Methods for Realistic Landscape Imaging

F. Kenton Musgrave
Yale University
1993

Abstract

There are five fundamental concerns in the synthesis of realistic imagery of fractal landscapes: 1) convincing geometric models of terrain; 2) efficient algorithms for rendering those potentially-large terrain models; 3) atmospheric effects, or *aerial perspective*, to provide a sense of scale; 4) surface textures as models of natural phenomena such as clouds, water, rock strata, and so forth, to enhance visual detail in the image beyond what can be modelled geometrically; and 5) a global context in which to situate the scenes. Results in these five areas are presented, and some aspects of the development of computer graphics as a new process and medium for the fine arts are discussed. Heterogeneous terrain models are introduced, and preliminary experiments in simulating fluvial erosion are presented to provide fractal drainage network features. For imaging detailed terrain models we describe *grid tracing*, a time- and memory-efficient algorithm for ray tracing height fields. To obtain aerial perspective we develop geometric models of aerosol density distributions with efficient integration schemes for determining scattering and extinction, and an efficient Rayleigh scattering approximation. We also describe physically-based models of the rainbow and mirage. Proceduralism is an underlying theme of this work; this is the practice of abstracting models of complex form and behaviors into relatively terse algorithms, which are evaluated in a lazy fashion. Procedural textures are developed as models of natural phenomena such as mountains and clouds, culminating a procedural model of an Earth-like planet which in the future may be explored interactively in a virtual reality setting.

KEYWORDS AND PHRASES: computer graphics, fractals, models of natural phenomena, ray tracing, procedural modelling, erosion, aerial perspective, atmospheric scattering, algorithmic art.

Methods for Realistic Landscape Imaging

A Dissertation
Presented to the faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
F. Kenton Musgrave
May 1993

"The thinker tries to determine and represent the nature of the world through logic. He knows that reason and its tool, logic, are incomplete -- the way an intelligent artist knows full well that his brushes or chisels will never be able to express perfectly the radiant nature of an angle or a saint. Still they both try, the thinker as well as the artist, each in his way. They cannot and may not do otherwise. Because when a man tries to realize himself through the gifts with which nature has endowed him, he does the best and only meaningful thing he can do."

- Herman Hesse

Table of Contents

Chapter 1: Introduction..... 1

1.1 The Thesis 1

1.2 Motivation 6

1.2.1 Fractal Terrain Models 6

1.2.2 Efficient Ray Tracing 10

1.2.2.1. Why Ray Trace? 10

1.2.2.2. Why Not to Ray Trace 12

1.2.2.3. Ray Tracing Height Fields 12

1.2.3 Procedural Textures 13

1.2.4 Atmospheric Phenomena 15

1.2.4.1. Atmospheric Scattering 15

1.2.4.2. Efficiency Concerns 17

1.2.4.3. Rainbow Model 17

1.2.4.4. Mirage Model 18

1.2.4.5. Cloud Models 19

1.2.5 Planetary Models 20

1.2.6 Parallel Computation 21

1.2.7 Artistic Opportunity 23

1.3 Manifesto 25

1.4 Contents and Organization of this Dissertation 28

2. Chapter 2: Terrain Modelling 30

2.1 Introduction 31

2.1.1. Origins of Fractal Terrain Models 32

2.1.2. Limitations of FBm Terrain Models 33

2.2 Previous Work 34
2.2.1. Fractional Brownian Motion 34
2.2.1.1. Characterization of fBm 34
2.2.1.2. Band-Limiting 36
2.2.1.3. Generation of fBm 37
2.2.1.4. Terminology 37
2.2.2. Fractal Terrain Models 39
2.2.2.1. Poisson Faulting 39
2.2.2.2. Midpoint Displacement 40
2.2.2.3. Successive Random Additions 41
2.2.2.4. Fourier Synthesis 41
2.2.2.5. Heterogeneous Models 41
2.2.2.6. Summing Band-Limited Basis Functions 42
2.2.3. Erosion Models 43
2.3 Original Terrain Synthesis Models 44
2.3.1. Noise Function 44
2.3.2. Local Control of Statistics 46
2.3.2.1. Homogeneous Procedural FBm Construction 46
2.3.2.2. Local Modulation of Fractal Dimension 48
2.3.2.3. Fractal Statistics by Altitude 49
2.3.2.3.1. The Algorithm 49
2.3.2.3.2. Discussion 50
2.3.2.3.3. Results 51
2.3.2.4. Modulating Lacunarity 52
2.3.2.5. A Large-Scale Terrain Model 53
2.3.2.5.1. The Algorithm 54
2.3.2.5.2. Discussion 55
2.3.2.5.3. Results 56
2.4 Dynamic Erosion Models 57

2.4.1. Fluvial Erosion 58
2.4.2. Thermal Weathering 61
2.4.3. Diffusive Erosion 62
2.4.4. Discussion of Erosion Models 63
2.5 Conclusions 64

Chapter 3: Grid Tracing for Rendering Height Fields 65
3.1 Introduction 66
3.2 The Algorithm 69
3.3 Implementation 71
3.3.1 Modified DDA Algorithm 72
3.3.2 Intersection Tests 73
3.3.3 Memory Requirements 74
3.4 Time Complexity 76
3.5 Future Directions 78
3.6 Conclusions 80
3.7 Code Segment of DDA Inner Loop 81

Chapter 4: Atmospheric Scattering Models 82
4.1 Some Methods for Aerial Perspective 83
4.1.1 Introduction 83
4.1.2 Problem Statement 84
4.1.3 Previous Work 87
4.1.4 Homogeneous Fog 88
4.1.5 Exponential Mist 89
4.1.6 Radial Fog or Planetary Atmosphere 92
4.1.6.1. Motivation 92
4.1.6.2. Density Profile of Atmosphere by Radius 93
4.1.6.3. Density Profile Along an Arbitrary Line 94
4.1.6.4. The Integrating the Atmosphere Function 95
4.1.6.5. Atmospheric umbra 96

4.1.7 A Model of Rayleigh Scattering 98
4.1.8 Conclusions 100
4.2 A Physical Model of Refraction and the Rainbow 102
4.2.1 Introduction 102
4.2.2 Problem Statement 103
4.2.3 Previous Work 105
4.2.3.1. Physics of Refraction 105
4.2.3.2. Physics of Dispersion 106
4.2.3.3. Rainbows 108
4.2.3.4. Computer Graphics 112
4.2.4 Solution 112
4.2.4.1. Sampling in the Frequency Domain of Light 113
4.2.4.2. Representing the Spectrum 115
4.2.4.2.1. Linear Spectrum Model 116
4.2.4.2.2. CIE Spectrum Model 118
4.2.4.3. Rainbow Models 120
4.2.4.3.1. Empirical Rainbow Model 120
4.2.4.3.2. Physical Rainbow Model 121
4.2.5 Conclusion 126
4.3 A Physical Model of the Mirage 127
4.3.1 Introduction 127
4.3.2 A Comment on "Ray Tracing Mirages" 127
4.3.3 Total Internal Reflection in Ray Tracing 130
4.4 Chapter Conclusion 131

5. Chapter 5: Procedural Textures 132
5.1 Introduction 132
5.1.1. Proceduralism as Paradigm 132
5.1.2. Proceduralism and Fractal Models 134
5.1.3. Procedural Solid Textures 135

5.1.4. Proceduralism and Parallel Computing 135
5.1.5. Overview 136
5.2 Planetary-Scale Clouds 136
5.3 A Cyclonic Storm 139
5.4 Venus 141
5.5 Jupiter and Saturn 142
5.6 Terran Procedural Texture 142
5.6.1. Continents and Oceans 143
5.6.2. Climatic Zones by Latitude 144
5.6.3. Fractally Perturbing the Climatic Zones 144
5.6.4. Adding Depth to the Oceans 145
5.6.5. Increasing Realism by Fractal Color Perturbation 145
5.6.6. Result 145
5.7 Conclusions 146
5.8 Sample Texture Code with Auxiliary Functions 148

Chapter 6: Conclusions 153
6.1. Summary of Results 153
6.1.1. Terrain Models 153
6.1.2. Ray Tracing Height Fields 154
6.1.3. Atmospheric Scattering 154
6.1.4. Procedural Textures 155
6.1.5. The Ensemble of Models 155
6.2. Future Directions 155
6.2.1. Terrain Models 156
6.2.1.1. Heterogeneous Terrain Models 156
6.2.1.1.1. Modelling Specific Features 157
6.2.1.1.2. Increasing Variety in Terrain 158
6.2.1.1.3 Using Novel Basis Functions 159
6.2.1.2. Erosion Models 159

6.2.1.2.1. Geophysical Simulation 159
6.2.1.2.2. Animation of Orogenesis 160
6.3. Rendering Terrain Models 160
6.3.1. Parametric Ray Tracing 160
6.3.2. Procedural Ray Tracing 161
6.3.2.1. Promise and Limitations of Procedural Terrain Rendering 161
6.3.2.2. Quad-Tree Method 162
6.3.2.2.1. Calculating Bounding Volumes 162
6.3.2.2.2. Memory Management 163
6.3.2.2.3. Parallel Implementation 163
6.3.2.3. Analytic Method 163
6.4. Atmospheric Scattering 164
6.4.1. The Multiple Scattering Problem 164
6.4.1.1. Rayleigh Scattering 164
6.4.1.2. Clouds 165
6.4.2. Monte Carlo Single Scattering 165
6.4.2.1. Ray Marching 165
6.4.2.2. Atmospheric Shadows 166
6.5. Procedural Textures 166
6.5.1. Clouds 166
6.5.1.1. Geometric Models 166
6.5.1.2. Modelling Turbulence 167
6.5.2. Planets 167
6.5.2.1. Heterogeneous Virtual Planets 168
6.5.2.2. Refining Parameter Space 168
6.6. Modelling Vegetation 168
6.7. Virtual Reality 169
6.7.1. Displays and Input Devices 169
6.7.2. Real-Time Rendering 170

6.7.3. Forecast 171

Appendix A. Parallel Computation 173

7.1. The Linda Paradigm 173

7.2. Ray Tracing 174

7.2.1. Screen-Space Subdivision 175

7.2.1.1. Single Scanline Tasks 176

7.2.1.2. Watermarks 177

7.2.1.3. Multiple Scanline Tasks 178

7.2.1.4. Postage Stamps 179

7.2.1.5. Boundary Tuples 180

7.2.2. Shared-Memory Shared-Bus vs. Distributed Architectures 180

7.2.3. Procedural Height Fields 181

7.2.3.1. Memory Management 182

7.2.3.2. Load Balancing 183

7.2.3.3. Piranha Implementation 184

7.2.4. Interactive Ray Tracing 184

7.2.4.1. Raypaint description 184

7.2.4.2. Granularity 185

7.2.4.3. Piranha Implementation 186

7.2.5. Results 186

7.3. Erosion Simulation 186

7.4. Conclusions 187

Appendix B. A Panoramic Virtual Screen for Ray Tracing 190

8.1. Introduction 190

8.2. Cylindrical Virtual Screen 192

8.3. Vertical Sampling 194

8.4. An Application: Martian Panorama 194

8.5. Conclusion 197

8.6. C Code Segment for Panoramic Virtual Screen 198

Appendix C. Essay: Formal Logic and Self-Expression 201

9.1. Introduction 201

9.2. The Thesis 202

9.3. Foundations 202

9.4. Artwork as Theorem 203

9.5. Theorem as Self-Expression 205

9.6. Deterministic Formalism and the Creative Process 207

9.7. Distinguishing the Process 208

9.7.1. Dimensionality 208

9.7.2. Visual Complexity: Fractal Models 209

9.7.3. Purity of Algorithmic Process 209

9.7.4. Proceduralism 211

9.7.4.1. Functions and Algorithms 211

9.7.4.2. Global Parametric Control 213

9.7.5. Self-Expression vs. Conceptualism 214

9.7.6. Lighting 214

9.7.7. A Model of the Creative Process 214

9.7.7.1. Searching N-Space for Aesthetic Maxima 216

9.7.7.2. Genetic Algorithms 217

9.8. What the Process Is Not 218

9.8.1. A 2-D Canvas 218

9.8.2. Local Control 219

9.8.3. "Of the Hand" 219

9.8.4. Pure Mathematics 220

9.8.5. Computer as Creator 220

9.9. The Process in Action 221

9.9.1. Creating a Metarepresentation 222

9.9.1.1. Creating the Formal System 222

9.9.1.2. Generating Axioms 224

9.9.1.3. Deriving the Theorem 225
9.9.1.4. The Loop of Scientific Discovery 225
9.9.1.5. The Role of Intuition 226
9.9.1.6. The Role of Serendipity 226
9.9.2. Interpreting the Metarepresentation 227
9.9.2.1. Numbers as Colors 228
9.9.2.2. The Finite Number of Possible Outcomes 229
9.9.2.3. Additive vs. Subtractive Color 229
9.9.2.4. Archival Reproduction 230
9.9.2.5. What is the Product? 232
9.10. Discussion 232
9.10.1. What Role Intent and Understanding? 233
9.10.2. What of Turnkey Systems? 233
9.10.3. The Role of Traditional Media 234
9.10.4. Mastering the Process and Medium 235
9.11. Conclusions 236
9.11.1. Constraints and Opportunities 237
9.11.1.1. Working in Three Dimensions 237
9.11.1.2. Algebraic Color 237
9.11.1.3. Proceduralism 238
9.11.1.4. Formal Logic 239
9.11.2. Some parting Questions 239

