'w');
            else returnValue = ValidFileName(inputFile,".tc",'w');
            if ((returnValue == 1) || (returnValue == 2)) {
                char *inputData = (char *) malloc(sizeof(char));
                inputData[0] = 0;
                while ((inputData[0] != 'y') && (inputData[0] != 'Y') &&
                        (inputData[0] != 'n') && (inputData[0] != 'N')) {
                    free(inputData);
                    inputData = prompt("File already exists.  Overwrite? (Y/N): ");
                }
                if ((inputData[0] == 'n') || (inputData[0] == 'N')) break;
            }
            if (returnValue != 0) {
                if (isRays == 1) SaveRay(inputFile,rays,graphingData);
                else SaveTC(inputFile,tc);
            }
            else {
                message("Invalid file name.");
                sleep(2);
                message("");
            }
            free(inputFile);
            graphingData = 0;
            break;
```

```
        /*NN*/case 'n':     // This is the case to create a new tc and display it
               message(""); // Clear any messages
               isRays = 0;
               DrawAxis();
               tc = DrawTransferChar(rays,grid);
               DrawText("Transfer Characteristic","Metres","Metres");
               DrawScale(tc->graphxmin,tc->graphxmax,tc->graphymin,tc->graphymax);
               break;
        /*TT*/case 't':     // This case changes the display from rays to a created
tc
               message(""); // Clear any messages
               if (tc != NULL) {
                   if (isRays == 1) {
                       isRays = 0;
                       DrawAxis();
                       RedrawTransferChar(tc,grid,1);
                       DrawText("Transfer Characteristic","Metres","Metres");
                       DrawScale(tc->graphxmin,tc->graphxmax,tc->graphymin,
                       tc->graphymax);
                   }
                   else {
                       message("Transfer Characteristic is already displayed.");
                   }
               }
               else {
                   message("Transfer Characteristic does not exist.");
               }
               break;
        /*CC*/case 'c':     // Secret Function to compare transfer characteristics
               if ((tc == NULL)||(isRays == 1)) {
                   message("No trasfer characteristic exists for comparison.");
                   sleep(2);
                   message("");
                   break;
               }
               else {
                   char *compareFile;
                   compareFile = prompt("Enter file name to compare: ");
                   if (ValidFileName(compareFile,".tc",'r') != 0) {
                       DrawCompareTC(tc,compareFile,grid);
                       DrawText("Transfer Characteristic","Metres","Metres");
                       DrawScale(tc->graphxmin,tc->graphxmax,tc->graphymin,
                                 tc->graphymax);
                   }
                   else {
                       message("File does not exist.");
                       sleep(2);
                       message("");
                       free(compareFile);
                   }
               }
               break;
        /*RR*/case 'r':     // This case changes the display from a tc to rays
               message(""); // Clear any messages
               if (isRays == 1) {
                  message("Rays are already displayed.");
               }
               else {
                  if ((rays != NULL) && (rays->ray[0] != NULL)) {
                      isRays = 1;
                      DrawRays(rays,grid);
                      DrawAxis();
                      DrawText("Rays","Metres","Kilometres");
                      DrawScale(rays->graphxmin,rays->graphxmax,
                                rays->graphymin,rays->graphymax);
```

```
                }
                else {
                    message("Error in ray data.  Exiting...");
                    closegraph();
                    return;
                }
            }
            break;
        /*QQ*/case 'q':     // Exits graphics mode and returns to the main program
            closegraph();
            return;
          default:    // Default case is to do nothing
            break;
        }
    }
}


//#############################################################################//
//                                                                             //
//                                  Functions                                  //
//                                                                             //
//#############################################################################//

//****************************** DrawAxis *********************************//
//                                                                           //
// Creates a Y-Axis from 70,20 to 70,370 with tick marks every 35 pixels     //
//             line from 70,10 to 70,373                                     //
// Creates a X-Axis from 70,370 to 570,370 with tick marks every 50 pixels   //
//             line from 67,370 to 580,370                                   //
//                                                                           //
//***************************************************************************//
void DrawAxis(void) {
   int i;  // Counter Variable

   // Clear old scale values
   setfillstyle(SOLID_FILL,BLACK);
   bar(XAxisMin-3,YAxisMin+3,XAxisMin,YAxisMax-10);  // Y-Axis
   bar(XAxisMin-3,YAxisMin+3,XAxisMax+10,YAxisMin);  // X-Axis

   // Set Graph axis Colour
   setcolor(LIGHTGREEN);

   /*********** y axis ***********/
   line(XAxisMin,YAxisMin+3,XAxisMin,YAxisMax-10);

   // Tick Marks
   for (i=0;i<11;i++)
      line(XAxisMin,YAxisMin-YSpacing*i,XAxisMin-3,YAxisMin-YSpacing*i);

   // Arrow Head
   line(XAxisMin,YAxisMax-10,XAxisMin-3,YAxisMax-7);
   line(XAxisMin,YAxisMax-10,XAxisMin+3,YAxisMax-7);

   /*********** x axis ***********/
   line(XAxisMin-3,YAxisMin,XAxisMax+10,YAxisMin);

   // Tick Marks
   for (i=0;i<11;i++)
      line(XAxisMin+XSpacing*i,YAxisMin,XAxisMin+XSpacing*i,YAxisMin+3);

   // Arrow Head
   line(XAxisMax+10,YAxisMin,XAxisMax+7,YAxisMin-3);
   line(XAxisMax+10,YAxisMin,XAxisMax+7,YAxisMin+3);
}
```

```
//****************************** DrawText *********************************//
//                                                                          //
// Place the text information on the screen                                 //
//                                                                          //
//**************************************************************************//
void DrawText(char *graphTitle, char *yLabel, char *xLabel) {
    // Clear old scale values
    setfillstyle(SOLID_FILL,BLACK);
    bar(0,0,639,23);            // Clear Old Title
    bar(0,0,20,YAxisMin);       // Clear Old Y-Axis Label
    bar(0,410,639,420);         // Clear Old X-Axis Label

    // Set Title Text Colour
    setcolor(RED);
    settextstyle(DEFAULT_FONT,0,2);

    // Calculate x position of title
    int position = 320 - (textwidth(graphTitle)/2);
    outtextxy(position,3,graphTitle);

    // Set Y-Axis Label style
    setcolor(YELLOW);
    settextstyle(DEFAULT_FONT,1,1);
    outtextxy(10,190,yLabel);

    // Output X-Axis Label
    setcolor(YELLOW);
    settextstyle(DEFAULT_FONT,0,1);
    position = XAxisMin+(XAxisMax-XAxisMin)/2-textwidth(xLabel)/2;
    outtextxy(position,410,xLabel);

    // Set Main Text Colour
    setcolor(WHITE);
    settextstyle(DEFAULT_FONT,0,1);

    // Output Menu Options
    outtextxy(40,430,"n - Create New TC  t - Display Old TC  r - Display Rays");
    outtextxy(40,440,
              "x,y - Rescale       g - Toggle Grid      f - Output To File  q -
Quit");
}


//****************************** DrawGrid *********************************//
//                                                                          //
// Place a grid on the graph  (if grid = 0 remove grid, if grid = 1 add)    //
//                                                                          //
//**************************************************************************//
void DrawGrid(int grid)
{
   int i;        // Counter Variable

   // Set the type on lines used for the grid
   setlinestyle(DOTTED_LINE,0xFFFF,NORM_WIDTH);

   // Determine whether to remove or add the grid
   if (grid==0) setcolor(BLACK);
   else setcolor(DARKGRAY);

   // Y-Axis Grid Lines
   for (i=1;i<11;i++)
      line(XAxisMin+1,YAxisMin-YSpacing*i,XAxisMax,YAxisMin-YSpacing*i);

   // X-Axis Grid Lines
```

```
    for (i=1;i<11;i++)
       line(XAxisMin+XSpacing*i,YAxisMin-1,XAxisMin+XSpacing*i,YAxisMax);

    // Reset the line type
    setlinestyle(SOLID_LINE,0xFFFF,NORM_WIDTH);
}


//****************************** DrawScale ********************************//
//                                                                          //
// Draw the Scale on the axis                                               //
//                                                                          //
//**************************************************************************//
void DrawScale(double xmin, double xmax, double ymin, double ymax)
{
    char msg[80];        // Dummy variable to output lables to screen
    int i;               // Counter variable
    int width;           // Width of message
    double xincrement;   // Increment along X-Axis
    double yincrement;   // Increment along Y-Axis

    // Clear old scale values
    setfillstyle(SOLID_FILL,BLACK);
    bar(20,YAxisMin+5,639,YAxisMin+15);
    bar(20,0,XAxisMin-4,YAxisMin+15);

    // Set axis lable colour
    setcolor(WHITE);

    /****************** x-axis *****************/
    // Output left most lable on x-axis
    sprintf(msg,"%5.1f",xmin);
    width=textwidth(msg);
    outtextxy(XAxisMin-width/2,YAxisMin+6,msg);

    // Output Middle lables
    xincrement = (xmax-xmin)/5;
    for (i=1; i<5; i++) {
       sprintf(msg,"%5.1f",xmin+xincrement*i);
       width=textwidth(msg);
       outtextxy(XAxisMin-width/2+XSpacing*2*i,YAxisMin+6,msg);
    }

    // Output right most lable on x-axis
    sprintf(msg,"%5.1f",xmax);
    width=textwidth(msg);
    outtextxy(XAxisMax-width/2,YAxisMin+6,msg);


    /***************** y-axis ******************/
    // Output top most lable on y-axis
    sprintf(msg,"%5.1f",ymax);
    width=textwidth(msg);
    outtextxy(XAxisMin-4-width,YAxisMax-4,msg);

    // Output Middle lables
    yincrement = (ymax-ymin)/10;
    for (i=1; i<10; i++) {
       sprintf(msg,"%5.1f",ymin+yincrement*i);
       width=textwidth(msg);
       outtextxy(XAxisMin-4-width,YAxisMin-4-YSpacing*i,msg);
    }

    // Output bottom most lable on y-axis
    sprintf(msg,"%5.1f",ymin);
```

```
    width=textwidth(msg);
    outtextxy(XAxisMin-4-width,YAxisMin-4,msg);
}


//***************************** DrawPoints *********************************//
//                                                                            //
// Draw Points for the current line of points (points)                        //
//                                                                            //
//****************************************************************************//
void DrawPoints(lineType *points, double xmin, double xmax,
                          double ymin, double ymax, int colour) {
    int i;                                  // Counter Variable
    int x,y;                              // Variables to plot point location
    double xvalue, yvalue;     // Variables to make calculation easier

    // Set Colour of Points that are drawn
    setcolor(colour);

    // Determine which points are in the graph and which are not
    for (i=0; i<points->number; i++) {
       xvalue = points->x[i];
       yvalue = points->y[i];
       if ((xvalue >= xmin) && (xvalue <=xmax) &&
             (yvalue >= ymin) && (yvalue <= ymax)) {
          points->inGraph[i] = TRUE;
       }
       else {
          points->inGraph[i] = FALSE;
       }
    }

    // Draw the points
    for (i=0; i<points->number; i++) {
       xvalue = points->x[i];
       yvalue = points->y[i];
       if (points->inGraph[i] == TRUE) {
          x = XAxisMin + Round((xvalue-xmin)*((XAxisMax-XAxisMin)/(xmax-xmin)));
          y = YAxisMin + Round((yvalue-ymin)*((YAxisMax-YAxisMin)/(ymax-ymin)));
          circle(x,y,3);
       }
    }
}


//***************************** DrawLine ***********************************//
//                                                                            //
// Draw A Line for the set of points (points)                                 //
//    - This function interpolates to the edges of the graph when the         //
//      first point found in not on an edge of the graph.                     //
//                                                                            //
//****************************************************************************//
void DrawLine(lineType *points, double xmin, double xmax,
                        double ymin, double ymax, int colour) {

   int i;                                       // Counter Variable

   // Set Colour of Points that are drawn
   setcolor(colour);

   // Determine which points are in the graph and which are not
   for (i=0; i<points->number; i++) {
      double xvalue = points->x[i];
      double yvalue = points->y[i];
      if ((xvalue >= xmin) && (xvalue <=xmax) &&
```

```
            (yvalue >= ymin) && (yvalue <= ymax)) {
            points->inGraph[i] = TRUE;
        }
        else {
            points->inGraph[i] = FALSE;
        }
    }

    // Allocate a new array of points
    lineType *pts = (lineType *) malloc(sizeof(lineType));

    // To prevent divide by zero, eliminate points that have same values
    pts->number = 1;
    pts->x[0] = points->x[0];
    pts->y[0] = points->y[0];
    pts->inGraph[0] = points->inGraph[0];
    for (i=1; i<points->number; i++) {
        if ((points->x[i] != points->x[i-1]) && (points->y[i] != points->y[i-1]))
{
            pts->x[pts->number] = points->x[i];
            pts->y[pts->number] = points->y[i];
            pts->inGraph[pts->number] = points->inGraph[i];
            (pts->number)++;
        }
    }

    // Draw the points
    for (i=0; i<pts->number; i++) {
        if (pts->inGraph[i] == TRUE) {
            double xvalue = pts->x[i];
            double yvalue = pts->y[i];
            i n t x = X A x i s M i n +
Round((xvalue-xmin)*((XAxisMax-XAxisMin)/(xmax-xmin)));
            i n t y = Y A x i s M i n +
Round((yvalue-ymin)*((YAxisMax-YAxisMin)/(ymax-ymin)));
            // If this is the first point in the line move to it.
            if (i == 0) moveto(x,y);
            // If the last point is in the graph draw a line from it to this one
            else if (pts->inGraph[i-1] == TRUE) lineto(x,y);
            // If the last point is not in the graph find the intersection of the
            // line joining the last point to the current one on a graph axis.
            else {
                double lastx = pts->x[i-1];
                double lasty = pts->y[i-1];
                if (lastx < xmin) {
                    lasty = (yvalue-lasty)*(xmin-lastx)/(xvalue-lastx)+lasty;
                    lastx = xmin;
                }
                if (lasty > ymax) {
                    lastx = (xvalue-lastx)*(ymax-lasty)/(yvalue-lasty)+lastx;
                    lasty = ymax;
                }
                if (lasty < ymin) {
                    lastx = (xvalue-lastx)*(ymin-lasty)/(yvalue-lasty)+lastx;
                    lasty = ymin;
                }
                i n t x L =
XAxisMin+Round((lastx-xmin)*((XAxisMax-XAxisMin)/(xmax-xmin)));
                i n t y L =
YAxisMin+Round((lasty-ymin)*((YAxisMax-YAxisMin)/(ymax-ymin)));
                moveto(xL,yL);
                lineto(x,y);
            }
        }
        // If last point to current is in graph, we must interpolate
```

```
        // the intersection of a line joining the two points at the edge
        // of the graph.
        else if ((i != 0) && (pts->inGraph[i-1] == TRUE)) {
            double xvalue = pts->x[i];
            double yvalue = pts->y[i];
            double lastx = pts->x[i-1];
            double lasty = pts->y[i-1];
            if (xvalue > xmax) {
                yvalue = yvalue-((yvalue-lasty)*(xvalue-xmax)/(xvalue-lastx));
                xvalue = xmax;
            }
            if (yvalue > ymax) {
                xvalue = xvalue-((xvalue-lastx)*(yvalue-ymax)/(yvalue-lasty));
                yvalue = ymax;
            }
            if (yvalue < ymin) {
                xvalue = xvalue-((xvalue-lastx)*(yvalue-ymin)/(yvalue-lasty));
                yvalue = ymin;
            }
            i n t          x          =          X A x i s M i n          +
Round((xvalue-xmin)*((XAxisMax-XAxisMin)/(xmax-xmin)));
            i n t          y          =          Y A x i s M i n          +
Round((yvalue-ymin)*((YAxisMax-YAxisMin)/(ymax-ymin)));
            lineto(x,y);
        } // End of else if
    } // End of for loop

    // Free old list of points
    free(pts);
} // End of function


//***************************** DrawTransferChar *****************************//
//                                                                              //
// Draw a transfer characteristic for a set of rays.                            //
//                                                                              //
//******************************************************************************//
tcType *DrawTransferChar(rayType *rays, int grid) {
    tcType *tc;          // tc created in this program
    double xobject;  // Distance to the object in metres
    char temp[40];   // Temporary string
    int i;                   // counter variable
    lineType *oneTC;     // One ray to be drawn

    // Allocate memory for tc structure
    tc = (tcType *) malloc(sizeof(tcType));

    // Initialize tc structure
    tc->number = rays->numberRays;
    tc->apparent = (double *) malloc(sizeof(double)*tc->number);
    tc->actual = (double *) malloc(sizeof(double)*tc->number);

    // Get the Distance to the object in metres
    char *inputStr;
    int valid = 0;
    while (valid == 0) {
        inputStr = prompt("Input distance to object (km): ");
        if ((inputStr[0] == '.') || isdigit(inputStr[0])) {
            for (i=0; (((inputStr[i] == '.') || isdigit(inputStr[i])) &&
                                (i < strlen(inputStr))); i++) {
                temp[i] = inputStr[i];
            }
            temp[i] = 0;
            valid = 1;
        }
```

```
        if (valid == 0) {
            message("Invalid input values... Try again.");
            sleep(2);
        }
    }
    sscanf(temp,"%lf",&xobject);

    // Conver object distance from km to metres
    xobject *= 1000;

    // Clear the Drawing Area
    setfillstyle(SOLID_FILL,BLACK);
    bar(XAxisMin+1,YAxisMax-3,XAxisMax+3,YAxisMin-1);

    // Redraw grid just erased
    DrawGrid(grid);

    // Calculate the apparent values
    for (i=0; i<tc->number; i++) {
        tc->apparent[i] = xobject*tan(rays->angles[i]/3437.747) + rays->eyeLevel;
    }