Appendix D. Essay: Mathematics -- the Language of Nature 242
10.1. Introduction 242
10.2. Mathematics: the Language of Nature 242
10.3. A Bit of History 245
10.4. Fractals in Science and Philosophy 248
10.5. Fractals in Art and Music 250
10.6. The Fractal Calendar 251

Appendix E. Color Plates..... 253

Appendix F. Curriculum Vitae 262

Acknowledgments..... 267

References 268

Figures

Figure 1.1 The trace of fBm. The fractal dimension is 1.2. 7

Figure 1.2 A log/log plot of the power spectrum of $1/f^{\beta}$ noise for various values of β. 8

Figure 2.1 The cubic interpolant $y = 3x^2 - 2x^3$ of the noise function. 45

Figure 2.2 The trace of the noise function. 45

Figure 4.1.1 Graph of the function $density = \gamma(r) = e^{-r^2}$. 93

Figure 4.1.2 Geometric construction of the atmospheric function. 94

Figure 4.1.3 Differing extinction coefficients for Rayleigh approximation. 99

Figure 4.2.1 The dispersion curve at an absorption band. 107

Figure 4.2.2 Descartes' raindrop. 109

Figure 4.2.3 The cone of a rainbow. 111

Figure 4.2.4. The rgb curves of the linear spectral representation. 116

Figure 4.2.5. The rgb curves of the empirical spectral representation. 118

Figure 4.2.6. Summed rgb values, with and without negative values. 119

Figure 4.2.7 The geometric optics of Descartes' raindrop. 122

Figure B.1. Angular width of a pixel as a function of position on the virtual screen. . 191

Figure B.2. Eye point rotation scheme for stereo panoramas. 196

Color Plate Captions

Plate 1.1 "The Road to Point Reyes" 253
Plate 1.2 "Misty Mountains" 253
Plate 1.3 "Zabriskie Point" 253
Plate 2.1 "Bahama" 253
Plate 2.2 Planar to space-filling patch. 254
Plate 2.3 Patch varying fractal dimension from 2.0 to 3.0. 254
Plate 2.4 Terrain model, varying spectral content by altitude. 254
Plate 2.5 "Spirit Lake" 254
Plate 2.6 Spectral content modulated by horizontal position. 254
Plate 2.7 Noise functions and fBm generated from them. 254
Plate 2.8 Heterogeneous terrain morphology on large scale. 254
Plate 2.9 "Hell" 254
Plate 2.10 Terrain patch before erosion simulation. 255
Plate 2.11 Terrain patch after erosion simulation. 255
Plate 2.12 Height field before thermal weathering. 255
Plate 2.13 Height field after thermal weathering. 255
Plate 4.1 "Slickrock" 255
Plate 4.2 "Carolina" 255
Plate 4.3 "Gaea & Selene" 255
Plate 4.4 Detail of "Gaea & Selene". 255
Plate 4.5 Quarter-lit planet with cylinder-umbra model. 256
Plate 4.6 Quarter-lit planet with shaded atmosphere model. 256
Plate 4.7 DLA cluster. 256
Plate 4.8 "Fractal Mandala" 256
Plate 4.9 Dispersing prism on checkerboard, with rainbow. 257

Plate 4.10 Spectral aliasing in the physical rainbow model. 257
Plate 4.11 Spectral antialiasing for physical rainbow model. 257
Plate 4.12 Ideal rainbow as a circle. 257
Plate 4.13 "Medicine Lake" 257
Plate 5.1 "Bay Fog" 257
Plate 5.2 A planetary-scale model of clouds. 257
Plate 5.3 Cloud model, with fBm-valued distortion vector. 258
Plate 5.4 A procedural model of a cyclonic storm system. 258
Plate 5.5 A procedural model of Venus. 258
Plate 5.6 Procedural model of Jupiter and Io. 258
Plate 5.7 Voyager image of Jupiter and Io. 258
Plate 5.8 "Other State" 258
Plate 5.9a-f Construction of a terran procedural texture. 258
Plate 8.1 A panoramic rendering of the Valles Marineris. 258
Plate 9.1 "Blessed State" 259

Chapter 1: Introduction

1.1 The Thesis

This dissertation is about advancing the state of the art in computer synthesis of realistic landscape imagery, or creating "fractal forgeries of Nature". Realistic imaging of synthetic landscapes comprises five basic concerns: 1) convincing geometric models of terrain, 2) efficient algorithms for rendering these potentially large models, 3) aerial perspective to provide a sense of scale, 4) surface textures to enhance visual detail, and 5) a global context in which to situate the landscape scenes. This dissertation presents the significant and original results in these five areas, as well as original work in other related areas such as parallel computation and computer graphics as a medium for the fine arts. These results represent work done since 1987 as part of Professor Benoit Mandelbrot's Fractals Group at Yale University.

Our work on synthetic geometric terrain models was originally motivated by the obvious shortcomings of fractional Brownian motion (fBm) terrain models, and by a suite of ideas on the part of Professor Mandelbrot and myself for improving and extending such models. We have made substantial progress in extending the naive fBm model of fractal terrain to include more of the heterogeneity and complexity that characterizes natural terrains [78]; much of this progress will be detailed in this dissertation.

When we undertook to create realistic (i.e., ray-traced) renderings of the resulting geometric terrain models, it immediately became clear that existing algorithms were not up to the task of rendering the usually-large (i.e., $10^3 - 10^6$ polygon) models reasonably quickly, and with amount of memory typically available on a workstation of the time. Thus we developed the time- and memory-efficient "grid tracing" algorithm.

Fast rendering of convincing geometric models is not, however, sufficient for creating realistic forgeries of nature. Other visual cues are necessary to convey a sense of scale and to enhance realism. A variety of ancillary models, falling into the two classes of *surface textures* and *atmospheric effects* have been developed for these purposes and are presented here.

Given the ability to create appealing three-dimensional landscape scenes and the fact that animation is perhaps the most exciting form of computer graphics, one inevitably wonders how to go about animating these scenes. There are few moving parts in a landscape, so the obvious kind of animation to make is a fly-by. But this begs the question: "Where do we come from, and where to we go to, in this fly-by?" This in turn indicates the development of a global context for the landscape scenes; to this end some results will be presented in developing planetary models. Indeed, the most exciting prospect for this work is the creation of a "virtual universe", populated with "virtual planets" with adaptive level of detail, which one may explore interactively in a virtual reality setting.

In the author's opinion, however, the greatest import of this work lies not in its technical merit as original research in the field of computer graphics, but rather in its significance as the development of a truly novel medium and creative process for the visual arts. The medium is numbers, strings, and logic; the process is distinguished by the use of deterministic formal logic, as embodied in a computer program, to obtain artistic self-expression in representational imagery (i.e., in realistic pictures "of some*thing*

familiar", as opposed to impressionistic or abstract imagery). The fascinating challenge of encapsulating maximal expressive power in terse logical formalisms motivates our emphasis on *procedural modelling.*

Proceduralism as a discipline in computer graphics involves the abstraction of underlying form and/or behavior through time into a compact, elegant procedure or algorithm which can then manifest the form or changes when and where needed, i.e., through lazy evaluation. Large, detailed, explicit specifications of complex models are reduced to encodings in relatively short and simple routines. As the compliment of table-lookup schemes, very little explicit descriptive information is stored; rather, a specification for derivation of the data is developed, and that functional description is evaluated when and where specifics are required. Thus the overhead is shifted from storage space to computation time. The most significant gain in this process is a striking elegance: all the color plates in this document were produced by a few computer programs, the sizes of which are surprisingly small compared to the visual complexity of their output.

The key to the visual complexity manifest in our procedural models and our images is *fractal geometry*. As the title of Mandelbrot's opus *The Fractal Geometry of Nature* [77] states, fractal geometry is very much a language of nature; arguably even more so than the more-familiar Euclidean geometry. Our work has been largely motivated by the goals of 1) developing the illustrations of the descriptive potency fractal geometry as a language of nature, and 2) of expanding fractal geometry's descriptive vocabulary within the domain of computer graphics. The first goal has lead to the creation of "forgeries of Nature" of unprecedented realism, while the second motivated the development of a variety of novel fractal models of natural phenomena for computer graphics.

One distinguishing aspect of fractals is that they are generally specified not by an equation such as $y = f(x)$, which may be solved explicitly, but rather by a recurrence

relation, e.g., $x_{i+2} = f(x_{i+1}) = f(f(x_i))$, which generally defies human patience and precision to evaluate manually. However, such descriptions can map directly to recursive or iterative routines in computer programs, and computers are capable of tireless calculation with great precision and accuracy. Thus fractals and computers are inextricably intertwined. Fractals are all also distinguished by their great complexity; their complexity is so great that in practice they can only be apprehended visually (as the greatest bandwidth to the human cerebral cortex by far, is through the sense of vision). Thus fractals and computer graphics are also inseparable: computer graphics are necessary for the visualization of fractals, and fractal images have, since the early days of computer graphics, provided some of the most interesting and striking examples of computer-synthesized imagery. Research in fractal geometry indeed represents some of the earliest visualization-driven scientific research. [79]

In the course of this work, we claim to have created fractal landscape models of unprecedented realism, an efficient algorithm for ray tracing them, and procedural fractal textures to add visual complexity to our scenes. But this is not yet sufficient for creating realistic forgeries of nature. The terrain models possess no intrinsic scale and thus will tend to be interpreted in a naive rendering as being of about the size of the rendering, i.e., on the scale of inches or feet. Special visual cues are required to convey the impression of grand scale intended in landscape renderings. For hundreds of years, landscape painters have employed *aerial perspective* -- a decrease in contrast and change in hue with distance due to atmospheric scattering of light -- as the primary visual cue for indicating truly large scale. Thus we have expended considerable effort in developing computationally efficient models for generating aerial perspective.

Aerial perspective is not a homogeneous effect. Scattering is a function of aerosol density; aerosol density is in turn a function of altitude, plus other factors such as temperature, humidity, etc., which we ignore for the sake of simplicity. Furthermore, a

ray's *optical path*[*] through the atmosphere is limited geometrically by the objects in the scene and, failing to intersect those, by the curvature of the atmosphere around the planet itself. (Otherwise a horizontal ray might have an optical path of infinite length; this is never the case in Nature.) To model scattering, then, we require realistic geometric models of aerosol density distributions and an effective model of scattering, over those distributions. Efficiency is of paramount importance, therefore the aerosol density distribution function should be easily integrable, and the scattering model should be computationally simple. The best models of nature from the scientific literature fit neither criterion.

Our need for aerial perspective has lead us to develop a suite of atmospheric effects, both aerosol distribution models and scattering models. These culminate in a continuous planetary atmosphere model, with a Rayleigh scattering approximation.

The availability of this model, plus the need for a global context for our landscapes, has lead us to the development of models of entire planets. Our ongoing research concerns itself with the technical problems of situating our realistic local scenes upon the face of an artificial planet. This entails making the planet model consistent over all areas, such as land and sea, night and day, dawn and dusk, and imbuing it with a realistic measure of heterogeneity. Having such a global model calls for *adaptive level of detail*, so that we my render it from any perspective or point of view, and have the rendering be automatically generated with the appropriate sized features in all visible areas. This in turn indicates use of a procedural model, as we lack the indefinite storage space for explicit models of such detail. Rendering with adaptive level of detail poses significant challenges, which are the object of research underway at the time of this writing.

---

[*] We define *optical path* as the interval of a ray's traversal through a participating medium, such as an atmosphere.

Five years of experience in generating realistic synthetic landscape imagery, has molded the author's considered opinion to be that these items -- good terrain models, efficient rendering schemes for them, visually complex surface textures, aerial perspective, and planetary contexts for the local terrain models -- are essential to a comprehensive and coherent capability for generating such images. This is the thesis we present in this dissertation.

1.2 Motivation

1.2.1 Fractal Terrain Models

This work began in 1987 when the author was employed by Professor Benoit Mandelbrot in the Yale University Department of Mathematics to program some improved schemes for generating fractal terrain models. At the time, there had recently been some controversy over the efficacy of polygon subdivision algorithms for creating fractional Brownian motion (fBm) approximations [32,76]. Mandelbrot had some ideas on how to eliminate the so-called *creasing problem*[*] and to vary the character of the final fBm surface. This work on polygon-based fractal terrain models culminated in Mandelbrot's appendix to *The Science of Fractal Images* [78] in which he describes the hexagon subdivision algorithms we developed together, to solve the creasing problem and to incorporate drainage networks in the terrain model.

---

[*] The *creasing problem* refers to unnatural linear artifacts in the fBm surface due to nonstationary statistics, which are traceable to the interpolant implicit in the subdivision scheme. [86] See the mountains in the background in Plate 1.1 for an example of the creasing problem.

Figure 1.1 The trace of fBm. The fractal dimension is 1.2.

Fractional Brownian motion is a random fractal function which happens to resemble the skyline of a mountain range. When extended from a curve to a surface, fBm of a suitable fractal dimension (i.e., 2.1-2.3) resembles mountains. However, fBm is by design homogeneous and isotropic, while real terrains are neither. In real terrains there is asymmetry across the horizontal plane, as peaks and valleys have different roughnesses: in rugged alpine terrains peaks tend to be more jagged than valleys (as valleys tend to fill up with detritus and undergo periodic smoothing and sculpting by glaciers), while in certain sub-alpine terrains (e.g., where diffusion is the primary erosional transport mode), the hilltops can be more rounded than the valleys. Furthermore, real terrains are heterogeneous on the large scale: mountains can rise out of flat plains, generally preceded by rolling foothills, and erosive processes create distinctive features which are globally coherent. (Indeed, fluvial and glacial erosion are the primary morphogenic agents shaping most natural terrains.) Naive fBm terrain models include none of these features; a part of this work has been to develop terrain models which do.

Fractional Brownian motion is perhaps most succinctly described by the Weierstrass-Mandelbrot function [152]:

$$V(t) = \sum_{f=-\infty}^{\infty} A_f r^{fH} \sin(2\pi r^{-f} t + \theta_f) \tag{1.2.1}$$

where A is a Gaussian random variable, r is the spatial resolution or *lacunarity*, θ is a uniform random variable in the range $[0, 2\pi]$ providing a random phase, and H is the Hölder exponent, which determines the fractal dimension. The Weierstrass-Mandelbrot function is basically an infinite sum of sine waves at discrete frequencies and with random phase, with a certain size gap between successive frequencies (the *lacunarity*), and an exponent H which scales amplitude by frequency. Amplitude relates to frequency f by the plot of the *power spectral density*[*] [152]

$$S_{WM} \propto \frac{1}{f^{H+1}} \tag{1.2.2}$$

and hence the result is referred to as $1/f$ *noise*. The stochastic fBm function is succinctly characterized by its power spectrum, as shown in Figure 1.2.

Figure 1.2 A log/log power spectrum of $1/f^{\beta}$ noise for various values of β.

[*] The *power spectral density* $S(f)$ is the power $P(f)$ of the signal at frequency f, or the mean squared power over interval Δf centered at f, divided by Δf: $S(f) = P(f)/\Delta f$. The plot of the power spectral density is known as the *power spectrum*.

In practice in computer graphics we generally utilize *discretized* fractional Brownian motion, in which the lacunarity is usually ~2.0 and the summation is over a rather small range, e.g., 3 to 8 octaves (*octaves* ≡ doublings of frequency). Larger summations are unnecessary-to-detrimental: frequencies much lower than the size of the rendering viewport are simply not visible except as a bias to the slope of the terrain, and frequencies appearing in image space at higher than half the screen resolution will, by the Nyquist Sampling Theorem [116], suffer *aliasing* and, because the underlying function is stochastic, appear as noise.

In the small spectral summations typically used for the construction of discretized fBm, the character of the basis function shows through clearly in the final result. In the Weierstrass-Mandelbrot function and *Fourier synthesis* methods [133,152] the basis function is a sine wave, whereas for polygon-subdivision schemes it is a sawtooth wave. The former suffers from periodicity artifacts, while the latter is by nature jagged, and may (as with *nested* [78] subdivision schemes) suffer from the "creasing" problem. To avoid these and other problems, we have developed the *noise synthesis* [106,119] method, wherein the basis function is a band-limited stochastic function, consisting of a piece-wise tri- or bi-cubic polynomial interpolant. This approach eliminates periodicity and yields a nearly isotropic, visually well behaved result, albeit with a somewhat non-intuitive frequency content. [134]

Another goal of our terrain-generation research has been to model erosion features. The local variation of frequency content in the heterogeneous terrain models of the noise-synthesis method [106] is designed to model locally coherent erosion features, such as valleys that are smoother than peaks. The problem of including the coherent, context-

sensitive* features of fluvial drainage networks in synthetic terrain models had previously been addressed in the literature [61], but not in the context of a fractal surface model.

Work has therefore been undertaken in developing simulations of fluvial erosion processes, which can be applied to preexisting terrain models. [106] These models, developed with the assistance of Professor Edward Bolton of the Yale Department of Geology & Geophysics, have been based on erosion laws from the literature of fluvial geomorphology. [3,135]

1.2.2 Efficient Ray Tracing

Again, a convincing geometric terrain model is only part of what is required to synthesize realistic images of natural scenes. Realism is a function of comprehensive lighting models and both gross form and fine detail in the geometric models. *Ray tracing* [158] provides a quite believable -- if not comprehensive -- lighting model, if at significant computational cost. A high level of geometric detail implies a large model, in terms of number of primitive objects. With ray tracing, the asymptotic time complexity is generally a function of the number of primitives, among other things. Larger models take longer to render.

1.2.2.1. Why Ray Trace?

At the outset of our work, we chose to ray trace our scenes for a variety of reasons. Foremost among these was realism: as a global illumination model, ray tracing can provide an excellent visual approximation to reality, within a conceptually elegant paradigm. The realism is largely due to the fact that perspective projection, shadows and

---

* As the formation of drainage networks corresponds to the drainage of potential from an area, what happens at every point is a function of what happened "upstream", thus the problem is context-sensitive. Note that it is very much like the electric discharge patterns in dielectric breakdown, another natural phenomenon which is computationally expensive to simulate.

reflections issue naturally from the geometric optics of the ray tracing algorithm. The elegance of the paradigm means that a single researcher can handle designing and implementing the rendering software as a "sideline" to his or her "real" research, which in this case is developing models of natural phenomena for computer graphics, and perfecting the images themselves.

Ray tracing is a semi-physical model of the propagation of light in the world, the primary simplifying assumptions being that the light rays propagate out from the eye into the world, as opposed to originating from light sources, and that a reasonably small number of point samples can be used to reconstruct acceptable approximations to illumination integrals. Invoking this model, the rest reduces to standard local illumination calculations, geometric optics, and numerical integration methods.*

The geometric optics of ray tracing nicely accommodates two important features of our renderings: procedural textures and atmospheric effects. Not all rendering algorithms make available the spatial information required for evaluation of these effects, i.e., the world-space coordinates of the ray endpoints. Scanline algorithms, for instance, operate in screen space, not in world space, thus the world-space coordinate values of the point samples are never computed. The geometric optics and recursion of the ray tracing algorithm make the required information available in a straightforward way, at the appropriate junctures in the rendering process.

In full disclosure, it must be admitted that the decision to use ray tracing was prejudiced as well by the fact that the author was already in possession of, and had thorough working knowledge of, the "Optik" [1] ray tracing program, written by John

---

* The integrand is generally an arbitrary function (such as the illumination over the area of the image plane that a pixel represents), therefore the integral is properly regarded as inherently intractable. As a result of this, the Nyquist sampling theorem, perceptual considerations, and the need for speed, numerical integration is generally accomplished with a Monte Carlo variant of the midpoint rule. [23]

Amanatides and Andrew Woo of the University of Toronto, and was loathe to start afresh with an unfamiliar and/or untried rendering package (much less write one from scratch!).

1.2.2.2. Why Not to Ray Trace

There are at least two good arguments against ray tracing: 1) it may be conceptually elegant, but it is a computationally expensive algorithm, and 2) it is notoriously slow for large models, as the time complexity of the naive algorithm is linear with the number of primitives in the model. The number of primitive polygons tessellating a height field surface varies as the square of the resolution of the height field, so this number can get quite large indeed. (Typically, in our scenes, the number of primitives is on the order of 10^4 to 10^6.) Fortunately, algorithmic advances described here and the parallel processing power made available by the C-Linda coordination language [16], put ray tracing within the realm of feasibility.

1.2.2.3. Ray Tracing Height Fields

At the outset, using a Sun 3/60 workstation with 8 Mb of RAM, it took nearly an hour of run time just to load into the ray tracing program the (several megabytes) ASCII text file in which each triangle was described separately, and that for a height field of only about 10^4 triangles! With this as motivation, it was noted that 1) there is enormous regularity and spatial coherence in a height field, and 2) there can be a far more compact representation than an explicit ASCII description of each polygon. The first admits the efficient use of spatial-subdivision schemes with rapid data-structure traversal. The second, along with the observation that no one is likely to want to directly inspect the height field data as such, indicates storing the height field data in raw, binary form. Some information, such as shared vertices in the final polygon mesh, is implicit while some, such as surface normals and polygon plane equations, is computed on-the-fly and not (permanently) stored.

As detailed terrain models for high-resolution images are perforce large, ray tracing them was, circa 1987, a primarily memory-bound problem. At the time when the "grid tracing" algorithm described here was developed, the primary published competing algorithm was the quad-tree decomposition of Kajiya [54] and Mastin et al. [82] The $O(\log_4 n)$ memory overhead of the quad-tree data structure was deemed excessive at the time (and is still occasionally found to be so at the date of this writing). Recently, Paglieroni and Peterson [115] have developed the very time-efficient "parametric" algorithm, based on so-called *distance estimators*, for ray tracing height fields. While the quad-tree algorithm is significantly faster than grid tracing, and Paglieroni's results indicate that the parametric algorithm is *much* faster than both, both quadtree and distance-estimation algorithms incur significant overhead in memory. While this is less of an issue by the time of this writing in 1993, the grid tracing algorithm remains the most memory-efficient of published height field ray tracing algorithms.

1.2.3 Procedural Textures

Convincing geometric terrain models and efficient ray tracing methods are still not sufficient for creating convincing landscape images. With a terrain model of fixed, homogeneous resolution, the perspective rendering projection will cause the geometric detail to appear insufficient in the foreground, while in the distance it might exceed the Nyquist limit in image space. Procedural geometric models with adaptive screen-space frequency content are difficult to program, generally quite slow to evaluate (i.e., to render), and are still under development at the time of this writing. Thus today they are still not an immediately viable solution to the problem of lack of geometric detail, and they would have been completely out of the question at the outset of this work, in 1987, because of the speed of the processors and the amount of main memory available then.

Fortunately, surface textures can be used to distract attention from overly-simple model geometry. *Texture maps* can add detail by modulating the color of the surface.

The apparent surface geometry, for the purposes of lighting calculations, can be modified the surface normal perturbation technique of *bump mapping*. [11] Fractal bump maps applied to Euclidean surfaces can add visual complexity and provide a natural (i.e., non-Euclidean) appearance.

Procedural or *solid* textures offer remarkable flexibility in texturing. In this approach, the texture is defined as a function $T: R^3 \rightarrow S$ mapping three-space to S, the set of surface properties such as color, orientation (as defined by the surface normal), transparency, etc. Thus the texture function T takes as arguments the world- or object-space coordinates of the point at which its value is required, and returns the surface properties it was designed to modify (most often, the color and/or surface normal). The function comprises the abstract description of the often-complex visual behavior of the texture.* Complex procedural textures are usually fractal, because of the terse abstraction of visual complexity available in the fractal paradigm. Procedural texture functions may also use information additional to the coordinates of the evaluation point: for instance, one might reference the distance from the eye point, in order to clamp the screen-space frequency content below the Nyquist limit.

The work on procedural textures presented here was largely motivated by the opportunity seen in Perlin's seminal 1985 SIGGRAPH paper [119]: the illustrations in his paper demonstrated a naturalism previously unapproached in computer-synthesized imagery, in the author's view. The power of Perlin's stochastic procedural approach was clear and called for detailed investigation; this dissertation is largely an exposition of the results of that investigation.

---

* See Plate 4.3, which consists only of three spheres, an atmosphere function, and four procedural textures: one for the planet surface, one for the clouds, another for the moon's surface, and another for the stars in the background. The visual complexity is almost entirely due to the procedural textures; the underlying geometric model is trivial and virtually featureless.

Apart from the interface design, the three key points to Perlin's paper are: 1) the procedural approach, in which the texture is described as a function over three-space, rather than a preexisting 2-D image to be mapped, 2) the band-limited stochastic "noise" function, the aperiodic nature of which makes it a favorable substitute for the sine wave used in spectral-synthesis of fractal models, and 3) the creation of $1/f^{\beta}$ noise textures using this function. While there is a variety of competing schemes [38,72-74,160], the *noise synthesis* method remains the author's first choice due to its generality, simplicity, and elegance. The repertoire of natural-looking visual effects which may be obtained with such textures can be expanded indefinitely.

1.2.4 Atmospheric Phenomena

Good geometric models, efficient rendering schemes, and natural looking surface textures are still not enough for creating realistic landscape images. There is no intrinsic scale in any of these models; yet we intend the images to represent scenes of vast size. Landscape painters have known for centuries that the primary scale cue in large scenes is *aerial* or *atmospheric perspective* [40], or the change in color and decrease in contrast with distance, due the effects of atmospheric scattering of light. Thus it has proved necessary (and rewarding) to develop some practical models of wavelength-dependent Rayleigh scattering and aerosol distributions. The latter consists of constructing geometric models of atmospheric density structure, which determines the local optical densities, which in turn determine the integral of the scattering along the ray path.

1.2.4.1. Atmospheric Scattering

Development of Rayleigh scattering models is motivated by the scale-cueing effectiveness of the color change with distance. It is Rayleigh scattering which generates the familiar "purple mountain majesties", the blue color of the daytime sky, and the red in sunsets. Development of geometric models of aerosol density distributions was

motivated first by the desire to make better looking pictures: pictures which are more realistic, in that the optical density is greater at lower altitudes. This lead to the original, though not unprecedented, development of an exponential-with-altitude aerosol distribution model. (See Plate 1.2.) This model is geometrically flawed, however, as all rays which miss the terrain will proceed to infinity (or numerical representation thereof) and may thereby integrate to the base color of the atmosphere (e.g., white). This problem can be side-stepped in static images by using a vertical plane in the near background to limit the optical path and thereby the integration; indeed, this is the approach we used in our early renderings. However, it was foreseen that the artificial horizon created at the background plane/terrain intersection might become visible in animated fly-bys, much to the detriment of realism.

For these reasons a radially-symmetric atmosphere model was developed, in which the aerosol distribution used in the integral is a close geometric approximation to that of a "real" planetary atmosphere, i.e., varying exponentially with radius from a point in space. Then the scattering integration is bounded by the "right" (i.e., geometric) constraints: even a horizontal ray will eventually exit the atmosphere; thus all rays have a finite optical path through the participating medium.