    // Calculate the actual values
    int hit,location;
    int numpoints = rays->xmax/rays->xstep+1;
    hit=0;                              // hit=1 if xobject is on interval boundary
    // Check if distance is on a boundary
    for (i=0;i<numpoints;i++)  if ((rays->xstep*i)==xobject) {location=i; hit=1;}
    // If distance is not on a boundary find closest boundary less than distance
    if (hit==0) {
        for (i=0; i<numpoints;i++)
            if (((i*rays->xstep)<xobject) &&
                    (((i+1)*rays->xstep)>xobject)) location=i;
    }
    // If distance is on a boundary use it otherwise interpolate
    if (hit==1) {
        for (i=0;i<tc->number;i++)     tc->actual[i]=rays->ray[i][location];
    }
    else {
        for (i=0;i<tc->number;i++)
            tc->actual[i]=xobject*(rays->ray[i][location+1] -
                                                    rays->ray[i][location])/rays->xstep
+
                                    rays->ray[i][location] -
                                    location*rays->ray[i][location+1] +
                                    location*rays->ray[i][location];
    }

    // Allocate the array oneTC
    oneTC = (lineType *) malloc(sizeof(lineType));

    // Set up the number of elements in the array oneRay
    oneTC->number = tc->number;

    // Copy TC values into a line to plot
    for (i=0; i<oneTC->number; i++) {
           oneTC->y[i] = tc->apparent[i];
           oneTC->x[i] = tc->actual[i];
    }

    // Find the Maximum values of the TC
    tc->graphxmax = -500;
    tc->graphymax = -500;
    tc->graphxmin = 500;
    tc->graphymin = 500;
```

```
        for (i=0; i<oneTC->number; i++) {
            if (tc->apparent[i] > tc->graphymax) tc->graphymax = tc->apparent[i];
            if (tc->actual[i] > tc->graphxmax) tc->graphxmax = tc->actual[i];
            if (tc->apparent[i] < tc->graphymin) tc->graphymin = tc->apparent[i];
            if (tc->actual[i] < tc->graphxmin) tc->graphxmin = tc->actual[i];
    }

    // If the max and min are the same... change them slightly
    if ((tc->graphxmax-tc->graphxmin) < 1) {
        tc->graphxmax += 5.0;
        tc->graphxmin -= 5.0;
    }
    if ((tc->graphymax-tc->graphymin) < 1) {
        tc->graphymax += 5.0;
        tc->graphymin -= 5.0;
    }

    // Draw the TC
    DrawLine(oneTC,tc->graphxmin,tc->graphxmax,tc->graphymin,tc->graphymax,CYAN);

DrawPoints(oneTC,tc->graphxmin,tc->graphxmax,tc->graphymin,tc->graphymax,YELLOW);

    // Free data used to draw TC
    free(oneTC);

    // return the tc to the main program
    return tc;
}


//*************************** RedrawTransferChar ***************************//
//                                                                          //
// Redraw the tc with new axis values.                                      //
//                                                                          //
//**************************************************************************//
void RedrawTransferChar(tcType *tc,  int grid, int newTC) {

    int i;                          // Counter Variables
    lineType *oneTC;        // One ray to be drawn

    // Clear the Drawing Area
    if (newTC == 1) {
        setfillstyle(SOLID_FILL,BLACK);
        bar(XAxisMin+1,YAxisMax-3,XAxisMax+3,YAxisMin-1);
        DrawGrid(grid);
    }

    // Allocate the array oneTC
    oneTC = (lineType *) malloc(sizeof(lineType));

    // Set up the number of elements in the array oneRay
    oneTC->number = tc->number;

    // Copy TC values into a line to plot
    for (i=0; i<oneTC->number; i++) {
          oneTC->y[i] = tc->apparent[i];
          oneTC->x[i] = tc->actual[i];
    }

    // Draw the TC
    if (newTC == 1) {

DrawLine(oneTC,tc->graphxmin,tc->graphxmax,tc->graphymin,tc->graphymax,CYAN);
      DrawPoints(oneTC,tc->graphxmin,tc->graphxmax,tc->graphymin,
                    tc->graphymax,YELLOW);
```

```
    }
    else {
        DrawLine(oneTC,tc->graphxmin,tc->graphxmax,tc->graphymin,tc->graphymax,RED);
DrawPoints(oneTC,tc->graphxmin,tc->graphxmax,tc->graphymin,tc->graphymax,GREEN);
    }

    // Free data used to draw TC
    free(oneTC);
}


//***************************** DrawCompareTC ******************************//
//                                                                            //
// Loads a tc from a file and compares it to and existing one.                //
//                                                                            //
//****************************************************************************//
void DrawCompareTC(tcType *tc, char *compareFile, int grid) {
    int i;    // Counter Variable
    FILE *fp = fopen(compareFile,"r");
    tcType *compareTC = (tcType *) malloc(sizeof(tcType));

    // Get number of lines in file
    compareTC->number = 0;
    char buffer[81];
    while (fgets(buffer,80,fp) != NULL) (compareTC->number)++;
    fclose(fp);

    // Allocate data for tc
    compareTC->apparent = (double *) malloc(sizeof(double)*compareTC->number);
    compareTC->actual = (double *) malloc(sizeof(double)*compareTC->number);

    // Get data in file
    fp = fopen(compareFile,"r");
    for (i=0; i<compareTC->number; i++) {
        double lf1,lf2;
        if (fscanf(fp,"%lf %lf",&lf1,&lf2) != 2) break;
        compareTC->apparent[i] = lf1;
        compareTC->actual[i] = lf2;
    }
    fclose(fp);

    // If all the data was not read exit case
    if (i != compareTC->number) {
        free(compareTC->apparent);
        free(compareTC->actual);
        free(compareTC);
        message("Error in reading file.");
        sleep(2);
        message("");
        return;
    }

    // Set Max and Min values
    compareTC->graphxmax = tc->graphxmax;
    compareTC->graphxmin = tc->graphxmin;
    compareTC->graphymax = tc->graphymax;
    compareTC->graphymin = tc->graphymin;

    // Print comparison on screen
    RedrawTransferChar(tc, grid, 1);
    RedrawTransferChar(compareTC, grid, 0);

    // Calculate absolute error if there is equal points
    if (tc->number <= compareTC->number) {
```

```
        double total = 0;
        for (i=0; i<tc->number; i++) {
            total += fabs(compareTC->actual[i] - tc->actual[i]);
        }
        total /= tc->number;
        char dummy[50];
        sprintf(dummy,"Absolute Error: %lf",total);
        message(dummy);
    }

    free(compareTC->apparent);
    free(compareTC->actual);
    free(compareTC);
}


//******************************** DrawRays *********************************//
//                                                                           //
// Draw the rays in a ray tracing.                                           //
//                                                                           //
//***************************************************************************//
void DrawRays(rayType *rays, int grid) {
    int i,j;                // Counter Variables
    lineType *oneRay;    // One ray to be drawn

    // Clear the Drawing Area
    setfillstyle(SOLID_FILL,BLACK);
    bar(XAxisMin+1,YAxisMax-3,XAxisMax+3,YAxisMin-1);

    // Redraw grid just erased
    DrawGrid(grid);

    // Allocate the array oneRay
    oneRay = (lineType *) malloc(sizeof(lineType));

    // Set up the number of elements in the array oneRay
    oneRay->number = rays->xmax/rays->xstep+1.0;

    for (i=0; i<rays->numberRays; i++) {
        for (j=0; j<(rays->xmax/rays->xstep+1); j++) {
            oneRay->x[j]=(j*rays->xstep)/1000.0;
            oneRay->y[j]=rays->ray[i][j];
        }
        DrawLine(oneRay,rays->graphxmin,rays->graphxmax,
                                    rays->graphymin,rays->graphymax,CYAN);
    }

    // Free data used to draw Ray
    free(oneRay);
}


//***************************** SetGraphics *********************************//
//                                                                           //
// Setup the graphics screen for writing                                     //
//                                                                           //
//***************************************************************************//
void SetGraphics(void) {
    int errorcode,graphdriver=VGA,graphmode=VGAHI;

    /***** set up graphics *****/
    errorcode=registerbgidriver(EGAVGA_driver);
    if (errorcode<0) {
        printf(" Graphics error in registering driver!!");
        exit(0);
```

```
    }

    // Initialize the Graphics Screen
    initgraph(&graphdriver,&graphmode,"");

    // Check the results of initialization
    errorcode=graphresult();
    if (errorcode != 0) {
       printf(" Graphics error in initiating graphics!!");
       exit(0);
    }

    /****** clear screen ******/
    cleardevice();
}


//******************************** Round *********************************//
//                                                                          //
// Round a double to become an integer value                                //
//                                                                          //
//**************************************************************************//
int Round(double number)
{
   return(number + 0.5);
}


//******************************** Message *******************************//
//                                                                          //
// Output a message on the message line                                     //
//                                                                          //
//**************************************************************************//
void message(char *text) {

   // Set message colour to white
   setcolor(LIGHTMAGENTA);

   // Output message or erase old one if text is blank
   if (strcmp(text,"")) {
      setfillstyle(SOLID_FILL,BLACK);
      bar(10,465,639,475);
      outtextxy(40,465,text);
   }
   else {
      setfillstyle(SOLID_FILL,BLACK);
      bar(10,465,639,475);
   }

   // Set Colour back to normal
   setcolor(WHITE);
}


//******************************** Prompt ********************************//
//                                                                          //
// Output a prompt string and return the response                           //
//                                                                          //
//**************************************************************************//
char *prompt(char *text) {
   // Set up a couple constants
   const int characterSize = 8;
   const int SPACE = 8;
   const int start = 40;
```

```
    // Output Prompt
    message(text);

    // Create variable to return to the calling function
    char *inputString = (char *) malloc(sizeof(char)*35);

    // Get user text
    int count=0;                        // counter variable
    char character;                     // input character
    int width=0;                        // width of previous text
    char smallStr[2] = " ";         // small string for outtextxy function
    while ((character=getch()) != 13) {
       if ((isprint(character)) && (count < 35)) {
          width = textwidth(text)+ SPACE + count*characterSize + start;
          smallStr[0] = character;
          outtextxy(width,465,smallStr);
          inputString[count] = character;
          count++;
       }
       else if (((character == 8)  || (character == 127)) && (count > 0)) {
          count--;
          width = textwidth(text)+ SPACE + count*characterSize + start;
          bar(width,465,width+SPACE,475);
       }
    }
    inputString[count] = '\0';

    // Clear Prompt String
    message("");

    // Return input string
    return inputString;
}


/******************************** SaveRay *********************************/
/*                                                                        */
/* The program saves a set of rays to a file                              */
/*                                                                        */
/**************************************************************************/
void SaveRay(char *fileName, rayType *rays, int addProfile) {
    FILE *OUT;  // Pointer to the output file stream
    int i,j;      // Counter variables

    // Open the file
    OUT = fopen(fileName,"wb");

    /***  write ray file  ***/
    if (addProfile == 0) {
       fprintf(OUT,"%8.2f\r\n",rays->time);
       fprintf(OUT,"%8.2f\r\n",rays->xstep);
       fprintf(OUT,"%8.2f\r\n",rays->xmax);
       fprintf(OUT,"%8.2f\r\n",rays->ymax);
       fprintf(OUT,"%d\r\n",rays->numberRays);
       for (i=0;i<rays->numberRays;i++) f p r i n t f ( O U T , "
%10.6f\r\n",rays->angles[i]);
       fprintf(OUT,"\r\n");
       for (i=0;i<rays->numberRays;i++) {
          for (j=0;j<(rays->xmax/rays->xstep+1);j++)
             fprintf(OUT,"%9.5f\r\n",rays->ray[i][j]);
          fprintf(OUT,"\r\n");
       }
    }
    else {
       int profiles;                    // Number of profiles
```

```
 double slopes[25];              // Slopes of the profiles
 int steps[25];                     // Steps of the profiles
 int dummyInt;                      // Dummy variable to read ints
 double dummyDouble;             // Dummy variable to read doubles
 char dummyString[80];           // Dummy variable to read strings

 // Get file name extension
 char *fileExt = strchr(rays->name,'.')+1;

 // Get necessary Data from file
 if (!strcmp(fileExt,"tp")) {
    // Graph Consists of one profile
    profiles = 1;
    // What is the slope of this profile
    slopes[0]=rays->slope;
 }
 else if (!strcmp(fileExt,"arf")) {
    FILE *IN;     // Pointer to the input file stream
    // Open file to read in data
    IN = fopen(rays->name,"r");
    // Get information from file
    GetLine(IN,dummyString);
    sscanf(dummyString,"%d\n",&profiles);
    for (i=0;i<4;i++) GetLine(IN,dummyString);
    GetLine(IN,dummyString);
    sscanf(dummyString,"%d\n",&dummyInt);
    for (i=0;i<dummyInt;i++) GetLine(IN,dummyString);
    for (j=0; j<profiles; j++) {
       GetLine(IN,dummyString);
       sscanf(dummyString,"%d\n",&dummyInt);
       steps[j] = dummyInt;
       GetLine(IN,dummyString);
       if (toupper(dummyString[0])=='F') {
          GetLine(IN,dummyString);
          sscanf(dummyString,"%lf\n",&dummyDouble);
          slopes[j]=dummyDouble;
       }
       GetLine(IN,dummyString);
    }
    // Close file
    fclose(IN);
 }
 else {
    fclose(OUT);
    return;
 }

 // Plot the distance steps in kilometres
 fprintf(OUT,"%9s ","Steps");
 for (j=0; j<(rays->xmax/rays->xstep+1); j++) {
    fprintf(OUT,"%9.3lf ",(j*(rays->xstep)/1000.0));
 }
 fprintf(OUT,"\r\n");

 // Convert all slopes to radians
 for (i=0; i<profiles; i++) slopes[i] /= 3437.747;

 // Plot all temperture levels
 for (i=-350; i<=rays->ymax; i+=50) {
    fprintf(OUT,"%8dT ",i);
    double height = i;
    if (height < 0) fprintf(OUT,"%9.3lf ",0.0);
    else fprintf(OUT,"%9.3lf ",height);
    for (int k=0; k<profiles; k++) {
       for (j=0; j<steps[k]; j++) {
```

```
                height += rays->xstep*tan(slopes[k]);
                if (height > rays->ymax) break;
                if (height < 0) fprintf(OUT,"%9.3lf ",0.0);
                else fprintf(OUT,"%9.3lf ",height);
            }
            if (height > rays->ymax) break;
        }
        fprintf(OUT,"\r\n\r\n");
    }

    // Offset Spaceing by one more
    fprintf(OUT,"\r\n");

    // Plot the rays
    for (i=0;i<rays->numberRays;i++) {
        fprintf(OUT,"%8dR ",i);
        for (j=0;j<(rays->xmax/rays->xstep+1);j++) {
            fprintf(OUT,"%9.5f ",rays->ray[i][j]);
        }
        fprintf(OUT,"\r\n\r\n");
    }
}

    // Close the file
    fclose(OUT);
}


/******************************** SaveTC *********************************/
/*                                                                       */
/* The program saves a transfer characteristic to a file                 */
/*                                                                       */
/*************************************************************************/
void SaveTC(char *fileName, tcType *tc) {
    FILE *IN;     // Pointer to the file stream
    int i;        // Counter variables

    // Open the file
    IN = fopen(fileName,"wb");

    /***  write .TC file  ***/
    for (i=0;i<tc->number;i++)
        fprintf(IN,"%10.5f   %10.5f \r\n",tc->apparent[i],tc->actual[i]);

    // Close the file
    fclose(IN);
}


//***************************** ValidFileName ******************************//
//                                                                          //
// Function checks if file name is passed to it is valid.                   //
// Returns: 0 - When file name is not valid                                 //
//          1 - When file name is valid but already exists (without ext.)   //
//          2 - When file name is valid but already exists (with ext.)      //
//          3 - When file name is valid and extension is not added          //
//          4 - When file name is valid and extension is added              //
//                                                                          //
//**************************************************************************//
int ValidFileName(char *file, char *extension, char readWrite) {
    char tempName[36];
    char filePath[36];
    char fileName[9];
    char fileExtension[5];
```

```
// Initialize to random string
filePath[35] = 0;
fileName[8]=0;
fileExtension[4]=0;
strcpy(tempName,file);

// Find the file path
if (strrchr(tempName,'\\') != NULL) {
    strcpy(filePath,tempName);
    // Remove file name and extension from path
    char *tempPtr = strrchr(filePath,'\\');
    tempPtr++;
    tempPtr[0] = 0;
    // Remove file path from tempName
    strcpy(tempName,(strrchr(file,'\\')+1));
}
else strcpy(filePath,"");

// Find the file extension
if (strchr(tempName,'.') != NULL) {
    strncpy(fileExtension,strchr(tempName,'.'),4);
    // Remove file extension from tempName
    char *tempPtr = strchr(tempName,'.');
    tempPtr[0] = 0;
}
else {
    strcpy(fileExtension,"");
}

// Find the file Name
strncpy(fileName,tempName,8);

// We now have a valid file name and extension
strcpy(file,filePath);
strcat(file,fileName);
strcat(file,fileExtension);

// Check to see if filename is valid
FILE *fp;
if (readWrite == 'r') {
    if (!strcmp(fileExtension,"")) {
        // Check if file name is valid without extension
        fp = fopen(file,"r");
        if (fp != NULL) {
            fclose(fp);
            return 1;
        }
        else fclose(fp);
        // Check if file name is valid with extension added
        strcat(file,fileExtension);
        fp = fopen(file,"r");
        if (fp != NULL) {
            fclose(fp);
            return 2;
        }
        else {
            fclose(fp);
            return 0;
        }
    }
    else {
        // Check if file name is valid
        fp = fopen(file,"r");
        if (fp != NULL) {
            fclose(fp);
```

```
                return 1;
            }
            else {
                fclose(fp);
                return 0;
            }
        }
    }
    else {
        int returnAdj = 0;
        // Add file extension if necessary
        if (!strcmp(fileExtension,"")) {
            strcat(file,extension);
            returnAdj = 1;
        }
        // Check to see if file exists
        if ((fp = fopen(file,"r")) != NULL) {
            fclose(fp);
            return (1+returnAdj);
        }
        fclose(fp);
        // Try to open file
        if ((fp = fopen(file,"w")) != NULL) {
            fclose(fp);
            return(3+returnAdj);
        }
        fclose(fp);
        return (0);
    }
}
```

Appendix C

Mirage Simulator Program

This mirage simulation program developed for this thesis consists of eight modules: *rgbmir.rc, rgbmir.h, rgbmir.cpp, rgbmirtc.h, rgbmirtc.cpp, rgbmirbm.h, rgbmirbm.cpp, rgbmirdv.h.* The program was compiled with Borland C++ version 4.0 and is included in this appendix for reference. The modules rgbmirbm.h and rgbmirbm.cpp are based on an image acquisition program provided in the book *Imaging and Animation for Windows* [Bar1].

It was compiled as a windows program and must be executed in this environment. When it is executed a window appears with a menu bar. To simulate a mirage, a non-mirage image is loaded into the program by selecting *Load* from the file menu. Once loaded, the image appears in the main program window. The next step is to locate and enter the maximum and minimum elevations in the image. This is done by clicking on the peak elevation with the left mouse button and entering the appropriate height in a user dialog box. The same procedure is followed with the right mouse button for the horizon height. Next, the transfer characteristic is loaded by selecting *Load T.C.* from the mirage menu. Another dialog box appears that allows a transfer characteristic file to be selected and loaded. This file should use an horizon elevation at zero metres. Once the program is loaded, the mirage can be simulated by selecting *Simulate* from the mirage menu. When the simulation is complete, the original image is converted into a mirage image. This image can then be saved to disk using the *Save* option from the file menu. The process can be repeated for as many mirages as required.