The atmosphere around the planet in Plate 4.3 demonstrates both the radial atmosphere model and our Rayleigh scattering approximation. The latter is evident in the region near where the planet occults the moon: notice in the close-up (Plate 4.4) that the color of the (homogeneous) atmosphere model transitions smoothly from scattering-dominated -- the blue against the black backdrop of space -- to extinction-dominated, where it filters the pale gray lunar surface to a smoggy orange color, as with sunsets on Earth. We maintain that it would be difficult to obtain realistic terrain renderings, in general, without such effects at our disposal.

1.2.4.2. Efficiency Concerns

As the primary concern in this research has been more at the generation of realistic images, than the development of elaborate and/or accurate models, simplicity in the models has been a key goal. Thus the elaborate "physical"* Rayleigh scattering models of Klassen [64] and others, have been eschewed in favor of a more computationally efficient approximation.

The goal of a Rayleigh scattering model for computer graphics purposes is to recreate the blue sky and the bluing of terrain with distance, as well as the reddening of pale colors over distance due to extinction. As Rayleigh scattering is a wavelength-dependent phenomenon, proper modelling would require integration of energy over the spectrum of visible light, as well as integration of scattering and extinction over the optical path. In production image synthesis, calculating these integrals on a per-ray basis is currently impractical. Therefore we have developed a greatly simplified *rgb* (red/green/blue) approximation to Rayleigh scattering and extinction, which may be evaluated over arbitrary ray paths at a constant cost, that cost being about three times that of non-wavelength-dependent scattering. While the model lacks some of the physical veracity of a more elaborate (i.e., Klassen-type) model, it is far more efficient and thus better suited to our goal of production image synthesis. We claim that it stands as a legitimate, practical counterpart to more-accurate "physical" models.

1.2.4.3. Rainbow Model

Nevertheless, we have not entirely shied away from rigorous, physical modelling of wavelength-dependent scattering effects. We present a physical simulation of the

---

* We use quotation marks around the word "physical" in this case because Lord Rayleigh's original model of wavelength-dependent atmospheric scattering [148] was in fact an empirical model, based on measurements, not on first principles of physics.

rainbow, as an implementation of the model first proposed by René Descartes in 1637 and an extension of the author's Masters thesis work. [101,102] This model is based on the simulation of the propagation of light through an idealized (spherical) raindrop, taking into account the wavelength-dependent phenomena of dispersion and Fresnel reflection and refraction. While this model suffers from the dearth-of-"bang-for-the-buck" typical of many physical models for computer graphics, the simulation need be performed only once: the axially symmetric nature of the rainbow means that the results may be stored in a one-dimensional lookup table for all future use in rendering. For this reason, a physical approach was deemed feasible for this model.

Work on this rainbow model, along with previous Masters degree research, elucidated the difficulty of working with monochromatic samples of the visible spectrum [95]: all such samples lie outside the color gamut of any reasonable display device, as may their linear combinations. This observation informed and motivated our decision to develop the rgb approximation for Rayleigh scattering.

1.2.4.4. Mirage Model

The appearance of a seemingly flawed exposition of a model of the mirage [8] led to our development of a semi-physical model. [95] We label this "semi-physical" because it incorporates certain physical interactions (i.e., the phenomenon of *total reflection* or *total internal reflection* seen when light rays encounter, at a glancing angle, a transition from higher index of refraction to lower index of refraction), while other aspects of Nature are replaced with simplified models (as the complex atmospheric density structure involved in mirages is replaced with a planar discontinuity in refractive index, with a fractally-perturbed surface normal).

Interestingly, development of this model led to one of our best-known images -- "Zabriskie Point" (Plate 1.3) -- which appeared, among other places, on the cover of

IEEE Computer Graphics & Applications [91], as a two-page spread in the Communications of the ACM [63], and in the 1991 SIGGRAPH Art Show. This image is significant in that it straddles the boundary between art and science, and represents an original result in both. It was conceived and designed as an illustration of a model for a technical paper [95], and as such represents scientific illustration. Yet the author makes the (incontrovertible) claim as an artist, that the image also represents artistic self-expression, obtained through the peculiar creative process we will illuminate in an appendix. The acclaim it has garnered as an artwork bespeaks its aesthetic success. Furthermore, it may represent a truly original work of fine art: S. D. Gedzelman, an authority on the uses of atmospheric effects in the fine arts, claims that there has been no prior appearance of a mirage in a landscape rendering. [40] At any rate, this image stands out as an example of simultaneous success and import in both art and science, and as such may represent the most interesting and significant of our images.

1.2.4.5. Cloud Models

A key feature in nearly all artistic landscape renderings which show a significant area of sky, is the clouds. Modelling clouds is an area of significant difficulty for computer graphics: while it may be fairly easy to conceive of reasonable three-dimensional fractal geometric models for clouds (as with fractal hypertextures [120]), realistic rendering of such models is, to date, impractical. This is due not only to the complexity of the geometric model, but also to the fact that multiple scattering by high-albedo particles is a critical element of their illumination. While applicable volumetric illumination models have been described [57,130] and could conceivably be evaluated in linear time [46] even for complex geometric models, this remains an open problem which is expected to be challenging in any case.

Thus, with the notable exception of Gardner's work [39], cloud models in computer graphics have generally been two-dimensional. In service of our landscape imagery, we

have developed some simple two-dimensional fractal cloud models for use as backdrops, as well as a straightforward procedural implementation of Gardner's ellipsoid-texturing approach. As our work on landscapes expanded to planetary-scale renderings, observations of planetary cloud systems lead to the development of various procedural texture models for such phenomena. At least one of these, in turn, proved useful in landscape renderings: in Plates 1.2 and 1.3, the cloud model which was developed for the planet model see in Plate 4.3 is used effectively on a more local scale.

1.2.5 Planetary Models

Again, as animated fly-bys of landscape scenes were contemplated, the questions arose: "What to do for the greater context? Where to come from? Where to go to?" This lead us to think, literally, in global terms. The need for a geometrically correct atmosphere model, in order to naturally limit integration of scattering effects, had already lead to the planetary atmosphere model. Concerns with modelling terrain at very large scales were leading to heterogeneous terrain models, which model the variety of natural terrains on the continental scale. Ideas for procedural textures to represent both planetary surfaces and cloud tops were manifold. Thus we undertook the development of models of a variety of planets.

Problems associated with this undertaking include: the development of terrain models at the largest scale, including features such as those associated with tectonics and orogenetic belts; development of convincing visual models of turbulent flow in planetary atmospheres, such as the cyclonic storm systems seen on Earth and in gas giant planets; transition of model geometry and lighting calculations from surface texture models at a distance, to full geometric description up close; managing image-space frequency content of the models over a wide range of scales; and abstraction of a reasonably small parameter space in which convincing descriptions of a multitude of interesting worlds

may be found by primarily stochastic means (to facilitate automatic, random population of a virtual universe with virtual worlds worth investigating).

The opportunity presented by a detailed, heterogeneous procedural fractal planet model is significant: as the model is fractal, it may be evaluated over a wide range of scales; as it is procedural, it need never be stored explicitly, rather its description may be stored in a compact, implicit algorithmic form. The net result will be the capability to populate a virtual universe with stochastic, procedural virtual worlds, which will be as full of surprise and serendipity for their creator as for the casual observer/explorer. In years to come, when the capability to render such complex scenes in real-time is in place, we will be able to explore these worlds interactively in a virtual reality setting -- this is indeed an exciting prospect. Work in this area is expected to continue for years into the future.

1.2.6 Parallel Computation

The capacity to develop and execute the algorithms described in this dissertation has hinged upon the availability of substantial parallel computing power, as provided by the C-Linda coordination language. [16] Computer graphics in general requires significant computational resources, in terms of both time and memory. The former is often addressed by the application of special-purpose hardware to common graphics routines such as line drawing and polygon scan-conversion; the latter is ever less an issue, due to the decreasing cost and increasing quantities of both volatile and non-volatile storage media. Graphics algorithms such as ray tracing, however, are too complex and ephemeral to merit the development of mass-produced special-purpose hardware, and algorithms which are still under development (as are most advanced graphics algorithms) are not generally strong candidates for implementation in custom VLSI chips. Also, the programs developed to implement graphics algorithms may be large, and generally much

effort has gone into optimizing the code. Such programs represent a significant investment, largely of programming time.

The net indication of these factors is the development of portable code packages which may readily be executed on the latest, improved computer architecture -- the "machine of the month". Development of platform-specific, non-portable code may be folly: the time required for a single researcher to write and debug, for instance, a full ray tracing program is potentially greater than the mean-time-to-obsolescence for any given platform, and such a time investment in rote code-rewriting could severely impact the time left for "real" research and for developing new code. Using C-Linda, we have been able to take our existing ray tracers (Optik [1] from the University of Toronto, and Rayshade [67] from Yale and Princeton Universities) written in vanilla (i.e., Kernigan & Ritchie) C and parallelize them, with only about a day or two of design and programming work. The capacity to compile and run sequentially was retained, through the use of conditional compilation flags, and a high degree of portability was retained, aside from assumptions made re shared-memory and distributed parallel environments (e.g., when to read the scene description file, before or after calling `eval()` to create more processes).

It is a time-honored heuristic in computer graphics, that users are willing to wait at most one to four hours for a rendering to complete; the author is no exception to this rule. Thus the quality of synthetic imagery is in part limited to what can be rendered in this time. This indicates two ways to improve capacity: develop faster code and algorithms, and use faster computers. Parallelism is a way to multiply the computational power at one's disposal, for appropriate applications. Fortunately, many graphics applications are "embarrassingly parallel". In particular, ray tracing readily admits to MIMD parallel computation: a simple screen-space decomposition, with the model globally distributed or in shared memory, yields a speedup which is nearly linear with the number of processors brought to bear. Because the algorithm is CPU-bound and, for the most part, locally

independent, there is little overhead due to communication and memory access contention. Thus parallel ray tracing as facilitated by C-Linda has allowed the use of models which would otherwise be impractical, due to the four-hour rule, to vastly improve the quality of our images.

Indeed, the extent to which these images excel in the current practice of computer graphics is largely creditable to the computational power that Linda has made available. Some of the elaborate procedural models we use would simply be impractical without the power of parallelism to evaluate them. With parallel computation, renderings at very high resolutions can be accomplished in a reasonable amount of time; thorough supersampling (e.g., up to 256 rays/pixel) for antialiasing purposes becomes more practical. The computational power laid at our doorstep by Linda is, fundamentally, the bedrock upon which this work is built; it is safe to say that without it, much of the work would not even have been undertaken. Best of all, from the author's point of view, is the simplicity and transparency of Linda in practice: like the computers we use and the cars we drive, we can take it for granted, without knowing much about its internals. We have very little to say about Linda -- and that is exactly how we would have it. That statement is rightly viewed as a powerful endorsement of the Linda paradigm for parallel computation.

1.2.7 Artistic Opportunity

The most exciting and significant aspects of this work, in the author's view, lie in the artistic opportunity and accomplishment it represents. What we are doing amounts to the development of a medium and process which is simultaneously new to the fine arts, significantly different from its predecessors, and of significant intellectual depth. This work has been not only about expanding our repertoire of models of natural phenomena for computer graphics, but also about expanding our visual vocabulary of form through the potent new dialect of fractal geometry, and about developing the artistic process of

obtaining self-expression in representational imagery strictly through the access provided by deterministic formal logic, as embodied in computer programs.

In contemporary art there is a strong emphasis on *process*, as opposed to *product*. Our work is significant not because of the character of the images produced (which are often deliberately prosaic), but rather because of the character of the process of their creation. This process is founded upon, and informed by, the disciplines of computer science, formal logic, mathematics, and the physical sciences. Adherence to a purism in process insures that all of our images are the deterministic result of a computation, and thus equivalent to a theorem proved in a formal system. This, we claim, is a novel and significant result for the visual arts. This purism-of-process also fortifies the legitimacy of the images as illustrations of the descriptive power of fractal geometry as a language of nature and of natural form: the images issue directly from the algorithms; they are not in any direct sense "of the hand". Their power in reflecting Nature is encoded entirely in the terse formalism of the generating algorithm; it is not the result of manual skills of the artist. The descriptive power of science and mathematics has been brought to bear in aesthetic expression.

This work has been motivated largely by the perceived opportunity of developing a new medium for the fine arts, and of developing this peculiar creative process. Thus it is important for the reader to recognize that over the five and one half years that this work was developed, even more time and effort went into aspects of mastering the medium and process, than went into aspects of computer graphics research. While we will not belabor our artistic concerns in this dissertation, it is hoped that some of the fruit of the aesthetic labors is recognizable in the images produced; and we will illuminate some ideas about the significance of this work to the fine arts, in an appendix.

1.3 Manifesto

This work has been image-driven. That is to say, the measure of success of the models developed here has generally been visual appearance rather than, for instance, scientific veracity. Thus the emphasis has been more on the creation of realistic-looking pictures, than on the construction of mathematically or physically correct models (the rainbow, mirage and erosion models being exceptions). Many of the models may seem *ad hoc*, but there is an underlying theme: they have been required to be (at least somewhat) elegant. That is, they are algorithmically minimal, implemented in a relatively small amount of C code, and reasonably efficient. As each subroutine in a renderer is typically called on the order of millions of times per image, efficiency is a primary concern in the discipline of image synthesis. Efficiency is often at odds with elegance; we have generally required both in our algorithms. Elegance in our algorithms is often attributable to their fractal nature, which leads to what Smith [140] has termed "database amplification": enormous geometric and visual complexity is abstracted into terse procedural descriptions. The fractional Brownian motion which is the basis of most of our visual complexity is an outstanding example of this: the code need simply describe the construction of a spectral sum with a $1/f^{\beta}$ power spectrum for a given exponent β.