```
//  RRRRRRR   GGGGGGG BBBBBBB  MM     MM IIII RRRRRRR
//  RRRRRRRR GGGGGGGG BBBBBBBB MMM   MMM  II  RRRRRRRR
//  RR    RR GG       BB    BB MMMMMMMM  II  RR    RR
//  RRRRRRRR GG   GGG BBBBBBBB MMMMMMMM  II  RRRRRRRR
//  RRRRRRR  GG   GGG BBBBBBBB MM MM MM  II  RRRRRRR
//  RR RR    GG    GG BB    BB MM    MM  II  RR  RR
//  RR   RR  GGGGGGGG BBBBBBBB MM    MM  II  RR   RR
//  RR    RR  GGGGGGG BBBBBBB  MM    MM IIII RR    RR

// rgbmir.rc - RESOURCE FILE FOR MIRAGE SIMULATOR

#ifndef WORKSHOP_INVOKED
    #include <windows.h>
#endif

#include "docview.rc"
#include <owl\except.rc>
#include <owl\mdi.rh>
#include <owl\statusba.rc>

#define CM_SETTINGS            308
#define CM_LOADTC              309
#define CM_SIMULATE            310
#define CM_ABOUT          311
#define CM_HIGHQUALITY    312

#define EV_PIXELINFO              1000
#define EV_LOADMSG                1001
#define EV_SAVEMSG                1002
#define    EV_PREPARING           1003
#define    EV_SIMULATING          1004
#define EV_TOPPTMSG               1005
#define EV_BOTPTMSG               1006
#define EV_TCMSG                  1007
#define EV_CHANGEIMAGESIZE    1008

#define FIRST_MESSAGE      EV_PIXELINFO
#define LAST_MESSAGE       EV_CHANGEIMAGESIZE

#define IDD_ABOUT          211

#ifdef RC_INVOKED
     #include <owl\inputdia.rc>

    COMMANDS MENU {
        POPUP "&File" {
            MENUITEM    "&Open...",                    CM_FILEOPEN
            MENUITEM    "&Save",                       CM_FILESAVEAS
            MENUITEM    "&Revert To Saved",    CM_FILEREVERT
            MENUITEM    "&Close",                      CM_FILECLOSE
            MENUITEM    SEPARATOR
            MENUITEM    "E&xit",                       CM_EXIT
        }
        POPUP "&Mirage" {
            MENUITEM    "&Settings...",            CM_SETTINGS
            MENUITEM    "&Load T.C.",              CM_LOADTC
            MENUITEM    "&High Quality",       CM_HIGHQUALITY
            MENUITEM    "S&imulate",               CM_SIMULATE
        }
        POPUP "&Window"    {
            MENUITEM  "&Cascade",           CM_CASCADECHILDREN
            MENUITEM  "&Tile",            CM_TILECHILDREN
            MENUITEM  "Arrange &Icons",   CM_ARRANGEICONS
            MENUITEM  "C&lose All",       CM_CLOSECHILDREN
            MENUITEM  "Add &View",        CM_VIEWCREATE
        }
        POPUP "&Help"  {
            MENUITEM "&About",                         CM_ABOUT
        }
    }

    STRINGTABLE {
        CM_SETTINGS,        "Check the settings for the selected image"
        CM_LOADTC,          "Load a transfer characteristic for the selected image"
```

```
        CM_HIGHQUALITY,     "Mirage quality is set to high when checked"
        CM_SIMULATE,    "Simulate the mirage based on information entered"
        CM_ABOUT,           "Open an information dialog box"
        CM_FILESAVEAS, "Save mirage image to a file"
        CM_FILEREVERT, "Revert back to original image"
    }

    // About Dialog
    IDD_ABOUT DIALOG 37, 25, 170, 95
    STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
    CAPTION "About Image Viewer"
    FONT 8, "MS Sans Serif"
    {
        DEFPUSHBUTTON "Ok", IDOK, 60, 63, 50, 14
        CTEXT "Copyright 1994 by Tom Legal", -1, 10, 45, 150, 8
        CTEXT "Colour Mirage Simulator", -1, 10, 15, 150, 8
        CTEXT "(Based on HEXMIR by Wanye Silvester)", -1, 10, 25, 150, 8
    }

    CM_FILEOPEN BITMAP LOADONCALL MOVEABLE
    {
        '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
        '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
        '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
        '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
        '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
        '00 00 80 80 80 00 C0 C0 C0 00 00 00 FF 00 00 FF'
        '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
        '00 00 FF FF FF 00 88 88 88 88 88 88 88 88 88 88'
        '46 00 80 00 00 00 00 00 88 88 88 88 00 00 80 87'
        '77 77 77 70 88 88 88 88 11 02 80 F8 88 88 88 70'
        '88 88 88 88 08 00 80 F9 98 88 88 70 88 88 88 88'
        '46 00 80 FF FF FF FF 80 88 88 88 88 00 00 80 00'
        '00 00 00 00 88 88 88 88 11 00 88 88 88 88 88 88'
        '88 88 88 88 08 30 88 80 08 88 88 88 88 88 88 88'
        '46 00 88 80 08 88 88 88 88 88 88 88 88 00 00 88 80'
        '08 88 88 84 44 44 44 48 11 00 88 80 08 80 88 84'
        'EF EF EF 48 08 00 88 80 07 80 08 84 F4 44 4E 48'
        '00 00 88 87 00 00 00 84 EF EF EF 48 55 55 88 88'
        '70 00 00 84 F4 44 4E 48 00 00 88 88 88 80 08 84'
        'EF EF EF 48 88 88 88 88 88 80 88 84 F4 4E 44 48'
        '00 00 88 88 88 88 88 84 EF EF 44 88 00 00 88 88'
        '88 88 88 84 44 44 48 88 00 00 88 88 88 88 88 88'
        '88 88 88 88 00 00'
    }

    CM_FILESAVEAS BITMAP LOADONCALL MOVEABLE
    {
        '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
        '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
        '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
        '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
        '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
        '00 00 80 80 80 00 C0 C0 C0 00 00 00 FF 00 00 FF'
        '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
        '00 00 FF FF FF 00 88 88 88 88 88 88 88 88 88 88'
        '46 00 88 88 88 88 88 84 44 44 44 48 00 00 88 88'
        '88 88 88 84 FF FF FF 48 11 00 88 88 88 88 88 84'
        'F4 44 4F 48 08 00 88 88 88 88 88 84 FF FF FF 48'
        '00 00 88 88 88 88 88 84 F4 44 4F 48 55 55 88 88'
        '88 88 88 84 FF FF FF 48 00 00 88 88 88 88 88 84'
        'F4 4F 44 48 88 88 88 88 88 88 88 84 FF FF 44 88'
        '00 00 88 88 88 88 88 84 44 44 48 88 00 00 84 44'
        '44 44 48 88 88 88 88 88 00 00 84 EF EF EF 48 88'
        '88 00 88 88 00 00 84 F4 44 4E 48 88 80 00 08 88'
        '00 00 84 EF EF EF 48 88 00 00 00 88 00 00 84 F4'
        '44 4E 48 88 88 00 88 88 00 00 84 EF EF EF 48 88'
        '87 00 88 88 11 11 84 F4 4E 44 48 00 00 07 88 88'
        '00 00 84 EF EF 44 88 00 00 78 88 88 00 00 84 44'
        '44 48 88 88 88 88 88 88 00 00 88 88 88 88 88 88'
        '88 88 88 88 0B 00'
    }

CM_SIMULATE BITMAP
```

```
 {
  '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
  '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
  '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
  '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
  '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
  '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
  '00 00 FF FF FF 00 77 77 77 77 77 77 77 77 77 77'
  '00 00 77 77 77 77 77 77 07 77 77 77 00 00 77 77'
  '77 77 77 70 B0 77 77 77 00 00 77 77 77 77 77 0B'
  'B0 77 77 77 00 00 77 77 77 77 70 BB 07 77 77 77'
  '00 00 77 77 77 77 0B BB 07 77 77 77 00 00 77 77'
  '77 70 BB B0 00 07 77 77 00 00 77 77 77 0B BB BB'
  'BB B0 77 77 00 00 77 77 77 70 0B BB BB 07 77 77'
  '00 00 77 77 77 70 BB BB B0 77 77 77 00 00 77 77'
  '77 0B BB BB 07 77 77 77 00 00 77 77 70 BB BB B0'
  '00 07 77 77 00 00 77 77 0B BB BB BB BB B0 77 77'
  '00 00 77 77 70 00 BB BB BB 07 77 77 00 00 77 77'
  '77 0B BB BB B0 77 77 77 00 00 77 77 70 BB BB BB'
  '07 77 77 77 00 00 77 77 0B BB BB B0 77 77 77 77'
  '00 00 77 70 BB BB BB 07 77 77 77 77 00 00 77 77'
  '00 00 00 77 77 77 77 77 00 00 77 77 77 77 77 77'
  '77 77 77 77 00 00'
 }

 CM_LOADTC BITMAP
 {
  '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
  '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
  '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
  '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
  '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
  '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
  '00 00 FF FF FF 00 77 77 77 77 77 77 77 77 77 77'
  '00 00 70 00 00 00 00 00 77 77 77 77 00 00 70 78'
  '88 88 88 80 77 77 77 77 00 00 70 F7 77 77 77 80'
  '77 77 77 77 00 00 70 F9 97 77 77 80 77 77 77 77'
  '00 00 70 FF FF FF FF 70 77 77 77 77 00 00 70 00'
  '00 00 00 00 77 77 77 77 00 00 77 77 77 77 77 77'
  '77 77 77 77 00 00 77 70 07 77 77 77 77 77 77 77'
  '00 00 77 70 07 77 77 77 77 77 77 77 00 00 77 70'
  '07 77 77 74 44 44 44 47 00 00 77 70 07 70 77 74'
  'AA 00 AA 47 00 00 77 70 08 70 07 74 A0 AA AA 47'
  '00 00 77 78 00 00 00 74 A0 00 0A 47 00 00 77 77'
  '80 00 00 74 AA AA 0A 47 00 00 77 77 77 70 07 74'
  'AA 00 AA 47 00 00 77 77 77 70 77 74 00 0A 44 47'
  '00 00 77 77 77 77 77 74 AA AA 44 77 00 00 77 77'
  '77 77 77 74 44 44 47 77 00 00 77 77 77 77 77 77'
  '77 77 77 77 00 00'
 }
 #endif  // RC_INVOKED
```

```
//  RRRRRRR   GGGGGGG BBBBBBB  MM     MM IIII RRRRRRR
//  RRRRRRRR GGGGGGGG BBBBBBBB MMM   MMM  II  RRRRRRRR
//  RR     RR GG        BB     BB MMMMMMMM  II  RR     RR
//  RRRRRRRR GG    GGG BBBBBBBB MMMMMMMMM  II  RRRRRRRR
//  RRRRRRR  GG    GGG BBBBBBBB MM MM MM  II  RRRRRRR
//  RR RR    GG     GG BB     BB MM     MM  II  RR  RR
//  RR   RR  GGGGGGGG BBBBBBBB MM     MM  II  RR   RR
//  RR    RR  GGGGGGG BBBBBBB  MM     MM IIII RR    RR

// rgbmir.h - MAIN HEADER FILE FOR MIRAGE SIMULATOR

/*****************************************************************************/
/*                                                                           */
/*                                Include Files                              */
/*                                                                           */
/*****************************************************************************/
// OWL Includes
#include <owl\owlpch.h>
#include <owl\applicat.h>
#include <owl\decmdifr.h>
#include <owl\dialog.h>
#include <owl\controlb.h>
#include <owl\buttonga.h>
#include <owl\statusba.h>
#include <owl\docmanag.h>
#include <owl\mdichild.h>
#include <owl\scrollba.h>  // Include for scroll bars

// C and C++ Includes
#include <stdio.h>


/*****************************************************************************/
/*                                                                           */
/*                            TMDIChildSB Class                              */
/*                                                                           */
/*****************************************************************************/
class TMDIChildSB : public TMDIChild
{
    public:
        TMDIChildSB(TMDIClient &parent, const char far *title =0,
                    TWindow *clientWnd = 0, BOOL shrinkToClient = FALSE,
                    TModule* module = 0);

    private:
        // Scroll Bars
        TScrollBar *hScroller;
        TScrollBar *vScroller;

        // Variable to keep track how big image is
        unsigned width;
        unsigned height;

        // Scroll Bar Handling Functions
        void addScrollBars(BOOL horiz, BOOL vert, TSize& size);
        void deleteScrollBars(BOOL horiz, BOOL vert);

        // Event Handling Functions
        LRESULT    EvChangeImageSize(WPARAM wParam, LPARAM lParam);
        void EvHScroll(UINT scrollCode, UINT thumbPos, HWND);
        void EvVScroll(UINT scrollCode, UINT thumbPos, HWND);
        void EvSize(UINT sizeType, TSize& size);

    DECLARE_RESPONSE_TABLE(TMDIChildSB);
};

// Set up border and scroll bar widths
const int  BORDERWIDTH = 8;
const int BORDERHEIGHT = 27;
const int SCROLLWIDTH = 17;
const int SCROLLHEIGHT = 17;
/*****************************************************************************/
/************************** End TMDIChildSB Class ****************************/
/*****************************************************************************/
```

```
/*****************************************************************************/
/*                                                                           */
/*                            TViewFrame Class                               */
/*                                                                           */
/*****************************************************************************/
const UINT  BLANK_HELP = UINT_MAX;

class TViewFrame : public TDecoratedMDIFrame {
    public:
        TViewFrame(const char far*  title,
                           TResId                       menuResId,
                           TMDIClient&      clientWnd = *new TMDIClient,
                           BOOL             trackMenuSelection = FALSE,
                           TModule*         module = 0);

    protected:
        BOOL    TrackMenuSelection;
        UINT    MenuItemId;

        void    EvMenuSelect(UINT menuItemId, UINT flags, HMENU hMenu);
        void  EvEnterIdle(UINT source, HWND hWndDlg);

    DECLARE_RESPONSE_TABLE(TViewFrame);
};
/*****************************************************************************/
/************************** End TViewFrame Class *****************************/
/*****************************************************************************/


/*****************************************************************************/
/*                                                                           */
/*                              TViewApp Class                               */
/*                                                                           */
/*****************************************************************************/
class TViewApp : public TApplication {
    public:
        TViewApp() : TApplication() {}

        BOOL PumpWaitingMDIMessages();

    protected:
        // Override methods of TApplication
        void InitInstance();
        void InitMainWindow();

        // Variables used in this application
        TMDIClient* Client;
        TTextGadget* tPtGadget;
        TTextGadget* bPtGadget;
        TTextGadget* pixelGadget;
        TTextGadget* valueGadget;
        TStatusBar* sb;

        // Event handlers
        void EvNewView  (TView& view);
        void EvCloseView(TView& view);
        void EvDropFiles(TDropInfo dropInfo);
        void CmAbout();
        LRESULT EvPixelInfo(WPARAM wParam, LPARAM lParam);
        LRESULT EvLoadMSG(WPARAM wParam, LPARAM);
        LRESULT EvSaveMSG(WPARAM wParam, LPARAM);
        LRESULT EvPreparing(WPARAM wParam, LPARAM);
        LRESULT EvSimulating(WPARAM wParam, LPARAM);
        LRESULT    EvBotPtMSG(WPARAM wParam, LPARAM lParam);
        LRESULT    EvTopPtMSG(WPARAM wParam, LPARAM lParam);
        LRESULT EvTCMSG(WPARAM wParam, LPARAM);

        DECLARE_RESPONSE_TABLE(TViewApp);
};
/*****************************************************************************/
/************************** End TViewApp Class *******************************/
/*****************************************************************************/
```

```
//  RRRRRRR   GGGGGGG BBBBBBB  MM     MM IIII RRRRRRR
//  RRRRRRRR GGGGGGGG BBBBBBBB MMM   MMM  II  RRRRRRRR
//  RR     RR GG       BB     BB MMMMMMMM  II  RR     RR
//  RRRRRRRR GG   GGG BBBBBBBB MMMMMMMM  II  RRRRRRRR
//  RRRRRRR  GG   GGG BBBBBBBB MM MM MM  II  RRRRRRR
//  RR RR    GG    GG BB     BB MM     MM  II  RR  RR
//  RR   RR  GGGGGGGG BBBBBBBB MM     MM  II  RR   RR
//  RR    RR  GGGGGGG BBBBBBB  MM     MM IIII RR    RR

// rgbmir.cpp - MAIN C++ SOURCE CODE FOR MIRAGE SIMULATOR

/*****************************************************************************/
/*                                                                           */
/*                                Include Files                              */
/*                                                                           */
/*****************************************************************************/
// Resource File Includes
#include "rgbmir.h"
#include "rgbmir.rc"

/*****************************************************************************/
/*                                                                           */
/*  How the windowing system works for this application:                     */
/*                                                                           */
/*      Main Window  (MW):    TDecoratedMDIFrame                             */
/*      Main Client  (MC):    TMDIClient                                     */
/*      Child Window (CW):    TMDIChildSB                                    */
/*      View Window  (VW):    TDrawView                                      */
/*                                                                           */
/*    |-------------------------------------------------------|              */
/*    |                                                       |  <- MW       */
/*    |                                                       |              */
/*    |       |-------------------------------|               |  <------ MC  */
/*    |       |                               |               |              */
/*    |       |                               |  <----------------- CW       */
/*    |       |                               |               |              */
/*    |       |                               |  <------------------- VW     */
/*    |       |                               |               |              */
/*    |       |                               |               |              */
/*    |       |                               |               |              */
/*    |       |-------------------------------|               |              */
/*    |                                                       |              */
/*    |-------------------------------------------------------|              */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                              TMDIChild Class                              */
/*                                                                           */
/*****************************************************************************/
DEFINE_RESPONSE_TABLE1(TMDIChildSB, TMDIChild)
    EV_WM_HSCROLL,
    EV_WM_VSCROLL,
    EV_WM_SIZE,
    EV_MESSAGE(EV_CHANGEIMAGESIZE, EvChangeImageSize),
END_RESPONSE_TABLE;

TMDIChildSB::TMDIChildSB(TMDIClient &parent, const char far *title,
                         TWindow *clientWnd, BOOL shrinkToClient, TModule* module)
    : TMDIChild(parent,title,clientWnd,shrinkToClient,module)
{
    // Adjust the size of the child window to fit the loaded bitmap
    this->Attr.W = clientWnd->Attr.W + 8;
    this->Attr.H = clientWnd->Attr.H + 27;

    // Set the Cursor inside the view Window to Cross-Hairs
    clientWnd->SetCursor(0,IDC_CROSS);

    // Set the background colour of this window
    this->SetBkgndColor(GetSysColor(COLOR_SCROLLBAR));
```

```
    // Initialize scrollers
    hScroller = 0;
    vScroller = 0;

    // Initialize width and height
    width = clientWnd->Attr.W;
    height = clientWnd->Attr.H;
}

void TMDIChildSB::addScrollBars(BOOL horiz, BOOL vert, TSize& size)
{
    // Pointer to the window that actually hold the image
    TWindow* clientWnd = this->GetFirstChild();

    // Size for each ScrollBar
    TSize hSize(size.cx,size.cy);
    TSize vSize(size.cx,size.cy);

    // If both are set size must be adjusted so that both scrollbar don't overlap
    if (horiz && vert) {
        hSize.cx = hSize.cx - SCROLLWIDTH;
        vSize.cy = vSize.cy - SCROLLHEIGHT;
    }

    // Initialize Horizontal Scroll Bar
    if (horiz) {
        if (hScroller != 0) delete hScroller;
        hScroller = new TScrollBar(this,0,0,0,hSize.cx,hSize.cy,TRUE,0);
        hScroller->Attr.Style = hScroller->Attr.Style | SBS_BOTTOMALIGN;

        // Create and Show window
        hScroller->Create();
        hScroller->Show(SW_SHOWNORMAL);

        // Set Options
        hScroller->SetRange(0,(width-clientWnd->Attr.W));
        hScroller->LineMagnitude = 1;
        hScroller->PageMagnitude = 10;
    }

    // Initialize Vertical Scroll Bar
    if (vert) {
        if (vScroller != 0) delete vScroller;
        vScroller = new TScrollBar(this,0,0,0,vSize.cx,vSize.cy,FALSE,0);
        vScroller->Attr.Style = vScroller->Attr.Style | SBS_RIGHTALIGN;

        // Create and Show window
        vScroller->Create();
        vScroller->Show(SW_SHOWNORMAL);

        // Set Options
        vScroller->SetRange(0,(height-clientWnd->Attr.H));
        vScroller->LineMagnitude = 1;
        vScroller->PageMagnitude = 10;
    }
}

void TMDIChildSB::deleteScrollBars(BOOL horiz, BOOL vert)
{
    if (horiz && (hScroller != 0)) {
        delete hScroller;
        hScroller = 0;
    }
    if (vert && (vScroller != 0)) {
        delete vScroller;
        vScroller = 0;
    }
}

LRESULT     TMDIChildSB::EvChangeImageSize(WPARAM wParam, LPARAM lParam) {
    width = wParam;
    height = lParam;
}
```

```
void TMDIChildSB::EvHScroll(UINT scrollCode, UINT thumbPos, HWND)
{
    // Pointer to the window that actually hold the image
    TWindow* clientWnd = this->GetFirstChild();

    // Get Current rectangle
    TRect tempRect(clientWnd->Attr.X, clientWnd->Attr.Y,
                               clientWnd->Attr.W+clientWnd->Attr.X,
                               clientWnd->Attr.H+clientWnd->Attr.Y);

    // Get range of scroll bar
    int min, max;
    hScroller->GetRange(min,max);

    switch (scrollCode) {
        case SB_BOTTOM:
            tempRect.left = -1*max;
            hScroller->SBBottom();
            break;
        case SB_TOP:
            tempRect.left = -1*min;
            hScroller->SBTop();
            break;
        case SB_THUMBPOSITION:
            tempRect.left = -1*thumbPos;
            hScroller->SBThumbPosition(thumbPos);
            break;
        case SB_LINEUP:
            tempRect.left += hScroller->LineMagnitude;
            if (tempRect.left > (-1*min)) tempRect.left = -1*min;
            hScroller->SBLineUp();
            break;
        case SB_LINEDOWN:
            tempRect.left -= hScroller->LineMagnitude;
            if (tempRect.left < (-1*max)) tempRect.left = -1*max;
            hScroller->SBLineDown();
            break;
        case SB_PAGEUP:
            tempRect.left += hScroller->PageMagnitude;
            if (tempRect.left > (-1*min)) tempRect.left = -1*min;
            hScroller->SBPageUp();
            break;
        case SB_PAGEDOWN:
            tempRect.left -= hScroller->PageMagnitude;
            if (tempRect.left < (-1*max)) tempRect.left = -1*max;
            hScroller->SBPageDown();
            break;
    }

    // Change client Window's position
    clientWnd->MoveWindow(tempRect,TRUE);

    // Pump messages
    (this->GetApplication())->PumpWaitingMessages();

    // Perform Default Window Processing
    DefaultProcessing();
}

void TMDIChildSB::EvVScroll(UINT scrollCode, UINT thumbPos, HWND)
{
    // Pointer to the window that actually hold the image
    TWindow* clientWnd = this->GetFirstChild();

    // Get Current rectangle
    TRect tempRect(clientWnd->Attr.X, clientWnd->Attr.Y,
                               clientWnd->Attr.W+clientWnd->Attr.X,
                               clientWnd->Attr.H+clientWnd->Attr.Y);

    // Get range of scroll bar
    int min, max;
    vScroller->GetRange(min,max);

    switch (scrollCode) {
```

```
        case SB_BOTTOM:
            tempRect.top = -1*max;
            vScroller->SBBottom();
            break;
        case SB_TOP:
            tempRect.top = -1*min;
            vScroller->SBTop();
            break;
        case SB_THUMBPOSITION:
            tempRect.top = -1*thumbPos;
            vScroller->SBThumbPosition(thumbPos);
            break;
        case SB_LINEUP:
            tempRect.top += vScroller->LineMagnitude;
            if (tempRect.top > (-1*min)) tempRect.top = -1*min;
            vScroller->SBLineUp();
            break;
        case SB_LINEDOWN:
            tempRect.top -= vScroller->LineMagnitude;
            if (tempRect.top < (-1*max)) tempRect.top = -1*max;
            vScroller->SBLineDown();
            break;
        case SB_PAGEUP:
            tempRect.top += vScroller->PageMagnitude;
            if (tempRect.top > (-1*min)) tempRect.top = -1*min;
            vScroller->SBPageUp();
            break;
        case SB_PAGEDOWN:
            tempRect.top -= vScroller->PageMagnitude;
            if (tempRect.top < (-1*max)) tempRect.top = -1*max;
            vScroller->SBPageDown();
            break;
    }

    // Change client Window's position
    clientWnd->MoveWindow(tempRect,TRUE);

    // Pump messages
    (this->GetApplication())->PumpWaitingMessages();

    // Perform Default Window Processing
    DefaultProcessing();
}

void TMDIChildSB::EvSize(UINT sizeType, TSize& size)
{
    // Pointer to the window that actually hold the image
    TWindow* clientWnd = this->GetFirstChild();

    // Repaint the Window Since client window could have been resized
    clientWnd->Invalidate(TRUE);

    switch (sizeType) {
        case SIZE_MINIMIZED:
            deleteScrollBars(TRUE,TRUE);
            break;
        default:  // Includes SIZE_RESTORED
            if (size.cx < width) {
                if ((size.cy - SCROLLHEIGHT) < height) {
                    TRect tempRect(0,0,size.cx-SCROLLWIDTH,size.cy-SCROLLHEIGHT);
                    clientWnd->MoveWindow(tempRect,TRUE);
                    addScrollBars(TRUE,TRUE,size);
                }
                else {
                    unsigned heightAdj = (size.cy-height-SCROLLHEIGHT)/2;
                    TRect tempRect(0,heightAdj,size.cx,height+heightAdj);
                    clientWnd->MoveWindow(tempRect,TRUE);
                    addScrollBars(TRUE,FALSE,size);
                    deleteScrollBars(FALSE,TRUE);
                }
            }
            else if (size.cy < height) {
                if ((size.cx - SCROLLWIDTH) < width) {
                    TRect tempRect(0,0,size.cx-SCROLLWIDTH,size.cy-SCROLLHEIGHT);
```

```
                clientWnd->MoveWindow(tempRect,TRUE);
                addScrollBars(TRUE,TRUE,size);
            }
            else {
                unsigned widthAdj = (size.cx-width-SCROLLWIDTH)/2;
                TRect tempRect(widthAdj,0,width+widthAdj,size.cy);
                clientWnd->MoveWindow(tempRect,TRUE);
                addScrollBars(FALSE,TRUE,size);
                deleteScrollBars(TRUE,FALSE);
            }
        }
        else {
            unsigned widthAdj = (size.cx-width)/2;
            unsigned heightAdj = (size.cy-height)/2;
            TRect tempRect(widthAdj,heightAdj,width+widthAdj,height+heightAdj);
            clientWnd->MoveWindow(tempRect,TRUE);
            deleteScrollBars(TRUE,TRUE);
        }
        break;
    }

    // Perform Default Window Processing
    DefaultProcessing();
}
/*****************************************************************************/
/************************** End TMDIChild Class ******************************/
/*****************************************************************************/


/*****************************************************************************/
/*                                                                           */
/*                              TViewFrame Class                             */
/*                                                                           */
/*****************************************************************************/
DEFINE_RESPONSE_TABLE1(TViewFrame, TDecoratedMDIFrame)
    EV_WM_ENTERIDLE,
    EV_WM_MENUSELECT,
END_RESPONSE_TABLE;

TViewFrame::TViewFrame(const char far* title,
                                        TResId                  menuResId,
                                        TMDIClient&     clientWnd,
                                        BOOL            trackMenuSelection,
                                        TModule*        module)
    : TDecoratedMDIFrame(title,menuResId,clientWnd,FALSE,module)
{
    TrackMenuSelection = trackMenuSelection;
    MenuItemId = 0;
}

void TViewFrame::EvMenuSelect(UINT menuItemId, UINT flags, HMENU hMenu)
{
    if (TrackMenuSelection)
        if (flags == 0xFFFF && hMenu == 0) {  // menu closing
            TStatusBar*  statusBar = (TStatusBar*)ChildWithId(IDW_STATUSBAR);
            CHECK(statusBar);

            statusBar->SetText(0);
            MenuItemId = 0;           // restore status bar to normal look

        } else if (flags & (MF_SEPARATOR | MF_POPUP | MF_MENUBREAK | MF_MENUBARBREAK)
            || (menuItemId >= IDW_FIRSTMDICHILD && menuItemId < IDW_FIRSTMDICHILD+9)) {
            MenuItemId = BLANK_HELP;  // display an empty help message
                                                    // could also restore bar at this
point too

        } else {
            MenuItemId = menuItemId;  // display a help message with this string Id
        }
}

void TViewFrame::EvEnterIdle(UINT source, HWND /*hWndDlg*/)
{
    if (source == MSGF_MENU && MenuItemId) {
```

```
        char          buf[128];
        TStatusBar*  statusBar = (TStatusBar*)ChildWithId(IDW_STATUSBAR);
        CHECK(statusBar);

        if (MenuItemId != BLANK_HELP) {
            int numBytes = GetModule()->LoadString(MenuItemId, buf, sizeof(buf));
            if (numBytes == 0 && MergeModule != 0)
                    numBytes = MergeModule->LoadString(MenuItemId, buf, sizeof(buf));

            WARNX(OwlWin, numBytes == 0, 0,
                    "TDecoratedFrame::EvEnterIdle LoadString("
                    << *GetModule() << "," << MenuItemId << ") Failed");
        } else
            *buf = 0;
        statusBar->SetText(buf);
        MenuItemId = 0;             // Don't repaint on subsequent EvEnterIdle's
    }
}
/*****************************************************************************/
/************************** End TViewFrame Class *****************************/
/*****************************************************************************/


/*****************************************************************************/
/*                                                                           */
/*                              TViewApp Class                               */
/*                                                                           */
/*****************************************************************************/
DEFINE_RESPONSE_TABLE1(TViewApp, TApplication)
    EV_OWLVIEW(dnCreate, EvNewView),
    EV_OWLVIEW(dnClose,  EvCloseView),
    EV_WM_DROPFILES,
    EV_COMMAND(CM_ABOUT, CmAbout),
    EV_MESSAGE(EV_PIXELINFO, EvPixelInfo),
    EV_MESSAGE(EV_LOADMSG, EvLoadMSG),
    EV_MESSAGE(EV_SAVEMSG, EvSaveMSG),
    EV_MESSAGE(EV_PREPARING, EvPreparing),
    EV_MESSAGE(EV_SIMULATING, EvSimulating),
    EV_MESSAGE(EV_BOTPTMSG, EvBotPtMSG),
    EV_MESSAGE(EV_TOPPTMSG, EvTopPtMSG),
    EV_MESSAGE(EV_TCMSG, EvTCMSG),
END_RESPONSE_TABLE;

// Makes frame open with no documents
void TViewApp::InitInstance()
{
    // constructs main window object, then creates main window
    TApplication::InitInstance();
    // Enables acceptance of files that are dropped into the window
    GetMainWindow()->DragAcceptFiles(TRUE);
}

// Setup the Main Window of the Application
void TViewApp::InitMainWindow()
{
    // Construct the decorated frame window
    TViewFrame* frame = new TViewFrame("Colour Mirage Simulator",0,
                                                                    *(Client    =    new
TMDIClient),TRUE);

    // Construct a status bar
    sb = new TStatusBar(frame, TGadget::Recessed);//, TStatusBar::Overtype |

    //TStatusBar::NumLock |

    //TStatusBar::CapsLock);

    // Construct the updatable Gadgets
    tPtGadget = new TTextGadget(0,TGadget::Recessed,TTextGadget::Left,14,"Top:  0,0  (0 m)");
    bPtGadget = new TTextGadget(1,TGadget::Recessed,TTextGadget::Left,14,"Bot:  0,0  (0 m)");
    pixelGadget = new TTextGadget(2,TGadget::Recessed,TTextGadget::Center,8,"0,0");
    valueGadget = new TTextGadget(3,TGadget::Recessed,TTextGadget::Center,4,"0");

    // Insert the Gadgets into the status bar
```

```
    sb->Insert(*tPtGadget);
    sb->Insert(*bPtGadget);
    sb->Insert(*pixelGadget);
    sb->Insert(*valueGadget);

    // Construct a control bar
    TControlBar *cb = new TControlBar(frame);
    cb->Insert(*new TButtonGadget(CM_FILEOPEN, CM_FILEOPEN, TButtonGadget::Command));
    cb->Insert(*new TButtonGadget(CM_FILESAVEAS, CM_FILESAVEAS, TButtonGadget::Command));
    cb->Insert(*new TSeparatorGadget(20));
    cb->Insert(*new TButtonGadget(CM_LOADTC, CM_LOADTC, TButtonGadget::Command));
    cb->Insert(*new TButtonGadget(CM_SIMULATE, CM_SIMULATE, TButtonGadget::Command));
    cb->SetHintMode(TGadgetWindow::EnterHints);

    // Insert the status bar and control bar into the frame
    frame->Insert(*sb, TDecoratedFrame::Bottom);
    frame->Insert(*cb, TDecoratedFrame::Top);

    // Set the main window and its menu
    SetMainWindow(frame);
    GetMainWindow()->SetMenuDescr(TMenuDescr("COMMANDS",1,0,0,0,1,1));

    // Install the document manager
    SetDocManager(new TDocManager(dmMDI));
}

// Setup a child window for each new view opened
void TViewApp::EvNewView(TView& view)
{
    // Get Window Associated with the newly created view
    TWindow *viewWindow = view.GetWindow();

    // Construct the new Child window around the view
    TMDIChildSB* child = new TMDIChildSB(*Client, 0, viewWindow);
    if (view.GetViewMenu())
     child->SetMenuDescr(*view.GetViewMenu());

    // Set the Background colour to the child window
    viewWindow->SetBkgndColor(GetSysColor(COLOR_SCROLLBAR));

    // Create the Child window
    child->Create();
}

// This operation is performed by lower levels
void TViewApp::EvCloseView(TView& /*view*/)
{  // nothing needs to be done here for MDI
}