There can be seen to be two polar extremes in approaches to modelling: the visually driven "if it looks good, it *is* good" approach, which we refer as *ontogenetic* modelling, and the more rigorous approach of developing computational implementations of mathematical models taken from the scientific literature, which in its most extreme form is sometimes referred to as *teleological* modelling [7]. Early work in the (still-young) field of computer graphics was primarily of the first sort; as the field matures and ability to readily produce reasonably good-looking images is more firmly established, the emphasis is shifting towards the second approach, as exemplified by "physically based" modelling. The first approach emphasizes efficiency; the second veracity. Not

surprisingly, the first approach generally leads to better-looking pictures more quickly, while the second leads to impressive-looking technical papers for publication and an increased understanding of the "right" way to model (as well as, often enough, experience in how *not* to tackle modelling problems).

Physical models can entrain great elegance and descriptive power: witness the spectacular global illumination effects afforded by the semi-physical ray tracing model, based on the geometric optics associated with the particle model of photons. They can also create computational nightmares: consider image synthesis based on the wave model of photons. [90] The radiative equilibrium models of *radiosity* exemplify a middle ground, where complexity of the implementation is moderate; computation time is generally significant; and the results striking and highly desirable, yet often suffer significant visual flaws due to simplifying assumptions. As the ultimate goal of realistic image synthesis is to have the right picture for the right reasons, both approaches to modelling are justified and should be pursued. As the field matures, the twain shall meet; in the meantime, work from both ends of the spectrum is justified. In the author's view, the key is intellectual honesty: when one is working from the *ad-hoc*, image-driven or ontogenetic approach, it behooves us to be clear about what is and is not modelled accurately and with veracity, and to what extent.

The work presented in this dissertation is perhaps best viewed as breadth-first research. The area of realistic imaging of synthetic landscapes was quite immature at the outset of this work. It is a broad field, as the variety of topics addressed here indicates. Thus the research presented here comprises a quantity of relatively small results in disparate areas, united under the umbrella of a common goal: improved landscape imagery. It is not a depth-first inquiry into a difficult and picayune problem in computer science, couched in the context of previous work on similar problems. Rather it

represents a survey of preliminary approaches to a variety of problems which shall continue command the attention of researchers for some time to come.

Most of our technical results were motivated by artistic needs; the rest by recognition of easy opportunity. The majority of the author's time has been spent on refining the artistic process, i.e., developing the procedural approach to realistic image synthesis of models of natural phenomena, and in mastering the medium, i.e., attaining the goal of superior craftsmanship by gaining control of the algorithmic processes, of devices for high-resolution display, and of the nonlinear transformations in color and contrast involved in color reproduction. (Unfortunately, most of the work in color reproduction simply adds up to practical experience, and does not constitute publishable research. [104]) A significant amount of time and energy has been expended on the highly subjective, aesthetic tasks of visual composition and color usage; the reward of this has been effective artistic self-expression. The net result of our image-driven approach to research in computer graphics has been the creation of images of aesthetic significance which simultaneously serve as illustrations of original image synthesis techniques and the descriptive power of fractal geometry, and as examples of artworks born of a novel creative process.

The primary purpose of this work has been to illustrate and expand the descriptive power of fractal geometry as a visual language of natural form. More specifically, it has focused on expanding and refining the uses of fractional Brownian motion as the basis for models of natural phenomena, and clarifying its limitations (e.g., in terrain and turbulence models). Significant progress has been made in this, and directions for future work are clear.

1.4 Contents and Organization of this Dissertation

This section presents a brief overview of the contents of this document. Most of the contents of this dissertation have already appeared in print; rather than specifically name the publications associated with each section, they will appear as bibliographic references.

Chapter 1, Introduction, consists of a statement of the thesis of this dissertation, a section on motivation for the work, a manifesto re the methodology pursued, and this section describing the contents. Section 1.1, The Thesis, attempts to specify clearly the conceptual cohesion of the many seemingly disparate results presented herein. [91,92,107,108] Section 1.2, Motivation, is a detailed description of the problems and opportunities for solutions, seen by the author in the course of this work. [96,107] The section is rather lengthy, as it is presumed that the reader has little familiarity with the concerns addressed, and their context in the discipline of computer graphics. Section 1.3, Manifesto, is essentially a statement of method and a justification therefor. [93] There are idiosyncrasies in the approach taken in this research which are clarified and put into context in this section. Section 1.4, Contents and Organization of this Dissertation, is this overview of the contents.

Chapter 2, Terrain Synthesis, addresses our work on fractal terrain models. [100,105,106]

Chapter 3, "Grid Tracing" for Rendering Height Fields, describes our DDA-based spatial subdivision scheme for efficient ray tracing of terrain models. [94,106]

Chapter 4, Atmospheric Scattering Models, describes the various models of atmospheric effects developed in the course of this research. [95,98,100]

Chapter 5, Procedural Textures, describes a small part of our work with procedural textures. Section 5.1 clarifies and motivates the procedural approach. [91,95,99,100,105]

The following sections illuminate the process of developing procedural textures, by presenting some procedural models of planets. [91,99,100,105]

Chapter 6, Conclusions, presents our conclusions and directions for future work. [68]

Appendix A, Parallel Computation Strategies, documents our use of C-Linda [16] in parallel ray tracing and erosion simulation. [91,92,94,96,100,103,105-108]

Appendix B, A Panoramic Virtual Screen for Ray Tracing, describes a method for mapping the entire celestial sphere onto the image plane, in image synthesis. [97] This represents part of the work done by the author while at the Visualization for Planetary Exploration Laboratory at NASA Ames, in the summer of 1991.

Appendix C, Essay: Formal Logic and Self-Expression, elaborates on the thesis that this work represents a significant event in the history of the creative process for the fine arts. [107] The text was developed, in part, for invited lectures at the Art and Mathematics Conference in Albany, New York, in June of 1992, and the Third International Symposium on Electronic Arts in Sydney, Australia, in November of 1992. [93] It consists of a draft essay, still under development, which is to be submitted for the proceedings the Fourth Symposium on Electronic Arts in Minneapolis, Minnesota in November of 1993. This appendix and the next, are included to establish a broader context for this work.

Appendix D, Mathematics: The Language of Nature, reproduces an essay written for the inside cover of the 1993 Fractals Calendar. [92]

Appendix E, Color Plates, contains captions for the various color illustrations used to illuminate points made throughout the text, and the plates themselves.

Appendix F, Curriculum Vitae, consists of the author's current curriculum vitae.

2. Chapter 2: Terrain Modelling

Chapter Abstract

This chapter covers our work in improving geometric models of fractal terrains. Our fundamental contributions to the area of terrain modelling have been the development of point-evaluated heterogeneous terrain models, and erosion processes to be applied to terrain models to generate drainage network features, fluvial deposits, and talus slopes. The heterogeneous terrain models expand the repertoire of terrains which may be described with fractal models, but have no more basis in physics than do conventional, homogeneous fractional Brownian motion terrain models. The erosion models, on the other hand, are derived from the scientific literature of fluvial morphology and represent physical simulations of natural processes.

Standard fractal terrain models based on fractional Brownian motion (fBm) lack realism partly because the statistical character of the surface is the same everywhere, i.e., it is a homogeneous, or *stationary*, stochastic function (or at least, it should be [76]). To expand the vocabulary of the fractal language of descriptions of Nature, we present a new approach to the synthesis of fractal terrain height fields. In contrast to previous techniques, this method features locally independent control of the spectral sum comprising the fBm, and thus local control of fractal dimension, lacunarity, and crossover scales. *Noise synthesis* [106] or *rescale and add* [134] achieves this flexibility by being

point-evaluated, or context-free. This distinguishes it from Fourier synthesis and polygon-subdivision schemes, which may not be so flexible.

In noise synthesis, modifications are made to the inner loop of the frequency summation in the construction of the fractal function, which remains a variation on fBm. The modifications are sufficiently minor that most of the elegance of the native fBm model is retained. Point-evaluation allows the parameterized modifications to vary locally, without reference to values of neighboring points, hence the context-independence. Varying the statistics of the terrain model with altitude or lateral position yields more realistic first approximations to eroded landscapes. A slightly more subtle manipulation of the fBm construction loop is shown to yield a model of terrain on large scales, e.g., across tens or hundreds of kilometers.

Preliminary physical erosion models are outlined which simulate fluvial, thermal, and diffusive erosion processes to create global stream/valley networks, talus slopes, and rounding of terrain features. At the time of this writing, the erosion models suffer from numerical instabilities in the finite difference scheme used to model fluid transport. Thus they have not yet been used in their foreseen application as computational testbeds for experimental verification of the published formal transport models of fluvial geomorphology. Work is currently underway to fix the transport problems, so that such research may commence.

2.1 Introduction

Our contributions in the area of fractal terrain models spring from four basic observations of fBm based models: 1) Existing fBm schemes are flawed. These flaws range from nonstationary statistics [76] or the creasing problem [78] in polygon subdivision schemes to periodicity in Fourier-synthesis schemes. [152] 2) The small sum of frequencies used in synthesizing fBm for computer graphics purposes, allows the

character of the basis function to show through clearly. The basis function is usually implicit in the generation algorithm: a saw-tooth wave in polygon subdivision schemes, a sine wave in Fourier synthesis, a piecewise tri-cubic polynomial in noise synthesis, etc. Varying the basis function affects the character of the final surface. 3) Pure fBm is insufficient as a model of natural terrains. FBm is statistically symmetric across the horizontal plane; real terrains are not. [78] FBm is homogeneous (i.e., stationary) and isotropic; real terrains are not. 4) Terrain models based on fBm lack river networks, deltas, alluvial deposits, talus slopes and other such erosional features which are salient in natural terrains. (This may be seen as part of observation 3, but in practice it raises a different set of problems.)

Convincing fractal drainage systems are difficult to arrange in the first-order algorithmic generation of the surface, specifically because they result from global processes and thereby require global communication in the generation algorithm. In nature, drainage systems are due not to the primary orogenesis (mountain-building process) itself, but to "post-processing" of the uplifted substrate by flowing water and glaciers. This indicates the efficacy of generating them by post-processing in synthetic models as well.

2.1.1. Origins of Fractal Terrain Models

The origin of fractal landscapes in computer graphics is this: some time ago, Benoit Mandelbrot noticed the similarity between the trace of fractional Brownian motion over time, and the skyline of jagged mountain peaks. [77] He reasoned that this process, extended to two dimensions, would result in a "Brownian surface" which would provide a visual approximation to mountains in nature. Some of Mandelbrot's earliest computer graphics images were of such surfaces [79] and indeed, they do capture the essence of "mountain-ness". Voss and Mandelbrot later used fractional Brownian noises to create

some very convincing forgeries of nature. [155] New terrain synthesis methods have since been proposed by Fournier et al [34], Miller [86], and Lewis. [73]

Most synthetic fractal terrains hitherto have been varieties of fractional Brownian motion, or, more loosely, $1/f^{\beta}$ surfaces. FBm is simply a mathematical phenomena which happens to resemble (very mountainous) natural terrains. There is no known, generative link between the shape of fBm and that of real landscapes. While this is not unusual in fractal descriptions of natural phenomena, it sometimes leads to questions about such descriptions, as noted by Sommerer [142]:

> Although physical fractal spatial patterns are frequently observed, a quantitative connection between the measured numerical value of the fractal dimension and underlying physics of the process has largely been lacking (exceptions are [20,47,136,145]). This lack of connection, while not reducing the utility of fractals for phenomenological characterization, has led to skepticism about the ultimate meaningfulness of fractal descriptions in physics. [51,128,129]

Fortunately, our discipline is image synthesis, not physics, and therefore good ontogenetic descriptions are often more useful to us than thorough teleological ones. Furthermore, lack of knowledge of linkage does not necessarily imply that there exists no such link; it may simply be as-yet undiscovered.

2.1.2. Limitations of FBm Terrain Models

The root of the problem with fBm surfaces as models of natural landscapes traces back to the fact that fBm was chosen for use as such a model, for purely ontogenetic reasons: fBm *looks* like mountains. But native fBm is a mathematical function designed to incorporate certain properties. Chief among these is stationarity: fBm is a stochastic function which is statistically invariant under translation. Real terrains decidedly are not. FBm terrains are statistically symmetrical about the horizontal plane due to the symmetry of the Gaussian distribution of displacements inherent in their definition. Again, real terrains are not. FBm terrains in general have no global erosion features, in part, because fBm is by design isotropic and stationary. Such homogeneity cannot encompass the

morphology of stream beds. In practice, fractal terrain models lack drainage features because of the difficulties inherent in implementation and computation of models of such global structures, which require global communication and context-sensitivity. Free of concerns with computational efficiency, real terrains generate such structures spontaneously, reliably, and ubiquitously.

In this chapter, we describe a flexible approach to the generation of variable smoothness and asymmetry in fractal terrain models in the terrain model generation stage, and suggest a global, physical erosion post-processing process for height fields which generates both local and global erosion features through a simulation of natural erosion processes.

2.2 Previous Work

We now review the prior state of the art in fractal terrain synthesis.

2.2.1. Fractional Brownian Motion

This section describes fBm and methods used for its generation in computer graphics. In an attempt to convey to the reader an intuitive grasp of fBm, we first offer several high-level views of fBm. For a thoroughly detailed presentation, we refer the reader to the excellent existing literature, particularly Voss [152] and Saupe. [133]

2.2.1.1. Characterization of fBm

FBm is a stochastic fractal function, which is characterized by its statistical behavior. Here are several ways of viewing it:

- *Definition:* FBm is the integral over time of the increments of the path of Gaussian Brownian motion, i.e., a random walk.