// Allows more than one file to be dropped into the application
void TViewApp::EvDropFiles(TDropInfo dropInfo)
{
    int fileCount = dropInfo.DragQueryFileCount();
    for (int index = 0; index < fileCount; index++) {
     int fileLength = dropInfo.DragQueryFileNameLen(index)+1;
     char* filePath = new char [fileLength];
     dropInfo.DragQueryFile(index, filePath, fileLength);
     TDocTemplate* tpl = GetDocManager()->MatchTemplate(filePath);
     if (tpl)
        tpl->CreateDoc(filePath);
     delete filePath;
    }
    dropInfo.DragFinish();
}

// Display Dialog Box "About"
void TViewApp::CmAbout()
{
    TDialog(GetMainWindow(), IDD_ABOUT).Execute();
}

// Function to display the location and pixel value of the mouse pointer
LRESULT TViewApp::EvPixelInfo(WPARAM wParam, LPARAM lParam)
{
```

```
    unsigned int x = lParam;
    unsigned int y = lParam >> 16;
    unsigned int value = wParam;

    // Set the pixel location gadget
    char temp[20];
    sprintf(temp,"%d,%d",x,y);
    pixelGadget->SetText(temp);

    // Set the pixel value gadget
    sprintf(temp,"%d",value);
    valueGadget->SetText(temp);

    return TRUE;
}

LRESULT TViewApp::EvLoadMSG(WPARAM wParam, LPARAM)
{
    // Clear the Hint Text
    sb->SetHintText(0);

    // Set-up and display Percentage Loaded Message
    char temp[30];

    // If valid value is received, set message otherwise clear it
    if (wParam <= 100) {
        sprintf(temp,"Loading Image:  %d%%",wParam);
    }
    else {
        sprintf(temp,"Loading Image:  Complete!");
    }
    sb->SetText(temp);

    return TRUE;
}

LRESULT TViewApp::EvSaveMSG(WPARAM wParam, LPARAM)
{
    // Clear the Hint Text
    sb->SetHintText(0);

    // Set-up and display Percentage Loaded Message
    char temp[30];

    // If valid value is received, set message otherwise clear it
    if (wParam <= 100) {
        sprintf(temp,"Saving Image:  %d%%",wParam);
    }
    else {
        sprintf(temp,"Saving Image:  Complete!");
    }
    sb->SetText(temp);

    return TRUE;
}

LRESULT TViewApp::EvPreparing(WPARAM wParam, LPARAM)
{
    // Clear the Hint Text
    sb->SetHintText(0);

    // Set-up and display Percentage Loaded Message
    char temp[30];

    // If valid value is received, set message otherwise clear it
    if (wParam <= 100) {
        sprintf(temp,"Preparing Simulation...");
        //:  %d%%",wParam);
    }
    else {
        sprintf(temp,"Preparation Complete!");
    }
    sb->SetText(temp);
```

```
    return TRUE;
}

LRESULT TViewApp::EvSimulating(WPARAM wParam, LPARAM)
{
    // Clear the Hint Text
    sb->SetHintText(0);

    // Set-up and display Percentage Loaded Message
    char temp[30];

    // If valid value is received, set message otherwise clear it
    if (wParam <= 100) {
        sprintf(temp,"Simulating Mirage:  %d%%",wParam);
    }
    else {
        sprintf(temp,"Simulating Mirage:  Complete!");
    }
    sb->SetText(temp);

    return TRUE;
}

// Function to display the current bottom point
LRESULT TViewApp::EvBotPtMSG(WPARAM wParam, LPARAM lParam)
{
    unsigned int x = wParam;
    unsigned int y = lParam >> 16;
    int elevation = lParam;

    // Set the pixel location gadget
    char temp[40];
    sprintf(temp,"Bot:  %d,%d (%d m)",x,y,elevation);
    bPtGadget->SetText(temp);

    return TRUE;
}

// Function to display the current bottom point
LRESULT TViewApp::EvTopPtMSG(WPARAM wParam, LPARAM lParam)
{
    unsigned int x = wParam;
    unsigned int y = lParam >> 16;
    int elevation = lParam;

    // Set the pixel location gadget
    char temp[40];
    sprintf(temp,"Top:  %d,%d (%d m)",x,y,elevation);
    tPtGadget->SetText(temp);

    return TRUE;
}

LRESULT TViewApp::EvTCMSG(WPARAM wParam, LPARAM)
{
    char temp[40];

    switch (wParam) {
        case 0:
            strcpy(temp,"Loading Transfer Characteristic...");
            break;
        case 1:
            strcpy(temp,"Transfer Characteristic Loaded!");
            break;
        default:
            strcpy(temp,"Transfer Characteristic was not Loaded!");
            break;
    }

    sb->SetText(temp);

    return TRUE;
}
/*****************************************************************************/
```

```
/*************************** End TViewApp Class ***************************/
/**************************************************************************/

// Main Program for a OWL Application
int OwlMain(int /*argc*/, char* /*argv*/ [])
{
    //Execute Application
    return TViewApp().Run();
}
```

```
//  RRRRRRR   GGGGGGG BBBBBBB  MM     MM IIII RRRRRRR  TTTTTTTT  CCCCCCC
//  RRRRRRRR GGGGGGGG BBBBBBBB MMM   MMM  II  RRRRRRRR TTTTTTTT CCCCCCCC
//  RR     RR GG        BB     BB MMMMMMMM  II  RR     RR     TT    CC
//  RRRRRRRR GG   GGG BBBBBBBB MMMMMMMM  II  RRRRRRRR    TT    CC
//  RRRRRRR  GG   GGG BBBBBBBB MM MM MM  II  RRRRRRR     TT    CC
//  RR RR    GG    GG BB     BB MM     MM  II  RR  RR     TT    CC
//  RR   RR  GGGGGGGG BBBBBBBB MM     MM  II  RR   RR    TT    CCCCCCCC
//  RR    RR  GGGGGGG BBBBBBB  MM     MM IIII RR    RR   TT     CCCCCCC

// rgbmirtc.h - HEADER FILE FOR TRANSFER CHARACTERISTIC MODULE

/*****************************************************************************/
/*                                                                           */
/*                                Include Files                              */
/*                                                                           */
/*****************************************************************************/

// OWL Includes
#include <owl\filedoc.h>

// C and C++ Includes
#include <classlib\arrays.h>
#include <iostream.h>

/*****************************************************************************/
/*                                                                           */
/*                    Transfer Characteristic Points Class                   */
/*                                                                           */
/*****************************************************************************/
class TCPoint {
    public:
        // Constructor
        TCPoint() {apparent = 0.0; actual = 0.0;}
        TCPoint(float apparent, float actual) {}

        // Functions to Access private variables
        float getApparent() { return apparent; }
        float getActual()  { return actual; }

        // Overloaded operators for use with sorted container
        int     operator<(const TCPoint& pt) const {
            return apparent < pt.apparent ? 1 : 0;
        }
        int operator==(const TCPoint& pt) const {
            return apparent == pt.apparent ? 1 : 0;
        }
        int operator!=(const TCPoint& pt) const {
            return apparent != pt.apparent ? 1 : 0;
        }

        // Overloaded operators to load and save points
        friend ipstream& operator >>(ipstream& is, TCPoint& pt) {
            is >> pt.apparent;
            is >> pt.actual;
            return is;
        }
        friend opstream& operator <<(opstream& os, const TCPoint& pt) {
            os << pt.apparent;
            os << ' ';
            os << pt.actual;
            return os;
        }
        friend istream& operator >>(istream& is, TCPoint& pt) {
            is >> pt.apparent;
            is >> pt.actual;
            return is;
        }
        friend ostream& operator <<(ostream& os, const TCPoint& pt) {
            os << pt.apparent;
            os << ' ';
            os << pt.actual;
            return os;
        }
```

```
    private:
        float apparent;
        float actual;
};
/*****************************************************************************/
/****************** End Transfer Characteristic Points Class *****************/
/*****************************************************************************/

// Create a type to handle an array of transfer characteristic points
typedef TSArray<TCPoint> TCPoints;
typedef TSArrayIterator<TCPoint> TCPointsIterator;


/*****************************************************************************/
/*                                                                           */
/*                        Transfer Characteristic Class                      */
/*                                                                           */
/*****************************************************************************/
class TC : public TCPoints {
    public:
        // Constructor
        TC() : TCPoints(10,0,1) {}

        // Destructor
        ~TC() {TC::Flush();}

        // Define the == operator for use with any container class
        BOOL operator ==(const TC& other) const {return &other == this;}

        // Functions to load and save transfer characteristics
        friend ostream& operator <<(ostream& os, const TC& tc);
        friend istream& operator >>(istream& is, TC& tc) {
            while (!is.eof()) {
                TCPoint newPoint;
                is >> newPoint;
                while (!is.eof() && (is.peek() <= ' ')) is.get();    //Eats Whitespace
                tc.Add(newPoint);
            }
            return is;
        }

        // Functions to load and save transfer characteristics
        int write(TOutStream& os);
        int read(TInStream& is);

        // Functions to extract information from transfer characteristic
        float actualHeight(const float apparentHeight);
};
/*****************************************************************************/
/******************** End Transfer Characteristic Class **********************/
/*****************************************************************************/
```

```
//  RRRRRRR   GGGGGGG BBBBBBB  MM     MM IIII RRRRRRR  TTTTTTTT  CCCCCCC
//  RRRRRRRR GGGGGGGG BBBBBBBB MMM   MMM  II  RRRRRRRR TTTTTTTT CCCCCCCC
//  RR     RR GG       BB     BB MMMMMMMM  II  RR     RR    TT    CC
//  RRRRRRRR GG   GGG BBBBBBBB MMMMMMMM  II  RRRRRRRR    TT    CC
//  RRRRRRR  GG   GGG BBBBBBBB MM MM MM  II  RRRRRRR     TT    CC
//  RR RR    GG    GG BB     BB MM     MM  II  RR  RR     TT    CC
//  RR   RR  GGGGGGGG BBBBBBBB MM     MM  II  RR   RR    TT    CCCCCCCC
//  RR    RR  GGGGGG BBBBBBB  MM     MM IIII RR    RR    TT     CCCCCC

// rgbmirtc.cpp - C++ SOURCE CODE FOR TRANSFER CHARACTERISTIC MODULE

/*****************************************************************************/
/*                                                                           */
/*                                Include Files                              */
/*                                                                           */
/*****************************************************************************/
#include "rgbmirtc.h"


/*****************************************************************************/
/*                                                                           */
/*                       Transfer Characteristic Class                       */
/*                                                                           */
/*****************************************************************************/
// Define output function
ostream& operator <<(ostream& os, const TC& tc) {
    // Write each point and a carriage return to the file
    for (int count=0; count<tc.GetItemsInContainer(); count++) {
        os << tc[count];
        os << '\n';
    }

    // Return the output stream
    return os;
}

// Function to write a transfer characeristic to a file
int TC::write(TOutStream& os)
{
    // If the file is not open for writing return with failure
    if (!os) return 0;

    // Otherwise write data to file
    TC& tc = *this;

    // Write transfer characteristic to file, one point at a time
    for (int count=0; count<tc.GetItemsInContainer(); count++) {
        os << tc[count];
        os << '\n';
    }

    // Return file which was written to
    return 1;
}

int TC::read(TInStream& is)
{
    // If the file is not open for reading return with failure
    if (!is) return 0;

    // Create tc for file to be read to
    TC& tc = *this;

    // Read from file one point at a time
    while (!is.eof()) {
        TCPoint newPoint;
        is >> newPoint;
        while (!is.eof() && (is.peek() <= ' ')) is.get();  // Eat Whitespace
        tc.Add(newPoint);
    }

    // Return file which was read from
    return 1;
}
```

```
// Get the Actual height of a point of apparent elevation
float TC::actualHeight(const float apparentHeight)
{
    TC& tc = *this;
    if (apparentHeight < tc[0].getApparent()) {
        return apparentHeight;
    }
    else if (apparentHeight > tc[tc.GetItemsInContainer()-1].getApparent()) {
        return apparentHeight;
    }
    else {
        for (int count=0; tc[count].getApparent()<apparentHeight; count++);
        float apparentLower = tc[count-1].getApparent();
        float actualLower= tc[count-1].getActual();
        float apparentUpper = tc[count].getApparent();
        float actualUpper = tc[count].getActual();
        if (actualUpper == actualLower) {
            return actualUpper;
        }
        else {
            float slope = (apparentUpper - apparentLower) / (actualUpper - actualLower);
            float intercept = apparentUpper - slope * actualUpper;
            return ((apparentHeight - intercept) / slope);
        }
    }
}
/*****************************************************************************/
/******************** End Transfer Characteristic Class **********************/
/*****************************************************************************/
```

```
//  RRRRRRR   GGGGGGG BBBBBBB  MM     MM IIII RRRRRRR  BBBBBBB  MM     MM
//  RRRRRRRR GGGGGGGG BBBBBBBB MMM   MMM  II  RRRRRRRR BBBBBBBB MMM   MMM
//  RR     RR GG       BB     BB MMMMMMMM  II  RR     RR BB     BB MMMMMMMM
//  RRRRRRRR GG   GGG BBBBBBBB MMMMMMMM  II  RRRRRRRR BBBBBBBB MMMMMMMM
//  RRRRRRR  GG   GGG BBBBBBBB MM MM MM  II  RRRRRRR  BBBBBBBB MM MM MM
//  RR RR    GG    GG BB     BB MM     MM  II  RR  RR   BB     BB MM     MM
//  RR   RR  GGGGGGGG BBBBBBBB MM     MM  II  RR   RR  BBBBBBBB MM     MM
//  RR    RR  GGGGGG BBBBBBB  MM     MM IIII RR    RR BBBBBBB  MM     MM

// rgbmirbm.h - HEADER FILE FOR BITMAP LOADING MODULE

/*****************************************************************************/
/*                                                                           */
/*                                Include Files                              */
/*                                                                           */
/*****************************************************************************/

// OWL Includes
#include <owl\filedoc.h>

// C and C++ Includes
#include <windowsx.h>

//Global Variables
const size_t maxwrite = 30*1024;      // Define Maximum Write Block size
const size_t maxread = 30*1024;          //Read 30K at a time

//Type Definitions
typedef struct PixelStruct {
    unsigned char  red;
    unsigned char green;
    unsigned char blue;
} Pixel;


/*****************************************************************************/
/*                                                                           */
/*                              ImageData Class                              */
/*                                                                           */
/*****************************************************************************/
class ImageData {
    public:
        friend class Image;
        friend class BMPImage;

    protected:
        ImageData() : p_dib(0), count(1), hpal(0), hbm_ddb(0), bytes_per_line(0),
                      w(0),h(0) {}
        ~ImageData();

        // The following points to a BITMAPINFOHEADER followed by the image data.
        LPVOID                p_dib;              // Device independent bitmap
        HPALETTE              hpal;               // Color pallette
        HBITMAP               hbm_ddb;            // Device dependent bitmap
        unsigned short     w, h;                  // Width and height
        unsigned short bytes_per_line;
        unsigned short count;
};
/*****************************************************************************/
/************************** End ImageData Class ******************************/
/*****************************************************************************/


/*****************************************************************************/
/*                                                                           */
/*                                Image Class                                */
/*                                                                           */
/*****************************************************************************/
class Image {
    public:
        // Constructors
        Image() { imdata = new ImageData; }
        Image(HBITMAP hbm, unsigned short w, unsigned short h) {
            imdata = new ImageData;
```

```
            imdata->hbm_ddb = hbm;
            imdata->w = w;
            imdata->h = h;
        }
        Image(HDC hdc, Image *img, short x, short y,
                  unsigned short w, unsigned short h);
        Image(const Image& img);

        // Destructor
        virtual ~Image() { if (--imdata->count <= 0) delete imdata; }

        // Operators
        Image& operator=(const Image& img);

        // Copy the imdata pointer from another image
        void image_data(const Image* img);

        // Functions to load and save images
        virtual int write(TOutStream& os, TWindow *mainWindow) { return 0; }
        virtual int read(TInStream& is, TWindow *mainWindow) { return 0; }

        int write_dib(ofstream& ofs);

        // Returns pointer to the Windows Device Independent Bitmap (DIB).
        LPVOID get_dib() { return imdata->p_dib; }

        // Function to return the handle to the device dependent bitmap
        HBITMAP get_ddb() { return imdata->hbm_ddb; }

        // Information functions
        unsigned short width() {
            if (imdata->p_dib != 0)
                return ((LPBITMAPINFOHEADER)imdata->p_dib)->biWidth;
            return imdata->w;
        }
        unsigned short height() {
            if (imdata->p_dib != 0)
                return ((LPBITMAPINFOHEADER)imdata->p_dib)->biHeight;
            return imdata->h;
        }
        int image_loaded() {
            if (imdata->p_dib == 0) return 0;
            return 1;
        }
        void detach() {
            if (--imdata->count == 0) delete imdata;
            imdata = new ImageData;
        }

        // Functions to make palette and convert to DDB
        void make_palette();
        void DIBtoDDB(HDC hdc);
        void DDBtoDIB();

        //Function to display the DIB on a Windows device specified by a device context
        void show(HDC hdc, short xfrom = 0, short yfrom = 0,
                                        short xto = 0,   short yto = 0,
                                        short width = 0, short height = 0,
                                        DWORD ropcode = SRCCOPY);
        unsigned int numcolors();
        HPALETTE palette() { return imdata->hpal; }

        // Pixel Access Functions
        Pixel getPixel(unsigned int x, unsigned int y);
        void putPixel(unsigned int x, unsigned int y, Pixel pixel);

        // Add lines to a bitmap
        LPVOID addLines(unsigned long numberOfLines);

    protected:
        ImageData* imdata;
};
/*****************************************************************************/
/***************************** End Image Class *******************************/
```

```
/*****************************************************************************/


/*****************************************************************************/
/*                                                                           */
/*                                BMPImage Class                             */
/*                                                                           */
/*****************************************************************************/
class BMPImage: public Image
{
    public:
        BMPImage() {}
        ~BMPImage() {}

        int write(TOutStream& os, TWindow *mainWindow);
        int read(TInStream& is, TWindow *mainWindow);

    private:
        BITMAPFILEHEADER bmphdr;
};
/*****************************************************************************/
/************************** End BMPImage Class *******************************/
/*****************************************************************************/
```

```
//  RRRRRRR   GGGGGGG BBBBBBB  MM     MM IIII RRRRRRR  BBBBBBB  MM     MM
//  RRRRRRRR GGGGGGGG BBBBBBBB MMM   MMM  II  RRRRRRRR BBBBBBBB MMM   MMM
//  RR     RR GG       BB     BB MMMMMMMM  II  RR     RR BB     BB MMMMMMMM
//  RRRRRRRR GG   GGG BBBBBBBB MMMMMMMM  II  RRRRRRRR BBBBBBBB MMMMMMMM
//  RRRRRRR  GG   GGG BBBBBBBB MM MM MM  II  RRRRRRR  BBBBBBBB MM MM MM
//  RR RR    GG    GG BB     BB MM     MM  II  RR  RR   BB     BB MM     MM
//  RR   RR  GGGGGGGG BBBBBBBB MM     MM  II  RR   RR  BBBBBBBB MM     MM
//  RR    RR  GGGGGGG BBBBBBB  MM     MM IIII RR    RR BBBBBBB  MM     MM

// rgbmirbm.cpp - C++ SOURCE CODE FOR BITMAP LOADING MODULE


/*****************************************************************************/
/*                                                                           */
/*                                Include Files                              */
/*                                                                           */
/*****************************************************************************/
#include "rgbmirbm.h"
#include "rgbmir.rc"

/*****************************************************************************/
/*                                                                           */
/*                                ImageData Class                            */
/*                                                                           */
/*****************************************************************************/
// Image Destructor
ImageData::~ImageData()
{
    // If a DIB exists, delete it.
    if (p_dib != 0) GlobalFreePtr(p_dib);

    // If a palette exists, free it also.
    if (hpal != 0) DeletePalette(hpal);

    // If a DDB exists, destroy it.
    if (hbm_ddb != 0) DeleteBitmap(hbm_ddb);
}
/*****************************************************************************/
/*************************** End ImageData Class *****************************/
/*****************************************************************************/


/*****************************************************************************/
/*                                                                           */
/*                                Image Class                                */
/*                                                                           */
/*****************************************************************************/
// Construct an image by copying a portion of the bitmap from another image
Image::Image(HDC hdc, Image *img, short x, short y,
                        unsigned short w, unsigned short h)
{
    imdata = new ImageData;
    if (img == NULL) return;

    unsigned short iw = img->width();
    unsigned short ih = img->height();

    if (x < 0) x = 0;
    if (y < 0) y = 0;

    // If width or height is 0, adjust them
    if (w == 0) w = iw;
    if (h == 0) h = ih;

    // Make sure width and height are not too large
    if ((w+x) > iw) w = iw - x;
    if ((h+y) > ih) h = ih - y;

    // Save width and height
    imdata->w = w;
    imdata->h = h;

    //Create a new bitmap for the new image
    imdata->hbm_ddb = CreateCompatibleBitmap(hdc, w, h);
```

```
    if (imdata->hbm_ddb != 0) {
        HDC memdcn = CreateCompatibleDC(hdc);
        HDC memdco = CreateCompatibleDC(hdc);
        if ((memdcn != 0) && (memdco != 0)) {
            HBITMAP ohbm = SelectBitmap(memdco, img->get_ddb());
            HBITMAP nhbm = SelectBitmap(memdcn, imdata->hbm_ddb);
            BitBlt(memdcn,0,0,w,h,memdco,x,y,SRCCOPY);
            SelectBitmap(memdco,ohbm);
            SelectBitmap(memdcn,nhbm);
            DeleteDC(memdco);
            DeleteDC(memdcn);
        }
    }
}

// Copy Constructor
Image::Image(const Image& img)
{
    imdata = new ImageData;
    img.imdata->count++;
    if (--imdata->count <= 0) delete imdata;
    imdata = img.imdata;
}

// Image assignment operator
Image& Image::operator=(const Image& img)
{
    img.imdata->count++;
    if (--imdata->count <= 0) delete imdata;
    imdata = img.imdata;
    return *this;
}

// Copy the ImageData pointer from another image
void Image::image_data(const Image* img)
{
    img->imdata->count++;
    if (--imdata->count <= 0) delete imdata;
    imdata = img->imdata;
}

// Write out the DIB starting at the current location in a stream
// (assumed to be opened with ios::out | ios::binary)
int Image::write_dib(ofstream& ofs)
{
    // If there is no image, return without doing anything
    if (imdata->p_dib == NULL) return 0;

    // Check if file is ok
    if (!ofs) return 0;

    // Set up BMP file header
    BITMAPFILEHEADER bfhdr;

    bfhdr.bfType = ('M' << 8) | 'B';
    bfhdr.bfReserved1 = 0;
    bfhdr.bfReserved2 = 0;
    bfhdr.bfOffBits = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) +
                                        numcolors() * sizeof(RGBQUAD);
    bfhdr.bfSize = (long) height() * (long) imdata->bytes_per_line + bfhdr.bfOffBits;

    // Wrtie the file header to the file
    ofs.write((unsigned char*)&bfhdr, sizeof(BITMAPFILEHEADER));

    //Save the file in big chunks.

    //Allocate a large buffer to be used when transferring data to the file
    unsigned char *wbuf = new unsigned char[maxwrite];
    if (wbuf == NULL) return 0;

    unsigned char huge *data = (unsigned char huge*)imdata->p_dib;
    unsigned int chunksize;
    long bmpsize = bfhdr.bfSize - sizeof(BITMAPFILEHEADER);
```

```
    unsigned int i;
    while(bmpsize > 0) {
        if (bmpsize > maxwrite) chunksize = maxwrite;
        else chunksize = bmpsize;
        // Copy image from DIB to buffer
        for (i=0; i<chunksize; i++) wbuf[i] = data[i];
        ofs.write(wbuf,chunksize);
        bmpsize -= chunksize;
        data += chunksize;
    }
    delete wbuf;
    return 1;
}

// Create a colour palette using information in the DIB
void Image::make_palette()
{
    // Set up a pointer to the DIB
    LPBITMAPINFOHEADER p_bminfo = (LPBITMAPINFOHEADER)(imdata->p_dib);
    if (p_bminfo == 0) return;

    // Free any existing palette
    if(imdata->hpal != 0) DeletePalette(imdata->hpal);

    // Set up the palette, if needed
    if (numcolors() > 0) {
        LPLOGPALETTE p_pal = (LPLOGPALETTE) GlobalAllocPtr(GHND, sizeof(LOGPALETTE)
sizeof(PALETTEENTRY));                                                    + numcolors() *
        if (p_pal) {
            p_pal->palVersion = 0x0300;
            p_pal->palNumEntries = numcolors();
            // Set up palette entries from DIB
            LPBITMAPINFO p_bi = (LPBITMAPINFO)p_bminfo;
            int i;
            for (i=0; i<numcolors(); i++) {
                p_pal->palPalEntry[i].peRed = p_bi->bmiColors[i].rgbRed;
                p_pal->palPalEntry[i].peGreen = p_bi->bmiColors[i].rgbGreen;
                p_pal->palPalEntry[i].peBlue = p_bi->bmiColors[i].rgbBlue;
                p_pal->palPalEntry[i].peFlags = 0;
            }
            imdata->hpal = CreatePalette(p_pal);
            GlobalFreePtr(p_pal);
        }
    }
}

// Create a device dependent bitmap from the DIB
void Image::DIBtoDDB(HDC hdc)
{
    // Set up a pointer to the DIB
    LPBITMAPINFOHEADER p_bminfo = (LPBITMAPINFOHEADER)(imdata->p_dib);
    if (p_bminfo == 0) return;

    // If a DDB exists, destroy it first.
    if (imdata->hbm_ddb != 0) DeleteBitmap(imdata->hbm_ddb);

    // Build the device-dependent bitmap.
    // Set up pointer to the image data (skip over BITMAPINFOHEADER and palette).
    LPSTR p_image = (LPSTR)p_bminfo + sizeof(BITMAPINFOHEADER)
sizeof(RGBQUAD);                                          +      numcolors()      *

    // Realize palette, if there is one. Note that this does not do anything
    // on the standard 16-color VGA driver because that driver does not allow
    // changing the palette, but the new palette should work on SVGA displays.
    HPALETTE hpalold = NULL;
    if (imdata->hpal) {
        hpalold = SelectPalette(hdc, imdata->hpal, FALSE);
        RealizePalette(hdc);
    }

    // Convert the DIB into a DDB (device dependent bitmap) and block
    // transfer (blt) it to the device context.
```

```
    //HBITMAP hbm_old;
    imdata->hbm_ddb = CreateDIBitmap(hdc,p_bminfo,CBM_INIT,p_image,
(LPBITMAPINFO)p_bminfo,DIB_RGB_COLORS);

    // Don't need the palette once the bitmap is converted to DDB format.
    if (hpalold) SelectPalette(hdc, hpalold, FALSE);
}

// Create a device independent bitmap from the DDB
void Image::DDBtoDIB()
{
    // Do nothing if the DDB does not exist or if the DIB exists
    if (imdata->hbm_ddb == 0) return;
    if (imdata->p_dib != NULL) return;

    BITMAP bm;
    GetObject(imdata->hbm_ddb, sizeof(bm), (LPSTR)&bm);

    // Set up a BITMAPINFOHEADER for the DIB
    BITMAPINFOHEADER bh;
    bh.biSize               = sizeof(BITMAPINFOHEADER);
    bh.biWidth                  = bm.bmWidth;
    bh.biHeight             = bm.bmHeight;
    bh.biPlanes                 = 1;
    bh.biBitCount           = (WORD)(bm.bmPlanes * bm.bmBitsPixel);
    bh.biCompression    = BI_RGB;

    imdata->w = bm.bmWidth;
    imdata->h = bm.bmHeight;

    // Compute bytes per line, rounding up to align at a 4-byte boundary
    imdata->bytes_per_line = ((long)bh.biWidth*(long)bh.biBitCount+31L)/32*4;

    bh.biSizeImage = (long)bh.biHeight*(long)imdata->bytes_per_line;

    bh.biXPelsPerMeter = 0;
    bh.biYPelsPerMeter = 0;

    // Determine number of colors in the palette
    short ncolors = 0;

    switch (bh.biBitCount) {
        case 1:
            ncolors = 2;
            break;
        case 4:
            ncolors = 16;
            break;
        case 8:
            ncolors = 256;
            break;
        default:
            ncolors = 0;
    }

    bh.biClrUsed = ncolors;
    bh.biClrImportant = 0;

    // Compute total size of DIB
    unsigned long dibsize = sizeof(BITMAPINFOHEADER) + ncolors * sizeof(RGBQUAD)

         + bh.biSizeImage;

    // Allocate memory for the DIB
    imdata->p_dib = GlobalAllocPtr(GHND, dibsize);
    if(imdata->p_dib == NULL) return;

    // Set up palette
    HDC hdc = GetDC(NULL);

    // Copy BIIMAPINFO structure bh into beginning of DIB.
    _fmemcpy(imdata->p_dib, &bh, (size_t)bh.biSize);
```

```
    LPSTR p_image = (LPSTR)imdata->p_dib + (WORD)bh.biSize
sizeof(RGBQUAD);                                                    +   ncolors   *

    // Call GetDIBits to get the image and fill the palette indices into a
    // BITMAPINFO structure
    GetDIBits(hdc, imdata->hbm_ddb, 0, (WORD)bh.biHeight, p_image,
                      (LPBITMAPINFO)imdata->p_dib, DIB_RGB_COLORS);

    // All done. Clean up and return.
    ReleaseDC(NULL, hdc);
}


// Display a DIB on a Windows device specified by a device context
void Image::show(HDC hdc, short xfrom, short yfrom, short xto, short yto,
                                                    short wdth, short hght, DWORD ropcode)
{
    // Set up a pointer to the DIB
    LPBITMAPINFOHEADER p_bminfo = (LPBITMAPINFOHEADER)(imdata->p_dib);
    if (p_bminfo != NULL) {
        // Set up the palette, if needed
        if ((imdata->hpal == 0) && numcolors() > 0) make_palette();
        // Convert to DDB, if necessary
        if (imdata->hbm_ddb == 0) {
            DIBtoDDB(hdc);
        }
    }

    // "Blit" the DDB to hdc
    if (imdata->hbm_ddb != 0) {
        HDC memdc = CreateCompatibleDC(hdc);
        if (memdc != 0) {
            HBITMAP hbm_old = SelectBitmap(memdc,imdata->hbm_ddb);
            // If width of height is zero, use corresponding dimension from the image.
            if (wdth == 0) wdth = width();
            if (hght == 0) hght = height();

            BitBlt(hdc, xto, yto, wdth, hght, memdc, xfrom, yfrom, ropcode);
            SelectBitmap(memdc, hbm_old);
            DeleteDC(memdc);
        }
    }
}

// Returns the number of colors used. Returns 0 if image uses 24-bit pizels
unsigned int Image::numcolors()
{
    if (imdata->p_dib == 0) return 0;
    LPBITMAPINFOHEADER p_bminfo = (LPBITMAPINFOHEADER)(imdata->p_dib);

    // If the biClrUsed field is nonzero, use that as the number of colors
    if (p_bminfo->biClrUsed != 0) return (unsigned int)p_bminfo->biClrUsed;

    // Otherwise, the number of colors depends on the bits per pixel
    switch (p_bminfo->biBitCount) {
        case 1: return 2;
        case 4: return 16;
        case 8: return 256;
        default: return 0;  // Must be 24-bit/pixel image
    }
}


// Function to return the pixel value at a specified point
Pixel Image::getPixel(unsigned int x, unsigned int y)
{
    Pixel pixel;

    unsigned long h = height();
    unsigned long w = width();

    // If the pixel selected is outside the image return dummy value
    if ((x>w)||(y>h)) {
```

```
            pixel.red = 0;
            pixel.green = 0;
            pixel.blue = 0;
            return pixel;
        }

        unsigned short colours = numcolors();

        // Line offset... offsets to the correct line in the image
        unsigned long lineOffset = (h-y-1)*imdata->bytes_per_line +
                                                    sizeof(BITMAPINFOHEADER) +
                                                    colours * sizeof(RGBQUAD);

        // Allocate pointer to the image data
        unsigned char huge *data = (unsigned char huge*)imdata->p_dib;

        // Find correct line in image
        data += lineOffset;

        // Declare variable for case 2 and 4
        unsigned bitsRemaining;
        unsigned char temp;

        // Depend on the number of colours, offset into image and get pixel value
        switch (colours) {
            case 0:
                data += 3L * x;
                pixel.blue = data[0];
                pixel.green = data[1];
                pixel.red = data[2];
                break;
            case 2:
                data += (1L*x)/8L;
                bitsRemaining = (1*x)%8;
                temp = data[0];
                temp = (temp << bitsRemaining);
                temp = (temp >> 6);
                pixel.blue = temp;
                pixel.green = temp;
                pixel.red = temp;
                break;
            case 16:
                data += (4L*x)/8L;
                bitsRemaining = (4*x)%8;
                temp = data[0];
                temp = (temp << bitsRemaining);
                temp = (temp >> 4);
                pixel.blue = temp;
                pixel.green = temp;
                pixel.red = temp;
                break;
            case 256:
                data += 1L * x;
                pixel.blue = data[0];
                pixel.green = data[0];
                pixel.red = data[0];
                break;
        }

        // Return Calculated Pixel
        return pixel;
}


// Function to return the RGB pixel values at a specified point
void Image::putPixel(unsigned int x, unsigned int y, Pixel pixel)
{
    unsigned long h = height();
    unsigned short colours = numcolors();

    // Line offset... offsets to the correct line in the image
    unsigned long lineOffset = (h-y-1)*imdata->bytes_per_line +
                                                    sizeof(BITMAPINFOHEADER) +
                                                    colours * sizeof(RGBQUAD);
```

```
        // Allocate pointer to the image data
        unsigned char huge *data = (unsigned char huge*)imdata->p_dib;

        // Find correct line in image
        data += lineOffset;

        // Declare variable for case 2 and 16
        unsigned bitsRemaining;
        unsigned char mask, temp;

        // Offset into line depending on number of colours and set pixel
        switch (colours) {
            case 0:
                data += 3L * x;
                data[0] = pixel.blue;
                data[1] = pixel.green;
                data[2] = pixel.red;
                break;
            case 2:
                data += (1L*x)/8L;
                bitsRemaining = (1*x)%8;
                mask = ~(1 << (7 - bitsRemaining));
                temp = data[0] & mask;
                if (pixel.red == 0) data[0] = temp;
                else data[0] = temp | ~mask;
                break;
            case 16:
                data += (4L*x)/8L;
                bitsRemaining = (4*x)%8;
                data[0] = (data[0]&(15<<bitsRemaining))|(pixel.red<<(4-bitsRemaining));
                break;
            case 256:
                data += x;
                data[0] = pixel.red;
                break;
        }

        // If a DDB exists, destroy it since it source dib is updated
        if (imdata->hbm_ddb != 0) {
            DeleteBitmap(imdata->hbm_ddb);
            imdata->hbm_ddb = 0;
        }
    }

    LPVOID Image::addLines(unsigned long numberOfLines)
    {
        unsigned long he = height();

        unsigned long originalSize = sizeof(BITMAPINFOHEADER) +
                                                        he*imdata->bytes_per_line +
                                                        numcolors() * sizeof(RGBQUAD);
        unsigned long sizeOfBitmap = originalSize + (numberOfLines*imdata->bytes_per_line);

        // create new device independent bitmap
        imdata->h = he + numberOfLines;
        imdata->p_dib = GlobalReAllocPtr(imdata->p_dib,sizeOfBitmap,GHND);

        // updata device independent bitmap height
        LPBITMAPINFOHEADER p_bi = (LPBITMAPINFOHEADER)imdata->p_dib;
        p_bi->biHeight = imdata->h;

        // If a DDB exists, destroy it since it source dib is updated
        if (imdata->hbm_ddb != 0) {
            DeleteBitmap(imdata->hbm_ddb);
            imdata->hbm_ddb = 0;
        }

        return imdata->p_dib;
    }
    /*****************************************************************************/
    /***************************** End Image Class *******************************/
    /*****************************************************************************/
```