- FBm is a non-differentiable (read: infinitely wiggly) function of time, with a characteristic power spectrum (e.g., graph of signal amplitude by frequency): the power spectrum corresponds to the function $1/f^{\beta}$, where f is frequency, and $\beta \in [1,3]$.

- FBm is statistically self-affine: its statistics (and appearance) are similar at all scales, modulo a vertical scaling factor (this need for a different change in vertical scale, with horizontal scaling, distinguishes self-affinity from self-similarity).

- FBm as a terrain model, is a two-dimensional extension of the (originally) one-dimensional fBm formalization.

- (Fourier synthesis model:) fBm is a sum of sine waves with random phase, and amplitude scaled as $1/f^{\beta}$.

- (Noise synthesis model:) fBm is a sum of band-limited random basis functions, with amplitude scaled as $1/f^{\beta}$, where f is the mean frequency of the band-limited basis function.

- (Midpoint displacement model:) fBm is a sum of sawtooth waves of successively doubled frequencies, with Gaussian offsets for peaks, scaled by frequency f as $1/f^{\beta}$.

Fractional Brownian motion is perhaps most succinctly described by the Weierstrass-Mandelbrot function [152]:

$$V(t) = \sum_{f=-\infty}^{\infty} A_f r^{fH} \sin(2\pi r^{-f} t + \theta_f) \tag{2.2.1}$$

where A is a Gaussian random variable, r is the spatial resolution or *lacunarity*, θ is a uniform random viable in the range $[0,2\pi]$ providing a random phase, and H is the Hölder exponent, which determines the fractal dimension. The Weierstrass-Mandelbrot function is basically an infinite sum of sine waves at discrete frequencies and with

random phase, with a certain size gap between successive frequencies (the lacunarity), and an exponent *H* which scales amplitude by frequency. Amplitude relates to frequency *f* by the plot of the power spectrum [152]

$$S_{WM} \propto \frac{1}{f^{H+1}} \tag{2.2.2}$$

and hence the result is referred to as $1/f$ *noise*.

The important point to note here is that pure fBm is composed of a sum of sine waves of discrete frequencies, with random phases, and amplitudes related to frequency by the $1/f^{\beta}$ weighting. Thus fBm for computer graphics purposes is generally constructed by summing discrete frequencies of some basis function, with the proper amplitude scaling for each frequency.

2.2.1.2. Band-Limiting

Note that the Weierstrass-Mandelbrot function involves an infinite sum, generally a bad idea for computer programs which are expected to terminate. It also contains an unbounded range of frequencies while, as prescribed by the Nyquist sampling theorem, digital displays have an upper limit to the spatial frequencies they can accurately reproduce. Thus, fBm for computer graphics is always *discretized band-limited fBm*, that is, fBm composed of a finite sum of discrete frequencies.

What does this mean, in practice? It means that we generally construct fBm as a sum of about three to eight frequencies of the basis function -- a relatively simple, fast operation for the computer. Note that we state a lower limit of three octaves of the basis function for fBm; with less than three octaves any claim of self-similarity or adherence to a particular characteristic power spectrum becomes rather vacuous.

2.2.1.3. Generation of fBm

FBm generation methods can be categorized, for the purposes of computer graphics, into procedural and non-procedural methods. Procedural models are evaluated when and where needed, at rendering time, whereas non-procedural models are evaluated globally and stored in advance of rendering. Non-procedural methods, such as polygon subdivision, are generally fast and easy to implement. Procedural methods, specifically noise synthesis, [106] are flexible, memory-efficient (as only visible parts of the model need be computed), and simple, given an implementation of the basis (i.e., "noise") function. They are not particularly efficient, in terms of computing time, as the noise function generally requires many floating point operations to evaluate. Nevertheless, due to the flexibility and elegance of the procedural approach, noise synthesis remains the author's fBm generation method of choice.

For background on procedural means for generating fBm and other stochastic procedural textures, see Gardner [39], Lewis [72,73], Mandelbrot et al [74], Musgrave et al [105,106], Perlin [119], Saupe [134], Stam et al [146], and van Wijk. [160]

2.2.1.4. Terminology

We now give a very brief description of some the mathematical terminology associated with the generation of fractal terrains. For greater depth, see Saupe. [133]

We define D_f as the *fractal dimension* of the surface, D_E as the *Euclidean dimension* of the surface, and H as the *fractal dimension parameter*. (Note that previous authors specifically, Bouville [15] and Miller, [86] sometimes erroneously refer to H as the fractal dimension.) For terrain models $D_E = 2$ and $D_f = 3 - H\ D_E$.

The fractal dimension D_f, Euclidean dimension D_E, fractal dimension parameter H, and *spectral exponent* β of $1/f^{\beta}$ noise or fBm are related as:

$$D_f = D_E + 1 - H = D_E + \frac{3-\beta}{2} \tag{2.2.3}$$

It follows that $\beta = 1 + 2H$ and $H = \frac{\beta - 1}{2}$. Since $D_E = 2$, for our purposes, $D_f = 3 - H = \frac{7-\beta}{2}$. Note that H is in the interval $[0,1]$ and β is in the interval $[1,3]$ for fBm, by convention. Outside this range, the function may not formally be fBm.

Fractional Brownian motion in one dimension is a stochastic process $X(t)$ with a power spectrum $S(f)$ scaling with f as

$$S(f) \propto \frac{1}{f^{\beta}} \tag{2.2.4}$$

where β is referred to as the *spectral exponent* of the fBm and is in the interval $[1,3]$.

FBm itself is not a stationary process, but its increments $I(\Delta t) = X(t + \Delta t) - X(t)$ are; that is, the expected value of $I(t, \Delta t)$ is zero for all t and Δt and the variance σ^2 of $I(t, \Delta t)$ does not depend on t. In the special case of Brownian motion, $H = 0.5$ and σ^2 varies as Δt^{2H}. Thus for $H = 0.5$ increments are uncorrelated; for $H > 0.5$ (as in fractal terrains, where H is approximately equal to 0.8) increments are positively correlated; for $H < 0.5$ they are negatively correlated (corresponding to a *very* rough surface). In more than one dimension fBm is a *random field* $X(x, y, \ldots)$ with X on any straight line being a $1/f^{\beta}$ noise.

Crossover scale is the scale at which fractal character vanishes. *Upper* crossover scale is heuristically defined as the scale where vertical and horizontal displacements are equal. Thus, for a mountain range rising from sea level to peaks which are at most one kilometer high, the upper crossover scale is one kilometer. *Lower* crossover scale would be the smaller scale at which the surface becomes smooth and non-fractal.

Lacunarity generally refers to gaps in fractals [77]; here it can be thought of as the gap between frequencies composing the fBm of the fractal terrain. Thus when iteratively

summing the frequencies composing fBm, if the frequency f_i added at stage i is a multiple λ of f_{i-1},

$$f_i = \lambda f_{i-1} \tag{2.3.5}$$

then λ is the lacunarity of the fBm. While lacunarity affects the texture of the fBm, this effect is usually only noticeable for $\lambda > 2$. Thus we usually let $\lambda = 2$, as lower values involve more computation for a given frequency range of fBm and larger values can affect the surface appearance.

2.2.2. Fractal Terrain Models

Most fractal terrain models have been based on one of five approaches: Poisson faulting [77,155], Fourier filtering [77,82,155], midpoint displacement [34,73,78,86], successive random additions [155], and summing band-limited noises. [37,86,134] The approach we have developed is of the last type, which we will refer to as the *noise synthesis* method. We will now briefly review these five techniques (for a more detailed review, see Voss [152] and Saupe [133]).

2.2.2.1. Poisson Faulting

The original terrain generation technique employed by Mandelbrot [77] and Voss [155] was Poisson faulting. The Poisson faulting technique involves applying Gaussian random displacements (faults, or step functions) to a plane or sphere at Poisson distributed intervals. The net result is a Brownian surface. This approach has been employed to create fractal planets by Mandelbrot and Voss. [77] It has the advantage of being suitable for use on spheres for creation of planets. Its primary drawback is the $O(n^3)$ time complexity.

2.2.2.2. Midpoint Displacement

Midpoint displacement methods were introduced to the computer graphics community as an efficient terrain generation technique by Fournier, Fussell, and Carpenter. [34] We classify the various midpoint displacement techniques by locality of reference: *wireframe* midpoint displacement, *tile* midpoint displacement, *generalized stochastic subdivision,* and *unnested* * subdivision.

Pure wireframe subdivision is used only in the popular triangle subdivision scheme [34] and involves the interpolation between two points in the subdivision process. Tile midpoint displacement involves the interpolation of three or more non-collinear points; it is used in the "diamond-square" scheme of Miller [86], the square scheme of Fournier et al [34], and the hexagon subdivision of Mandelbrot and Musgrave. [78] Generalized stochastic subdivision [73] interpolates several local points, constrained by an autocorrelation function. Miller [86] also proposed an unnested "square-square" subdivision scheme.

Wireframe and tile midpoint displacement methods are generally efficient and easy to implement, but have fixed lacunarity and are nonstationary due to nesting (see Miller [86] for a disposition of the resulting artifacts). Generalized stochastic subdivision and unnested subdivision schemes are stationary; the former is flexible but not particularly easy to implement, while the latter features fixed lacunarity and is very simple to implement. Note that all midpoint displacement techniques produce true fractal surfaces [108] but simply have the wrong statistical characteristics to qualify as pure fractional Brownian motion. [76]

---

* For an explanation of the "nesting" issue, see Mandelbrot. [78]

2.2.2.3. Successive Random Additions

Successive random additions is a flexible unnested subdivision scheme. When points determined in previous stages of subdivision are re-used, they are first displaced by addition of a random variable with an appropriate distribution. Previous points need not be re-used; new grid points to be displaced can be determined from the previous level of subdivision by linear or nonlinear interpolation. Successive random additions features continuously variable level of detail, which is useful for zooms in animation, and arbitrary lacunarity. The lacunarity λ depends on the change of resolution at successive generations; time complexity of the algorithm is a function of λ and the final resolution R. The successive random additions algorithm is easy to implement.

2.2.2.4. Fourier Synthesis

Fourier synthesis generates fBm by taking the Fourier transform of a two dimensional Gaussian *white noise*, then multiplying it in frequency space with an appropriate filter, and interpreting the inverse Fourier transform of the product as a height field. Alternatively, one can simply choose the coefficients of the discrete Fourier transform, subject to the proper constraints, and interpret the inverse Fourier transform as above. [133] Advantages of this approach include the availability of arbitrary lacunarity and precise control of global frequency content. Disadvantages include periodicity of the final surface, which can require that substantial portions of the computed height field patch be discarded, the $O(n \log n)$ time complexity of the FFT algorithm, the level of complexity of implementation, and lack of local control of detail.

2.2.2.5. Heterogeneous Models

The issue of statistical symmetry across the horizontal plane in fractal terrain models has been addressed by Mandelbrot and Voss [77,155] through the use of nonlinear scaling in a post-processing step, and by Mandelbrot [78] through the use of random

variables with non-Gaussian distributions in the displacement process. These approaches yield peaks which are more jagged and valleys which are smoother, but they still lack global erosion features. A global river system, created algorithmically at terrain generation time, has been demonstrated by Mandelbrot and Musgrave [78] but with less-than-satisfactory results, e.g., the resulting surface is far too regular to appear convincingly natural (see Plate 2.1).

2.2.2.6. Summing Band-Limited Basis Functions

What we call *noise synthesis* can be described as the iterative addition of tightly band limited frequencies, each of which has a randomly varying, or "noisy", amplitude. Noise synthetic surfaces have been used by Miller, [86] Gardner [37] and Saupe. [134] Miller has used Perlin's procedural $1/f^{\beta}$ noise [87] as a *displacement map* [22] to add detail to the (otherwise straight) edges of polygons tessellating a Brownian surface of similar spectral content. Gardner has interpreted his noise function, based on a "poor man's Fourier series" [39] (actually a variation of the Mandelbrot-Weierstrass function) as a height field. The quantization of altitude values of the height field yields terraced land, such as mesas. Our approach differs from Gardner's in that we exercise local control over frequency content based on the amplitude of existing signal and other functions. The Perlin noise function is notably more isotropic than Gardner's noise function, and is not periodic; Gardner's terrains and textures suffer visible artifacts due to these factors. In addition, Gardner's noise function requires some critical values of the constants for good results, which values must be determined through subjective experimentation. Driven by table lookups, the Gardner noise function is much faster than the Perlin function.

Saupe independently developed an approach to noise synthesis similar to ours; he terms it *rescale and add*. Saupe's publication of that work featured an emphasis on mathematical foundations, while ours have emphasized applications. For a thorough

mathematical treatment of the issues of noise synthesis which is complimentary to this document, see Saupe. [134]

2.2.3. Erosion Models

Kelley et al [61] have used empirical hydrology data to derive a system for the generation of stream network drainage patterns, which are subsequently used to determine the topography of a terrain surface. This approach features the global dependence necessary for realistic fluvial erosion patterns, and has a strong basis in measurements of real physical systems. This approach to modelling fluvial erosion is relatively efficient; what it lacks is the detail of a fractal surface. While the stream network may be fractal, the "surface under tension" used for the terrain surface is not, and cannot be readily made so without disturbing the drainage basins and stream paths.

We propose a simple fluvial erosion simulation in which water is dropped on each vertex in a fractal height field and allowed to run off the landscape, eroding and depositing material at different locations as a function of energy and sediment load of water passing over each vertex. The erosion laws are taken from the literature of fluvial geomorphology. [3,135] The model features the global communication necessary to create global features, and is slow despite the $O(n)$ time complexity. We also present a global model for simulation of what we refer to as *thermal weathering*. While fluvial erosion creates valleys and drainage networks, thermal weathering wears down steep slopes and creates *talus slopes* at their feet. The thermal weathering simulation can create realistic results in much less computing time than the fluvial erosion simulation, and is also $O(n)$ in time complexity. Diffusive erosion simulation is trivial; it simply consists of a progressive low-pass spatial filter over time. These models are discussed in section 2.4.

2.3 Original Terrain Synthesis Models

We now present our original models for fractal terrain synthesis. First we describe our basis function, the so-called "noise" function, then we describe how we use it to generate novel terrain models.

2.3.1. Noise Function

Noise-synthetic terrain generation is accomplished by the addition of successive frequencies of a band-limited "noise" function. The source of the noise we use is a version of the Perlin [119] noise function. The ideal noise function for our purposes would be *monochromatic* (i.e., single-frequency), *homogeneous* (invariant under translation), and *isotropic* (invariant under rotation). The Perlin function supplies a band-limited signal of random amplitude variation; it is stationary and nearly isotropic.*

The Perlin noise function $N: R^n \rightarrow R$ is implemented as a set of random gradient values defined at integer points of a lattice or grid in space (of dimension $n = 1, 2, 3,$ or 4) which are interpolated by a cubic function. At lattice points in space (points in space with integer coordinates), the value of the function is zero (a *zero crossing*) and its rate of change is the gradient value associated with that lattice point. The trajectory of the function, given by the gradient value at the integer points, is interpolated at non-integer points with the cubic function $y = 3x^2 - 2x^3$, which interpolant features second derivative continuity and zero rate of change at the end points, where $x = 0$ and $x = 1$ (see Figure 2.1). Since the gradient might be, for instance, increasing at two consecutive lattice

---

* It is geometrically impossible to reconcile the three criteria of monochromaticity, stationarity, and isotropism in a multidimensional Perlin noise function. If the frequency of an n dimensional Perlin function is f along the axes of the grid upon which it is defined, then the frequency will be $\sqrt{n}f$ along the long diagonal of that grid.

points i and $i+1$, there may also be a zero crossing between lattice points (see Figure 2.2). This gives rise to frequencies in the noise function higher than f, f being the primary frequency which is that of the spacing of the integer lattice.

Figure 2.1. The cubic interpolant $y = 3x^2 - 2x^3$ of the noise function.

Figure 2.2. The trace of the noise function.

The noise function can be modified to have an arbitrary, non-zero value at the lattice points. This increases the variance of the function, but adds low frequency components to the signal which cannot be controlled or subsequently removed [73]; this has implications for the statistics of the surfaces to be generated. For an analysis of the spectral characteristics of such a noise function see Saupe. [134]

The band-limited Perlin noise function $N(\vec{p})$ outputs a signal with a fixed lower frequency f equal to half the frequency of integers in the domain $\vec{p} \in R^n$. To scale the frequency of N by a factor u, one simply performs a scalar multiplication $u\vec{p}$ of the domain vectors supplied to N. This has the effect of scaling the reference points in the

noise lattice, producing the desired frequency shift in the output of N. We will see this practice used below.

2.3.2. Local Control of Statistics

We now begin to describe how we have employed the flexibility of procedural methods to construct terrain models of improved realism.

Subjective observation of natural landscapes reveals that in certain types of mountain ranges there is a marked change in the statistics of the surface as one moves from the foothills to the highest peaks. The foothills are more rounded, while the higher mountains are more jagged. Sometimes, as in the eastern slope of the Sierra Nevada, the entire mountain range rises in a relatively short distance from a nearly flat plain. This change of character can be characterized as a change of fractal dimension D_f, crossover scale, or both.

Using the noise synthesis technique we can easily devise terrain models with such features, by modulating the power spectrum of the surface as a function of horizontal position and/or vertical altitude, or of prior values in the spectral sum. These techniques produce heterogeneous terrain models. Note that we might also refer to such models as nonstationary, but as history has imbued that term with a pejorative cast in fractal terrain modelling [76], we prefer to use the term "heterogeneous".

We will begin our exposition with a description of a procedural construction of homogeneous fBm. We will then describe our modifications to this algorithm, for generating heterogeneous terrains.

2.3.2.1. Homogeneous Procedural FBm Construction

Homogeneous fBm is simply a stochastic function with a $1/f^{\beta}$ power spectrum, for $\beta \in [1, 3]$. In a procedural context it may be described as:

$$fBm(\vec{p}) = \sum_{i=1}^{n} N(\vec{f}_i)\varpi^i$$

where $\vec{p}$ is the point at which the function is evaluated, n is typically between 3 and 12, N is the basis (e.g., "noise") function, $\vec{f}_i = \vec{p}\lambda^i$ (usu. lacunarity λ=2.0), and $\varpi = \lambda^{-0.5\beta}$ (a constant, determining the fractal dimension D_f). The randomness in the function is embodied in the noise function N.

Below is pseudo-code for Noise-based fBm:

```
fBm( p, H, r, n )
        result := Noise( p )
        for octave := 2 to n
                p := p * r
                amplitude := pow( r, -H * octave )
                result := result + amplitude * Noise( p )
        return ( result )
```

where H is the fractal dimension parameter, r is the lacunarity (usu. equal to 2.0), and n is the number of octaves. Note how terse the fBm specification is.

C code to generate such fBm might look like:

```
double fBm( point, spectral_exp, lacunarity, octaves );
      Vector      point;
      double      spectral_exp, lacunarity, octaves;
{
      register double, i, result, amplitude, frequency=1.0;

      result = Noise3( point );
      for( i=octaves-1; i>0.0; i-- ){
            point.x *= lacunarity;
            point.y *= lacunarity;
            point.z *= lacunarity;
            frequency *= lacunarity;
            amplitude = pow( frequency, -spectral_exp );
            if ( i < 1.0 ) amplitude *= i; /* for octaves remainder */
            result += amplitude * Noise3( point );
            if (amplitude < VERY_SMALL) break;
      }
      return( result );
} /* fBm() */
```

where `point` is the point at which the function is being evaluated; `omega`, the fractal dimension parameter, is typically ~0.5; the lacunarity `lambda` is almost always 2.0 (in fact, this might as well be hard-coded); and `octaves` determines the number of

frequencies in the spectral sum (note that the variable name "`octaves`" is only appropriate when `lambda` equals 2.0, as "octave" implies a frequency doubling). This fBm function rolls off the remainder of the floating point value of `octaves`, for smooth transitions in adaptively band-limited textures.

Note that this function should not be regarded as statistically-accurate, as Saupe has shown [134] that the cubic polynomial interpolant employed in the noise functions affects the power spectrum of the final discretized fBm sum in unexpected ways. The power spectrum of Perlin's "`Turbulence()`" function [119] is, for instance, $1/f^3$ rather than the expected $1/f^2$. Therefore, we should regard this and the other versions of fBm functions described here as parameterized versions of random fractal functions, the parameter values of which are to be interpreted subjectively rather than empirically. Formal correspondence between parameter values and fractal statistics could be established using the results described by Saupe, but we have not found this to be necessary or useful for successful computer graphics practice.

2.3.2.2. Local Modulation of Fractal Dimension

In our procedural approach, fractal dimension can be modulated locally by varying β with position. This is a flexibility previously unavailable in fBm generation schemes. Plate 2.2 shows a patch which is planar on the left, to space filling on the right (modulo the upper and lower crossover scales, which are approximately 7 and 1/128, respectively). In this case, we have $\beta = x^{-1}$ (corresponding to $H = \frac{1}{x} - \frac{3}{2}$), and x in the interval (0,1].

In Plate 2.3, we linearly change fractal dimension D_f from 2 ($H = 1,\ \beta = 3$) to 3 ($H = 0,\ \beta = 1$) on the right. Note that this is not the same as going from planar ($1/f^{\infty}$) to filling all of 3 space ($1/f^0$), as in Plate 2.2.

2.3.2.3. Fractal Statistics by Altitude

We now begin to describe our heterogeneous procedural terrain models. Our first model varies the spectral exponent in the spectral summation by the magnitude of the contribution of previous, lower frequencies. The idea is to keep the terrain smooth near an artificial "sea level", while allowing it to get rougher at higher altitudes, in a first approximation to naturally occurring, eroded terrains.

2.3.2.3.1. The Algorithm

Presuming that our basis (e.g., noise) function $N: R^n \rightarrow R$ has a range in the interval $[-1,1]$, we may wish to translate N by a constant c_t so that it is, for instance, always or nearly always positive (the sign will be important in later iterations). We may also wish to scale N by a factor c_s to reduce or expand its range (the positive portion of which we may wish to keep normalized to a maximum value of 1, for instance). In the patch illustrated in Plate 2.4, we insert the lowest frequency first:

$$A_0 = \left(N\left(\vec{f}_0 \right) + c_t \right) c_s + c_0$$

where A_0 is the initial height of a point in the height field, $\vec{f}_0$ is the initial object space coordinate vector, $\vec{p}_0$, of the height field position being calculated, and c_0 is an offset constant which determines the zero value or "sea level" of the terrain. Note that $\vec{f}_0$ can be an element of R^2 or R^3, depending on the arity of the noise function domain.

Iterating fBm noise at lacunarity $\lambda > 1$ requires that, at iteration n, the frequency added is proportional to $\left(f_0 \lambda^n \right)^{-0.5\beta}$. Setting the lowest frequency $f_0 = 1$ gives a frequency increment at iteration n of $\lambda^{-0.5\beta n}$. Thus we have for the altitude A_i at stage $i > 0$:

$$A_i = A_{i-1} + A_{i-1}\left(N\left(\vec{f}_i \right) + c_t \right) c_s \varpi^i$$

where $\varpi = \lambda^{-0.5\beta}$ (a constant) and $\vec{f}_i = \vec{p}_{i-1}\lambda$. Note that for a noise function $N: R^2 \rightarrow R$, we have $\vec{f}_i = \vec{p}_0\lambda^i$. For $N: R^3 \rightarrow R$, we may have $\vec{f}_i \neq \vec{p}_0\lambda^i$ due to vertical displacement.

To clarify, here is pseudo-code for the algorithm. Note that it is only a minor modification to the fBm routine described above.

```
hfBm( p, H, lacunarity, octaves, sea_level )
        result := Noise( p ) + sea_level        /* Noise: R^3 → [-1,1] */
        for octave := 2 to octaves
                p := p * lacunarity       /* lacunarity usu. ≅ 2.0 */
                amplitude := pow( lacunarity, -H * octave )
                result := result + min( 1.0, result ) + amplitude * Noise( p )
        return ( result )
```

Below is C code implementing this algorithm. The point is not to show the details of the code, but rather its brevity.

```
      /* heterogeneous fractional Brownian motion routine */
double
hfBm( point, omega, octaves )
      Vector      point;
      double      omega, octaves;
{
      register double   lacunarity, a, o, prev;
      register int      i;
      register Vector   tp;

      lacunarity = 2.0;  a = 0.0;  o = omega; tp = point;
            /* get initial value */
      a = prev = 0.7 + Noise3( point );
      for( i=1; i<octaves; i++ ) {
            tp.x *= lacunarity; tp.y *= lacunarity; tp.z *= lacunarity;
                  /* get subsequent values, weighted by previous value */
            prev = prev * o * ( 0.7 + Noise3(tp) ) ;
            a += prev;
            if (prev < SMALL) break;
            o *= omega;
      } /* for */

      return( 0.15/omega * a );
} /* hfBm() */
```

2.3.2.3.2. Discussion

This algorithm does not use a uniform spectral exponent β for all frequencies: it is a monotonically decreasing function, as frequency increases, bounded above by the "base" fractal dimension which is the highest dimension attainable in the terrain. This amounts to modulating both lower crossover scale and fractal dimension with altitude. Yet it is

not even as simple as that: the spectral exponent, which determines the fractal dimension, varies with frequency, since altitude also varies with frequency as the spectral summation proceeds. We have not yet elegantly characterized the complex statistical behavior of this and the below heterogeneous fractal functions.

It is interesting to note that subjective experience indicates that modulation of crossover scale is more important than modulation of fractal dimension, for generating realistic looking terrain. That is, it is not so much that terrains have different roughnesses at different locations, as that they have a rather uniform roughness, but expressed at different scales. That changing crossover scale alone would have such a dramatic effect is not surprising, for as Mandelbrot has pointed out [80], the fractal dimension of the Himalayas is approximately the same as that of the runway at the JFK airport; it is simply that the lower crossover scale of the latter is on the order of millimeters while that of the former is on the order of hundreds of kilometers. Future work in terrain models might profit from experiments with models of uniform fractal dimension, but heterogeneous crossover scales.

2.3.2.3.3. Results

Plate 2.5 is a rendering of a detail of a 50×50 patch similar to that in Plate 2.4. Note that the triangles, which are barely visible due to bump mapping but can be discerned around the snow-covered peak, are quite large in comparison with the overall image. By including only relatively low frequencies in the terrain, and leaving high frequency details to the texture map, we can get realistic terrains from very small height fields. Such height fields can be rendered very rapidly. In Plates 2.4 and 2.5, as in subsequent plates of terrain patches, $\lambda = 2$.