```
/*****************************************************************************/
/*                                                                           */
/*                                BMPImage Class                             */
/*                                                                           */
/*****************************************************************************/
// Write a Windows .BMP image to a file (in Device Independent Bitmap format)
int BMPImage::write(TOutStream& os, TWindow *mainWindow)
{
    // If there is no image, return without doing anything
    if(imdata->p_dib == 0) return 0;

    // If stream is not open return in error
    if (!os) return 0;

    // Set up BMP file header
    bmphdr.bfType = ('M' << 8) | 'B';
    bmphdr.bfReserved1 = 0;
    bmphdr.bfReserved2 = 0;
    bmphdr.bfOffBits = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER)
numcolors() * sizeof(RGBQUAD);
    bmphdr.bfSize = (long)height()*(long)imdata->bytes_per_line+bmphdr.bfOffBits;

    // Write the file header to the file
    os.write((unsigned char*)&bmphdr, sizeof(BITMAPFILEHEADER));

    // Save file in big chunks
    // Allocate a large buffer to be used when transferring data to the file
    unsigned char *wbuf = new unsigned char[maxwrite];
    if (wbuf == NULL) return 0;

    unsigned char huge *data = (unsigned char huge*)imdata->p_dib;
    unsigned int chunksize;
    long bmpsize = bmphdr.bfSize - sizeof(BITMAPFILEHEADER);
    unsigned int i;
    unsigned long originalSize = bmpsize;

    while (bmpsize > 0) {
        if (bmpsize > maxwrite) chunksize = maxwrite;
        else chunksize = bmpsize;

        // Copy image from DIB to buffer
        for (i=0; i<chunksize; i++) wbuf[i] = data[i];
        os.write(wbuf, chunksize);
        bmpsize -= chunksize;
        data += chunksize;

        // This is a bad location for this but I'll put it here anyway
        WPARAM wParam = (originalSize-bmpsize)*100/originalSize;
        mainWindow->SendMessage(EV_SAVEMSG,wParam);
        (mainWindow->GetApplication())->PumpWaitingMessages();
    }
    delete wbuf;

    return 1;
}

// Read image information from an open file
int BMPImage::read(TInStream& is, TWindow *mainWindow)
{
    // If there is an existing image, detach the imagee data before reading new image
    if (imdata->p_dib != 0) detach();

    if (!is) {
        // Error reading file. Return 0.
        return 0;
    }

    // Get length of file
    is.seekg(0,istream::end);
    long length = is.tellg();
    is.seekg(0,istream::beg);

    // Read the file header
```

```
    is.read((unsigned char*)&bmphdr, sizeof(BITMAPFILEHEADER));

    //Check if image file format is acceptable (the type must be 'BM')
    if (bmphdr.bfType != (('M' <<8) | 'B')) return 0;

    //If file length in header is not equal to actual length then reset length
    if (bmphdr.bfSize != length) bmphdr.bfSize = length;

    // Determine size of DIB to read -- that's the file size (as specified by the
    // bfSize field of the BITMAPFILEHEADER struncture) minus the size of the
    // BITMAPFILEHEADER
    long bmpsize = bmphdr.bfSize - sizeof(BITMAPFILEHEADER);

    // Alocate space for the bitmap
    imdata->p_dib = GlobalAllocPtr(GHND, bmpsize);

    // If memory allocation fails, return 0
    if (imdata->p_dib == 0) return 0;

    // Load the file in big chunks.  We don't have to interpret because our
    // internal format is also BMP.

    // Allocate a large buffer to read from file
    unsigned char *rbuf = new unsigned char[maxread];
    if (rbuf == NULL) {
        detach();
        return 0;
    }

    unsigned char huge *data = (unsigned char huge *)imdata->p_dib;
    unsigned int chunksize;
    unsigned int i;
    unsigned long originalSize = bmpsize;

    while (bmpsize > 0) {
        if (bmpsize > maxread) chunksize = maxread;
        else chunksize = bmpsize;
        is.read(rbuf,chunksize);

        // Copy into DIB
        for (i=0; i<chunksize; i++) data[i] = rbuf[i];
        bmpsize -= chunksize;

        // This is a bad location for this but I'll put it here anyway
        WPARAM wParam = (originalSize-bmpsize)*100/originalSize;
        mainWindow->SendMessage(EV_LOADMSG,wParam);
        (mainWindow->GetApplication())->PumpWaitingMessages();

        data += chunksize;
    }
    delete rbuf;

    // Compute bytes per line, rounding up to align at a 4-byte boundary
    LPBITMAPINFOHEADER p_bi = (LPBITMAPINFOHEADER)imdata->p_dib;
    imdata->bytes_per_line = ((long)p_bi->biWidth*(long)p_bi->biBitCount+31L)/32*4;

    return 1;
}
/*****************************************************************************/
/***************************** End BMPImage Class ****************************/
/*****************************************************************************/
```

```
//  RRRRRRR   GGGGGGG BBBBBBB  MM     MM IIII RRRRRRR  DDDDDDD  VV     VV
//  RRRRRRRR GGGGGGGG BBBBBBBB MMM   MMM  II  RRRRRRRR DDDDDDDD VV     VV
//  RR     RR GG       BB     BB MMMMMMMM  II  RR     RR DD     DD VV     VV
//  RRRRRRRR GG   GGG BBBBBBBB MMMMMMMM  II  RRRRRRRR DD     DD  VV   VV
//  RRRRRRR  GG    GGG BBBBBBBB MM MM MM  II  RRRRRRR  DD     DD  VV   VV
//  RR RR    GG     GG BB     BB MM     MM  II  RR  RR   DD     DD   VVVV
//  RR   RR  GGGGGGGG BBBBBBBB MM     MM  II  RR   RR  DDDDDDDD    VV
//  RR    RR  GGGGGGG BBBBBBB  MM     MM IIII RR    RR DDDDDDD     VV

// rgbmirdv.cpp - C++ SOURCE CODE FOR WINDOWS MODULE

/*****************************************************************************/
/*                                                                           */
/*                                Include Files                              */
/*                                                                           */
/*****************************************************************************/
// OWL Includes
#include <owl\owlpch.h>
#include <owl\dc.h>
#include <owl\inputdia.h>
#include <owl\opensave.h>
#include <owl\chooseco.h>
#include <owl\gdiobjec.h>
#include <owl\docmanag.h>
#include <owl\filedoc.h>
#include <owl\listbox.h>
#include <owl\window.h>   // For TCommandEnabler

// C and C++ Includes
#include <classlib\arrays.h>
#include <stdio.h>
#include <math.h>
#include <fstream.h>
#include <windowsx.h>
#include <string.h>
#include <limits.h>

// Resource File Includes
#include "rgbmir.rc"
#include "rgbmir.h"
#include "rgbmirbm.h"
#include "rgbmirtc.h"

/*****************************************************************************/
/*                                                                           */
/*                              Global Variables                             */
/*                                                                           */
/*****************************************************************************/
// Transfer Characteristic for this Bitmap
TC tc;

// Points on Image and their Actual Elevation;
TPoint topPoint(0,0);
float topElevation = 0.0;
TPoint bottomPoint(0,0);
float bottomElevation = 0.0;

/*****************************************************************************/
/*                                                                           */
/*                            TDrawDocument Class                            */
/*                                                                           */
/*****************************************************************************/
class _DOCVIEWCLASS TDrawDocument : public TFileDocument
{
    public:
        TDrawDocument(TDocument* parent = 0) : TFileDocument(parent)
        {
            image = 0;
            mainWindow = ((this->GetDocManager()).GetApplication())->GetMainWindow();
        }
        ~TDrawDocument() { delete image; }

        // implement virtual methods of TDocument
        BOOL   Open(int mode, const char far* path=0);
```

```
        BOOL    Close();
        BOOL    IsOpen() { return image != 0; }
        BOOL    Commit(BOOL force = FALSE);
        BOOL    Revert(BOOL clear = FALSE);

        // data access functions
        Image* GetImage();

    protected:
        Image* image;
        TWindow* mainWindow;
};

// Open an image document
BOOL TDrawDocument::Open(int /*mode*/, const char far* path)
{
    if (image != 0) delete image;
    image = new BMPImage;
    if (path) SetDocPath(path);
    if (GetDocPath()) {
        TInStream* is = InStream(ofRead | ios::binary);
        if (!is) return FALSE;
        // Read data into image
        image->read(*is,mainWindow);
        delete is;
    }
    SetDirty(FALSE);
    return TRUE;
}

// Close a document containing an image
BOOL TDrawDocument::Close()
{
    delete image;
    image = 0;
    return TRUE;
}

// Save a changed image
BOOL TDrawDocument::Commit(BOOL force)
{
    if (!IsDirty() && !force) return TRUE;

    // Open output stream
    TOutStream* os = OutStream(ofWrite | ios::binary);
    if (!os) return FALSE;

    // Write Document to file
    if (image == NULL) return FALSE;
    image->write(*os,mainWindow);

    // Close output stream
    delete os;

    // Set document to clean and exit
    SetDirty(FALSE);
    return TRUE;
}

// Revert to original image if it has changed
BOOL TDrawDocument::Revert(BOOL clear)
{
    if (!TFileDocument::Revert(clear))
     return FALSE;
    return TRUE;
}

// Returns the current image to the caller function
Image* TDrawDocument::GetImage()
{
    if (IsOpen()) return image;
    else if (Open(ofRead | ofWrite)) return image;
    else return 0;
}
```

```
/*****************************************************************************/
/************************** End TDrawDocument Class **************************/
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                                TDrawView Class                            */
/*                                                                           */
/*****************************************************************************/
class _DOCVIEWCLASS TDrawView : public TWindowView
{
    public:
     TDrawView(TDrawDocument& doc, TWindow *parent = 0);
    ~TDrawView() {}

    protected:
     // Document That Controls a Bitmap
     TDrawDocument* DrawDoc;

     // Quality of Mirage Transformation
     BOOL highQuality;
     BOOL newView;

     // Message response functions
     void EvLButtonDown(UINT, TPoint&);
     void EvMButtonDown(UINT, TPoint&);
     void EvRButtonDown(UINT, TPoint&);
     void EvMouseMove(UINT, TPoint&);
     void CmHighQuality();
     void CeHighQuality(TCommandEnabler &tce);
     void CmSettings();
     void CmLoadtc();
     void  CmSimulate();
     BOOL VnRevert(BOOL clear);
     void Paint(TDC&, BOOL, TRect&);

     DECLARE_RESPONSE_TABLE(TDrawView);
};

DEFINE_DOC_TEMPLATE_CLASS(TDrawDocument, TDrawView, DrawTemplate);
DrawTemplate drawTpl("Windows Bitmaps (*.BMP)","*.BMP",0,"BMP",dtAutoDelete|dtUpdateDir);

DEFINE_RESPONSE_TABLE1(TDrawView, TWindowView)
    EV_WM_LBUTTONDOWN,
    EV_WM_MBUTTONDOWN,
    EV_WM_RBUTTONDOWN,
    EV_WM_MOUSEMOVE,
    EV_VN_REVERT,
    EV_COMMAND(CM_HIGHQUALITY,CmHighQuality),
    EV_COMMAND_ENABLE(CM_HIGHQUALITY,CeHighQuality),
    EV_COMMAND(CM_SETTINGS,CmSettings),
    EV_COMMAND(CM_LOADTC,CmLoadtc),
    EV_COMMAND(CM_SIMULATE,CmSimulate),
END_RESPONSE_TABLE;

TDrawView::TDrawView(TDrawDocument& doc,TWindow *parent)
            : TWindowView(doc,parent), DrawDoc(&doc)
{
    DrawDoc->Open(ofRead | ofWrite);

    Image *image;
    image = DrawDoc->GetImage();

    TWindow* thisWindow = TDrawView::GetWindow();
    thisWindow->Attr.W = image->width();
    thisWindow->Attr.H = image->height();

    highQuality = TRUE;
    newView = TRUE;
}

// This Function selects a top elevation when the left button is clicked
void TDrawView::EvLButtonDown(UINT, TPoint& point)
{
```

```
    Image *image;
    image = DrawDoc->GetImage();

    float newElevation;

    if ((point.x > (image->width()-1)) || (point.x < 0) ||
        (point.y > (image->height()-1)) || (point.y < 0)) {
        MessageBox("Point selected is not in the image.","Message",MB_OK);
    }
    else {
        char inputText[40] = "";
        if (TInputDialog(this,"Message","Enter elevation:",inputText,
                                    sizeof(inputText)).Execute() == IDOK) {
            sscanf(inputText,"%f",&newElevation);

            // if new top point is below bottom point and not zero give error message
            if (((newElevation < bottomElevation) || (point.y > bottomPoint.y)) &&
                !((bottomPoint.x==0) && (bottomPoint.y==0))) {
                if (point.y > bottomPoint.y) {
                    MessageBox("Top point was selected below bottom point.\nTry again.",
                                    "Message",MB_OK);
                }
                else {
                    MessageBox("Top Elevation was selected below bottom Elevation.\nTry
again.",
                                    "Message",MB_OK);
                }
            }
            else {
                topElevation = newElevation;
                topPoint = point;

                // Locate Point for User
                WPARAM wParam = (unsigned int) point.x;
                unsigned long y = point.y;
                int elevation = topElevation + 0.5;
                LPARAM lParam = (y << 16) + elevation;
                this->Parent->Parent->Parent->SendMessage(EV_TOPPTMSG,wParam,lParam);
            }
        }
    }
}

void TDrawView::EvMButtonDown(UINT, TPoint&)
{
}

// This Function selects a bottom elevation when the right button is clicked
void TDrawView::EvRButtonDown(UINT, TPoint& point)
{
    Image *image;
    image = DrawDoc->GetImage();

    float newElevation;

    if ((point.x > (image->width()-1)) || (point.x < 0) ||
        (point.y > (image->height()-1)) || (point.y < 0)) {
        MessageBox("Point selected is not in the image.","Message",MB_OK);
    }
    else {
        char tempString[40] = "";
        if (TInputDialog(this,"Message","Enter elevation:",tempString,
                                    sizeof(tempString)).Execute() == IDOK) {
            bottomPoint = point;
            sscanf(tempString,"%f",&newElevation);

            // if new top point is below bottom point and not zero give error message
            if (((newElevation > topElevation) || (point.y < topPoint.y)) &&
                !((topPoint.x==0) && (topPoint.y==0))) {
                if (point.y < topPoint.y) {
                    MessageBox("Bottom point was selected above top point.\nTry again.",
                                    "Message",MB_OK);
                }
                else {
```

```
                    MessageBox("Bottom elevation was selected above top elevation.\nTry
again.",
                                                "Message",MB_OK);
                }
            }
            else {
                bottomElevation = newElevation;
                bottomPoint = point;

                // Locate Point for User
                WPARAM wParam = (unsigned int) point.x;
                unsigned long y = point.y;
                int elevation = bottomElevation + 0.5;
                LPARAM lParam = (y << 16) + elevation;
                this->Parent->Parent->Parent->SendMessage(EV_BOTPTMSG,wParam,lParam);
            }
        }
    }
}

void TDrawView::EvMouseMove(UINT, TPoint& point)
{
    // Set LPARAM to the x and y coordinates
    unsigned long x = point.x;
    unsigned long y = point.y;
    LPARAM lParam = x + (y << 16);
    WPARAM wParam = 0;

    // Set the WPARAM to the intensity of the pixel at x,y
    Image* image = DrawDoc->GetImage();

    // Get pixel value of point if it is within the image
    if (x<image->width() && y<image->height()) {
        Pixel pixel = image->getPixel((unsigned) x, (unsigned) y);
        wParam = (pixel.red+pixel.green+pixel.blue)/3;
    }

    // Send data to status bar
    this->Parent->Parent->Parent->SendMessage(EV_PIXELINFO,wParam,lParam);
}

void TDrawView::CmHighQuality()
{
    // Change the Quality Flag
    highQuality = !highQuality;
}

void TDrawView::CeHighQuality(TCommandEnabler &tce)
{
    // Set the Check Mark if quality is high
    if (highQuality) {
        tce.SetCheck (TCommandEnabler::Checked);
    }
    else {
        tce.SetCheck (TCommandEnabler::Unchecked);
    }
}

void TDrawView::CmSettings()
{
    char tcText[40];
    if (tc.GetItemsInContainer() == 0) {
        sprintf(tcText,"TC:  None loaded.");
    }
    else {
        sprintf(tcText,"TC:  A %d point TC is loaded.",tc.GetItemsInContainer());
    }

    char qualityText[25];
    if (highQuality) {
        sprintf(qualityText,"Quality:  High");
    }
    else {
        sprintf(qualityText,"Quality:  Low");
```

```
    }

    char messageText[255];
    sprintf(messageText,"%s (%d,%d)  %s %.1f\n%s (%d,%d)  %s %.1f\n\n%s\n\n%s\n",
                "Top Point:",topPoint.x,topPoint.y,"Elevation:",topElevation,
                "Bottom Point:",bottomPoint.x,bottomPoint.y,"Elevation:",bottomElevation,
                qualityText,tcText);
    MessageBox(messageText,"Message",MB_OK);
}

void TDrawView::CmLoadtc()
{
    TOpenSaveDialog::TData *FileData;

    // Get the name of the file to be loaded
    FileData = new TOpenSaveDialog::TData(OFN_HIDEREADONLY|OFN_FILEMUSTEXIST,
Characteristics (*.TC)|*.tc|",                                                    "Transfer
                                                                                  0, "","TC");
    if ((TFileOpenDialog(this, *FileData)).Execute() == IDOK) {

        // Message:  Loading transfer characteristic.
        WPARAM wParam = 0;
        this->Parent->Parent->Parent->SendMessage(EV_TCMSG,wParam);
     (this->GetApplication())->PumpWaitingMessages();

        // Find the stream associated with the file name
        ifstream is(FileData->FileName);

        // If the stream isn't valid send error box
        if (!is) {
            MessageBox("No file loaded","Message",MB_OK);
        }
        else {
            // Clear old transfer characteristic
            tc.Flush();

            // Load new transfer characteristic
            is >> tc;

            // Message: Transfer characteristic is loaded.
            WPARAM wParam = 1;
            this->Parent->Parent->Parent->SendMessage(EV_TCMSG,wParam);
            (this->GetApplication())->PumpWaitingMessages();
        }
    }
    else {
        // Message: No file was loaded.
        WPARAM wParam = 2;
        this->Parent->Parent->Parent->SendMessage(EV_TCMSG,wParam);
        (this->GetApplication())->PumpWaitingMessages();
    }
}

void TDrawView::CmSimulate()
{
    // Give a message that we are beginning the simulation
    WPARAM wParam = 0;
    this->Parent->Parent->Parent->SendMessage(EV_PREPARING,wParam);
    (this->GetApplication())->PumpWaitingMessages();

    // If transfer characteristic isn't loaded -> return
    if (tc.GetItemsInContainer() == 0) {
        MessageBox("No Transfer Characteristic Loaded!","Message",MB_OK);
        return;
    }

    // If top point isn't set -> return
    if ((topPoint.x == 0) && (topPoint.y) == 0) {
        MessageBox("Top Point is not set.","Message",MB_OK);
        return;
    }
```

```
    // If bottom point isn't set -> return
    if ((bottomPoint.x == 0) && (bottomPoint.y) == 0) {
        MessageBox("Bottom Point is not set.","Message",MB_OK);
        return;
    }

    // Get the image associated with this document
    Image* image;
    image = DrawDoc->GetImage();

    // Calculate scale of transfer function in meters/pixel
    float imageScale = (topElevation-bottomElevation)/(bottomPoint.y-topPoint.y);
    int horizon = bottomPoint.y;
    int mirageHorizon = horizon;
    int horizonElevation = bottomElevation;
    const int skyPixels = 10;

    // Calculate the total number of lines to be added
    int totalLinesAdded = 0;

    // Redraw image Bigger (when horizon elevation is greater than zero)
    if (horizonElevation > 0.0)   {
        // Calculate the number of pixels to increase image size
        int increasedImageSize = (horizonElevation/imageScale + 0.5);

        // Change Bitmap Size by increaseImageSize number of lines
        // Note: Due to BMP format lines are added to beginning of BMP
        if (image->addLines(increasedImageSize) == NULL) {
            MessageBox("Could not increase image size!","Message",MB_OK);
            return;
        }

        // Inform parent window that the image changed size
        this->Parent->SendMessage(EV_CHANGEIMAGESIZE,
                                                    (WPARAM) image->width(),
                                                    (LPARAM) image->height());
        (this->GetApplication())->PumpWaitingMessages();

        // Total the number of lines added to the image.
        totalLinesAdded += increasedImageSize;

        ////////////////////////////////////////////////////////////////////////
        //                                                                    //
        //   ----------------                 ----------------                //
        //   |              |                 |              |                //
        //   |              |                 |              |                //
        //   |--------------| <- Horizon      |--------------| <- Horizon     //
        //   |**************|                 |    dummy     |                //
        //   |**************|                 |     data     |                //
        //   ----------------                 |--------------| <- Mirage Horizon //
        //                                    |**************|    (zero elev.) //
        //                                    |**************|                //
        //                                    ----------------                //
        //                                                                    //
        //  Copy left image to right image with dummy data in between.        //
        //                                                                    //
        ////////////////////////////////////////////////////////////////////////

        // Row on new image where actual horizon will be
        mirageHorizon = horizon + increasedImageSize;

        // Copy known area above horizon to top of picture
        for (int row=increasedImageSize; row<=(horizon+increasedImageSize); row++)
            for (int column=0; column<image->width(); column++)
                image->putPixel(column,row-increasedImageSize,image->getPixel(column,row));

        // Fill in unknown area below horizon with dummy information
        for (row=(horizon+1); row<=mirageHorizon; row++)
            for (int column=0; column<image->width(); column++)
                image->putPixel(column,row,
                                        image->getPixel(column,(mirageHorizon+1)));
    }

// Locate the top and bottom of the mirage image (in pixels)
```

```
        int mirageTop = mirageHorizon -
                                    (tc[tc.GetItemsInContainer()-1].getApparent()/imageScale);
        int mirageBottom = mirageHorizon - (tc[0].getApparent()/imageScale);

        // Check if the top of the mirage is above the image top
        // If so, expand the image size
        if (mirageTop < 0) {
            int increasedImageSize = skyPixels - mirageTop;

            // Change Bitmap Size by increaseImageSize number of lines
            // Note: Due to BMP format lines are added to beginning of BMP
            if (image->addLines(increasedImageSize) == NULL) {
                MessageBox("Could not increase image size!","Message",MB_OK);
                return;
            }

            // Inform parent window that the image changed size
            this->Parent->SendMessage(EV_CHANGEIMAGESIZE,
                                                        (WPARAM) image->width(),
                                                        (LPARAM) image->height());
            (this->GetApplication())->PumpWaitingMessages();

            // Update the total number of lines added
            totalLinesAdded += increasedImageSize;

            ////////////////////////////////////////////////////////////////////////////
            //
            //    ----------------                 ----------------                    //
            //    |**************|<- Sky Line      |**************|                    //
            //    |              |                 |**************| <                  //
            //    |              |                 |**************| <- Copied Lines    //
            //    |              |                 |**************| <                  //
            //    |              |                 |              |                    //
            //    ----------------                 |              |                    //
            //                                     |              |                    //
            //                                     |              |                    //
            //                                     ----------------                    //
            //                                                                         //
            //  Copy Sky Line a number of times to add room for mirage.                //
            //                                                                         //
            ////////////////////////////////////////////////////////////////////////////

            // Copy first line into all new lines
            for (int row=0; row<increasedImageSize; row++)
                for (int column=0; column<image->width(); column++)
                    image->putPixel(column,row,
image->getPixel(column,(increasedImageSize+(row%skyPixels))));

            mirageTop = skyPixels;
            mirageBottom = mirageBottom + increasedImageSize;
            mirageHorizon = mirageHorizon + increasedImageSize;
        }

        // Check for case when Mirage goes below horizon.  In this case
        // add space with-out changing any of the image below the horizon.
        // That is, add extra space at horizon.
        if (mirageBottom > mirageHorizon) {
            int increasedImageSize = mirageBottom - mirageHorizon;

            // Change Bitmap Size by increaseImageSize number of lines
            // Note: Due to BMP format lines are added to beginning of BMP
            if (image->addLines(increasedImageSize) == NULL) {
                MessageBox("Could not increase image size!","Message",MB_OK);
                return;
            }

            // Inform parent window that the image changed size
            this->Parent->SendMessage(EV_CHANGEIMAGESIZE,
                                                        (WPARAM) image->width(),
                                                        (LPARAM) image->height());
            (this->GetApplication())->PumpWaitingMessages();

            // Update the total number of lines added
```

```
        totalLinesAdded += increasedImageSize;

        ////////////////////////////////////////////////////////////////////////////
        //                                                                        //
        //   ----------------                 ----------------                    //
        //   |              |                 |              |                    //
        //   |              |                 |              |                    //
        //   |--------------|<- Mirage        |--------------| <- Mirage Horizon  //
        //   |              |   Horizon       |**************| <                  //
        //   |              |                 |**************| <- Added Lines     //
        //   ----------------                 |**************| <                  //
        //                                    |              |                    //
        //                                    |              |                    //
        //                                    ----------------                    //
        //                                                                        //
        //  Add extra lines below horizon to allow expansion of mirage.           //
        //                                                                        //
        ////////////////////////////////////////////////////////////////////////////

        // Copy stuff above mirage horizon above added area
        for (int row=increasedImageSize; row<=(mirageHorizon+increasedImageSize); row++)
            for (int column=0; column<image->width(); column++)
                image->putPixel(column,row-increasedImageSize,image->getPixel(column,row));

        // Fill in added area with a dummy horizon line
        Pixel dummyInformation = {0, 255, 0};
        for (row=(mirageHorizon+1); row<=(mirageHorizon+increasedImageSize); row++)
            for (int column=0; column<image->width(); column++)
                image->putPixel(column,row,dummyInformation);
    }

    // type to hold information of how the lines of an image are swapped
    typedef struct imageSwapLinesStruct {
        int lineNumber;
        float smoothingValue;
    } imageSwapLines;

    // declare a variable of type imageSwapLines
    imageSwapLines *lines = (imageSwapLines*) calloc(image->height(),sizeof(imageSwapLines));

    // from transfer function calculate how image lines are swapped and
    // how to smooth from line to line.
    for (int index=0; index<image->height(); index++) {
        float pointHeight = tc.actualHeight((mirageHorizon-index)*imageScale);
        lines[index].lineNumber = ceil(mirageHorizon-(pointHeight/imageScale));
        // if line is below horizon don't smooth it
        if (pointHeight >= 0.0) {
            lines[index].smoothingValue = lines[index].lineNumber -
(mirageHorizon-(pointHeight/imageScale));
        }
        else {
            lines[index].smoothingValue = 0.0;
        }
    }

    // Now check to make sure all lines in array are within the image
    for (index=0; index<image->height(); index++) {
        if (lines[index].lineNumber <= 0) {
            lines[index].lineNumber = 1;
            lines[index].smoothingValue = 0.9999;
        }
        if (lines[index].lineNumber >= image->height()) {
            lines[index].lineNumber = image->height()-1;
            lines[index].smoothingValue = 0.0;
        }
    }

    // Define a column of pixels that will be swapped
    Pixel *transformColumn = (Pixel*) calloc(image->height(),sizeof(Pixel));

    // Now perform mirage transformation defined in lines, one column at a time
    for (int column=0; column<image->width(); column++) {
        // Set up Mirage Simulation Message
```

```
        WPARAM wParam = (column+1)*100/image->width();
        this->Parent->Parent->Parent->SendMessage(EV_SIMULATING,wParam);
        (this->GetApplication())->PumpWaitingMessages();

        // First if -> determines number of colours to perform transform on
        // Second if -> determines whether or not to smooth the transform
        // For loops are inside the if statement to speed transform
        if (image->numcolors() == 0) {
            if (highQuality) {
                for (int row=1; row<image->height(); row++) {
                    Pixel previousPixel = image->getPixel(column,lines[row].lineNumber-1);
                    Pixel currentPixel = image->getPixel(column,lines[row].lineNumber);
                    transformColumn[row].red = (1-lines[row].smoothingValue)*currentPixel.red
+

lines[row].smoothingValue*previousPixel.red;
                    transformColumn[row].blue =
(1-lines[row].smoothingValue)*currentPixel.blue +

lines[row].smoothingValue*previousPixel.blue;
                    transformColumn[row].green =
(1-lines[row].smoothingValue)*currentPixel.green +

lines[row].smoothingValue*previousPixel.green;
                }
            }
        else {
                for (int row=1; row<image->height(); row++) {
                    Pixel currentPixel = image->getPixel(column,lines[row].lineNumber);
                    transformColumn[row].red = currentPixel.red;
                    transformColumn[row].blue = currentPixel.blue;
                    transformColumn[row].green = currentPixel.green;
                }
            }
        }
        else if (image->numcolors() == 256) {
            if (highQuality) {
                for (int row=1; row<image->height(); row++) {
                    Pixel previousPixel = image->getPixel(column,lines[row].lineNumber-1);
                    Pixel currentPixel = image->getPixel(column,lines[row].lineNumber);
                    transformColumn[row].red = (1-lines[row].smoothingValue)*currentPixel.red
+

lines[row].smoothingValue*previousPixel.red;
                }
            }
            else {
                for (int row=1; row<image->height(); row++) {
                    Pixel currentPixel = image->getPixel(column,lines[row].lineNumber);
                    transformColumn[row].red = currentPixel.red;
                }
            }
        }
        else {
            for (int row=1; row<image->height(); row++) {
                Pixel currentPixel = image->getPixel(column,lines[row].lineNumber);
                transformColumn[row].red = currentPixel.red;
            }
        }

        // Perform actual transformation in image
        for (int row=1; row<image->height(); row++) {
            image->putPixel(column,row,transformColumn[row]);
        }
    }

    // free dynamically allocated temporary memory
    free(lines);
    free(transformColumn);

    // Cause the window to be repainted with new bitmap size
    this->Invalidate();
    TRect tempRect;
    tempRect.left = this->Parent->Attr.X;
```

```
    tempRect.right = tempRect.left + this->Attr.W + 8;
    tempRect.top = this->Parent->Attr.Y;
    tempRect.bottom = tempRect.top + this->Attr.H + totalLinesAdded + 27;
    this->Parent->BringWindowToTop();
    this->Parent->MoveWindow(tempRect, TRUE);
    (this->GetApplication())->PumpWaitingMessages();

    // Image is now Dirty -- tell the document this
    DrawDoc->SetDirty(TRUE);
}

BOOL TDrawView::VnRevert(BOOL clear)
{
    if (!clear) DrawDoc->Open(0);
    else return TRUE;

    // Clear the window of current information
    this->Invalidate();

    // Get the image
    Image *image = DrawDoc->GetImage();

    // Resize the window to the image
    TRect tempRect;
    tempRect.left = (this->GetWindow())->Attr.X;
    tempRect.right = image->width() + 8 + tempRect.left;
    tempRect.top = (this->GetWindow())->Attr.Y;
    tempRect.bottom = image->height() + 27 + tempRect.top;
    this->Parent->MoveWindow(tempRect, TRUE);

    return TRUE;
}

void TDrawView::Paint(TDC& dc, BOOL, TRect&)
{
    Image *image;
    if ((image = DrawDoc->GetImage()) != 0) image->show(dc);

    if (newView) {
        char tempString[255];
        sprintf(tempString,"%s%s\n\n%s%s\n",
                        "1. Select a top or peak point (Left Mouse Button)",
                        " and enter its elevation.",
                        "2. Select a bottom or horizon point (Right Mouse Button)",
                        " and enter its elevation.");
        MessageBox(tempString,"Instructions",MB_OK);
        newView = FALSE;
    }
}
/*****************************************************************************/
/************************** End TDrawView Class **************************/
/*****************************************************************************/
```

Bibliography

[Bar1] Barkakati, N., *Imaging and Animation for Windows*, Sams Publishing, New York, 1993.

[Don1] Donn, William L., *Meteorology*, McGraw-Hill Book Company, New York, 1975.

[Fra1] Fraser, Alistair B. and Mach, W.H., "Mirages", *Scientific American*, January 1976, p. 102-111.

[Fra2] Fraser, David M., "The Correlation and Display of Sequences of Mirages", M.Sc. Thesis, 1994, University of Manitoba

[Fri1] Friesen, Wesley J., "Ray Tracing in a 3-D Atmosphere Model", M. Sc. Thesis, 1991, University of Manitoba

[Gos1] Gossard, Earl E. and Hooke, Willian H., "Waves in the Atmosphere", Elsevier Scientific Publishing Company, Amsterdam, 1975.

[Gre1] Greenler, R., *Rainbows, Halos, and Glories*, Cambridge University Press, New York, 1980.

[Kro1] Kropla, W. C. and Lehn, Waldemar H., "Differential Geometric Approach to Atmospheric Refraction", *Journal of the Optical Society of America A*, Vol. 9, No. 4, p. 601-608 (1992).

[Leh1] Lehn, W. H., "A Simple Parabolic Model for the Optics of the Atmosphere Surface Layer", *Applied Mathematical Modelling*, Vol. 8, No. 12, p. 47-453 (1985).

[Leh2] Lehn, W. H., "Inversion of Superior Mirage Data to Compute Temperature Profiles", *Journal of the Optical Society of America*, Vol. 73, No. 12, p. 1622-1625 (1983).

[Leh3] Lehn, W.H., "The Novaya Zemlya Effect: An Arctic Mirage", *Journal of the Optical Society of America*, Vol. 69, No. 5, p. 776-781 (1979).

[Leh4] Lehn, W. H. and Friesen, W., "Simulation of Mirages", *Applied Optics*, Vol. 31, No. 9, p. 1267-1273 (1992).

[Leh5] Lehn, W.H., Silvester, Wayne K. and Fraser, David M., "Mirages with Atmospheric Gravity Waves", *Applied Optics*, Vol. 33, No. 21, p. 4639-4643 (1994).

[Leh6] Lehn, W.H. and El-Arini, M.B., "Computer-Graphics Analysis of Atmospheric Refraction", *Applied Optics*, Vol. 17, No. 19, p. 3146-3151 (1978).

[Leh7] Lehn, W.H. and Schroeder, I.I., "Polar Mirages as Aids to Norse Navigation", *Polarforschung*, Vol. 49, No. 2, p. 173-187 (1979).

[Leh8] Lehn, W.H. and Schroeder, I., "The Norse Merman as an Optical Phenomenon", *Nature*, Vol. 289, No. 5796, p. 362-366 (1981).

[Lil1] Liljequist, G. H., "Refraction Phenomena in the Polar Atmosphere", in *Scientific Results, Norwegian-British-Swedish Antarctic Expedition, 1949-52*, Vol. 2, Part 2, Oslo University, Oslo, 1964.

[Per1] Pernter, J.M. and Exner, F.M., *Meteorologische Optik*, W. Braümuller, Vienna, 1922.

[Sch1] Schiele, W.E., "Zur Theorie der Luftspiegelungen", *Veroeff. Geophys. Inst. Univ. Leipzig*, Vol. 7, p. 103-188 (1935).

[Sha1] Shackleton, E., *South-The Story of Shackleton's Last Expedition 1914-1917*, MacMillan, New York, 1920.

[Sil1] Silvester, Wayne K., personnel communication concerning (untitled) M.Sc. Thesis, 1995, University of Manitoba.

[Vin1] Vince, "Observations on a Unusual Horizontal Refraction of the Air with Remarks on the Variations to which the Lower Parts of the Atmosphere are Sometimes Subject", London Philosoph. Transact. 1799, Part 1. p. 12.

[Vis1] Visser, S. W., "The Novaya-Zemlya Phenomenon", *K. Ned. Akad. Wet. Afd. Natuurkd*, Vol. 59, p. 375-385 (1956).