Plate 4.2 illustrates a convincing model of ancient, heavily eroded models produced by the above terrain model. Here the difference in statistics between the (visible) low

areas and the peaks is relatively small, and the fractal dimension is quite low throughout. Such smooth, rolling terrain could not be generated by standard polygon subdivision algorithms, as it requires a smooth basis function, rather than a saw-tooth wave. Careful inspection reveals that the polygons tessellating the terrain surface are very large indeed -- note the piece-wise linear ridgelines. Thus the terrain model is composed of relatively few polygons, and renders very rapidly.

In Plate 2.6 the spectral exponent varies with both altitude and horizontal position. Here we have:

$$A_0 = F(x)\left(N\left(\vec{f}_0\right) + c_t\right)c_s + c_0$$

with $F(x) = \lfloor 2x, 2 - 2x \rfloor$, assuming that x varies from 0 to 1. To give the ridge a more natural path than that of a straight line, we add some noise to x before calculating $F(x)$. The contribution of higher frequencies is again scaled as:

$$A_i = A_{i-1} + A_{i-1}\left(N\left(\vec{f}_i\right) + c_t\right)c_s\varpi^i.$$

2.3.2.4. Modulating Lacunarity

It is readily apparent that the global value for lacunarity λ is subject to exact user control in the noise synthesis scheme. Computational cost in the creation of a model instance varies directly with the number of frequencies used. Generally, there is some desired bandwidth for a given fractal model, dictated by display resolution, available storage space, desired level of geometric detail, etc. Cost per unit bandwidth varies as the inverse of the lacunarity. Thus surfaces generated with small lacunarity will be more expensive to compute than those with large lacunarity.

Due to the point-evaluated character of the noise synthesis method, we may also exercise local control over lacunarity. This can be accomplished by displacing the initial coordinate $\vec{p}_0$ supplied to the noise function by a vector valued noise function $\vec{N}$ (e.g.,

Perlin's "DNoise()" [119]). The effects of such local change of lacunarity are shown in Plate 2.7b where we modulate intensity I on the image plane as:

$$I = N\left(\vec{p}_0 + \vec{N}\left(\vec{p}_0\right)\right).$$

Note that local change of lacunarity interferes with the precise local control of frequency, as it amounts to expanding the band limits of the basis function. While it is not immediately apparent that this local modulation of lacunarity is enormously useful for terrain synthesis, it has demonstrated value in the synthesis of textures such as clouds, smoke, and flames. The function illustrated in Plate 2.7b may be used as a novel basis function for the construction of fBm. Again, due to the small (usu. $\sim 3-12$ octaves, at lacunarity $\lambda = 2$) spectral sums used in constructing fBm for computer graphics purposes, the character of the basis function shows through clearly in the result. Compare the fBm constructions in Plate 2.7c & d. A variation of the fBm shown in Plate 2.7d is used in the clouds appearing in Plates 1.2 and 1.3, providing a "feel" to the cloud texture hitherto unavailable. Presumably, similar terrain models would also have a unique "feel" or appearance, but we have not to date experimented with such models.

2.3.2.5. A Large-Scale Terrain Model

The simple observation that local minima (valleys) should be smoother at all scales has led us to develop a novel terrain model, which turns out be useful for representing terrains on very large scales. Over large distances heterogeneity in topography is salient: plains, rolling hills, foothills, and alpine areas might all be present in a large area of terrain. This kind of variety is not present in naive fBm, which again is designed to be a homogeneous and isotropic function. Such heterogeneity in a fractal terrain model is useful, however, not only as a novel type of terrain model in its own right, but also in realistic modelling of terrains on planetary scales. Such capability is required to realize

our goal of situating our realistic landscape scenes in a coherent, global context, eventually to be explored interactively, perhaps in a VR (virtual reality) setting.

A surprisingly simple variation of the fBm functions described above allows us to generate highly heterogeneous terrains, which are more appropriate to very large scale terrain modelling than prior fractal terrain models. This model has applications in both planetary modelling (see Plate 4.3) and more local terrain modelling (Plates 2.8 and 2.9).

2.3.2.5.1. The Algorithm

In the above model, we made the fractal statistics of terrain models a function of altitude and spatial position. The observation that local minima (valleys) on all scales should fill up with debris and thereby become smoother than local maxima, led to the following construction, in which the amplitude of subsequent (higher) frequencies $v_i = N(\vec{f}_i)$ in the spectral sum are weighted by the amplitude of the previous (lower) frequency v_{i-1}:

$$A_i = A_{i-1} + \frac{F(v_{i-1})v_i}{f_i^{\beta}}$$

where $F(v)$ is displaced, scaled and clamped linear function of v:

$$F(v) = \begin{cases} 1 & : \quad v < 0 \\ 0 & : \quad v > c_c \\ \dfrac{v}{c_c} & : \quad o.w. \end{cases}$$

The dynamics of this spectral construction is perhaps more clear in pseudo-code:

```
Terrain( p⃗, H, lacunarity, octaves, offset, threshold )
        signal  := 0.5 * ( Noise( p⃗ ) + offset )                    /* scale to range of 1 */
        result := signal
        for octave := 2 to octaves
                p⃗ := p⃗ * lacunarity
                amplitude := pow( lacunarity, -H * octave )
                weight := clamp( signal/threshold, 0.0, 1.0)        /* clamp to interval [0, 1] */
                signal := weight * 0.5 * ( Noise( p⃗ ) + offset )
```

```
            result := result + amplitude * signal
        return ( result )
```

Below is C code implementing this algorithm. Again, the point is not to show the details of the code, but rather its brevity.

```
            /* a highly heterogeneous fractal terrain function */
double Terrain( point, spectral_exp, lacunarity, octaves, offset, threshold )
      Vector      point;
      double      spectral_exp, lacunarity, octaves, offset, threshold;
{
      register double   i, result, amplitude, frequency=1.0;
      register double signal, weight;

      signal = result = 0.5*(offset + Noise3( point ));
      for( i=octaves-1; i>0.0; i-- ) {
            point.x *= lacunarity;
            point.y *= lacunarity;
            point.z *= lacunarity;
            frequency *= lacunarity;
            amplitude = pow( frequency, -spectral_exp );
            if ( i < 1.0 ) amplitude *= i;      /* for octaves remainder */
            /* weight successive contributions by previous signal */
            weight = signal / threshold;
            if ( weight > 1.0 ) weight = 1.0;
            if ( weight < 0.0 ) weight = 0.0;
            signal = weight * 0.5*(offset + Noise3( point ));
            result += amplitude * signal;
            if (amplitude<VERY_SMALL || weight<VERY_SMALL) break;
      }
      return( result );
} /* Terrain() */
```

Comparing this code to that for homogeneous fBm, as seen in Section 2.3.2.1, we see that the elegance of the fBm model has not been too severely compromised, yet we have gained much complexity in the statistical behavior if the resulting fractal function.

2.3.2.5.2. Discussion

The operative feature of this algorithm is that subsequent contributions of higher frequencies are weighted by the (displaced) value of previous, lower frequencies. Thus, once a local minimum has been established at a particular frequency, all higher frequency contributions will be small-to-zero, keeping that area "smooth" at smaller scales. This is equivalent to a monotonically-decreasing power spectrum, or a monotonically-increasing spectral exponent. The fractal dimension of the function will never exceed that specified

by the spectral exponent β, but can be lower-to-flat at any scale. Thus we have stochastically modulated the fractal dimension, upper crossover scale, and bandwidth of the spectral sum.

Note that this algorithm is fundamentally flawed, as an implementation of the stated intention. That idea was to smooth out local minima at all scales, in imitation of valleys filling up with dirt, mud, etc. This algorithm smoothes out areas where there is a local minimum in a higher frequency in the current spectral sum, not in the resulting function. The minima in higher frequencies may occur where the local slope of the existing terrain is arbitrarily steep, due to the previous contributions of lower frequencies. Correct implementation of the idea would require knowing the local derivative of the sum-to-date. This should be implemented in the future. The current model, flawed though it may be, has nevertheless proved a powerful descriptor of natural terrains.

2.3.2.5.3. Results

This model is useful for modelling very large-scale terrain, as seen in Plate 2.8: it can generate plains in some areas, rolling hills in others, and jagged alpine mountains in still others. Again, this better mimics the appearance of Earthly terrains than the homogeneous fBm usually used for fractal terrain models, but with little added complexity in the generation function or compromise to the fundamental elegance of the naive fBm model. One way to think of this model is that fBm represents the simplest possible form of complexity -- a precisely defined spectral sum -- whereas this model bumps up the complexity by a notch: there is a complex, stochastic definition of the spectral sum itself.

In Plate 4.3 we have used the model in the creation of continents on an Earth-like planet. Note that the fractal dimension, or "wigglyness", of the coastline varies widely

from place to place; also, we have areas of the continents which are rather bland and featureless, along with highly complex and detailed behavior in other areas.

Plate 2.9 illustrates the utility of the model in creating more interesting terrains on the local scale. Here we have used $1 - abs\left(N\left(\vec{p}\right)\right)$ as the basis function. Taking the absolute value of the (cubic polynomial) noise function introduces discontinuities in the derivative of the surface, which create ridges at all scales. This is the heterogeneous version of terrain model in Plate 4.1 (rendered non-procedurally, i.e., with fixed level of detail)

2.4 Dynamic Erosion Models

The terrain models described above were designed as attempts to create convincing emulations of eroded landscapes, at terrain generation-time. They are fairly successful in increasing both realism of fBm-based terrains and the repertoire of landforms which can be so represented. Yet there remain many aspects of erosion features in Nature, which are not addressed by these models: stream and river networks, alluvial fans, deltas, talus slopes, lakes, glacial valleys, cirques, moraines, etc. The most difficult to model of these are the globally coherent features, notably the drainage networks associated with rivers and glaciers.

Such globally-coherent features require global communication, or context sensitivity. Without it, problems such as avoiding self-intersection of riverbeds cannot be reliably solved. Context sensitive problems are notoriously difficult to solve in a computationally efficient manner. In this section we offer a preliminary presentation of some dynamic erosion processes, designed to be used on existing terrain models in a post-processing pass to add some of these challenging but important morphological features. One of these models deserves no claim of computational efficiency; the other two are reasonably efficient in generating the desired features. Unfortunately it is, of course, the former algorithm and its intended results which are of by far the greatest interest.

Of all the areas addressed in this dissertation, this is the least developed. Work is very much underway in refining and extending the models and results discussed here. Thus we disclaim this as a preliminary presentation of preliminary results. The promise of this research is great, as are the difficulties faced (primarily in obtaining a stable, efficient numerical solution to the nonlinear partial differential equations involved in modelling fluid transport, and their boundary conditions). Work in the area will proceed into the foreseeable future.

Our erosive processes fall into three categories: *fluvial erosion*, *thermal weathering*, and *diffusive erosion*. Hydraulic erosion is that caused by running water. What we term "thermal weathering" subsumes the non-fluvial processes which cause rock to flake off steep inclines and form talus slopes at their bases. Diffusive erosion includes processes which transport substrate laterally, without regard to local slope (i.e., whether uphill or downhill). In this section we will illuminate these erosion simulation algorithms.

2.4.1. Fluvial Erosion

The fluvial erosion model involves depositing water ("rain") on vertices of the height field and allowing the water, along with sediment suspended therein, to move to lower neighboring vertices. The erosive power of a given amount of water is a function of its volume, the local slope of its transport path and the amount of sediment already carried in the water; this function is known as the *erosion law*. [3,135]

The fluvial erosion model is implemented by associating with each vertex v at time t an altitude a_t^v, a volume of water w_t^v and an amount of sediment s_t^v suspended in the water. At each time step, we pass excess water and suspended sediment from v to each neighboring vertex u. The amount of water passed, Δw, is defined as:

$$\Delta w = \min\left(w_t^v,\ \left(w_t^v + a_t^v\right) - \left(w_t^u + a_t^u\right)\right)$$

If Δw is less than or equal to zero, we simply allow a fraction of the sediment suspended in the water at v to be deposited at v:

$$a_{t+1}^{v} = a_t^{v} + K_d s_t^{v}$$
$$s_{t+1}^{v} = (1 - K_d) s_t^{v}$$

This amounts to asymptotic settling out of suspended sediment in standing water. Otherwise, we set

$$w_{t+1}^{v} = w_t^{v} - \Delta w$$
$$w_{t+1}^{u} = w_t^{u} + \Delta w$$
$$C_s = K_c \Delta w$$

Here, C_s is the *sediment capacity* of Δw. When passing sediment from v to n, we remove at most this amount of sediment from s_t^{v} and add it to s_{t+1}^{u}. If C_s is greater than s_t^{v}, a fraction of the difference is subtracted from a_t^{v} and is added to s_{t+1}^{u}, which constitutes the erosion of soil from v. Finally, we allow a fraction of the sediment remaining at v to be deposited as above. Thus, if $s_t^{v} \geq C_s$, we set:

$$s_{t+1}^{u} = s_t^{u} + C_s$$
$$a_{t+1}^{v} = a_t^{v} + K_d (s_t^{v} - C_s)$$
$$s_{t+1}^{v} = (1 - K_d)(s_t^{v} - C_s)$$

Otherwise:

$$s_{t+1}^{u} = s_t^{u} + s_t^{v} + K_s (C_s - s_t^{v})$$
$$a_{t+1}^{v} = a_t^{v} + K_s (C_s - s_t^{v})$$
$$s_{t+1}^{v} = 0$$

The constants K_c, K_d, and K_s are, respectively, the *sediment capacity constant*, the *deposition constant* and the *substrate softness constant*. K_c specifies the maximum amount of sediment which may be suspended in a unit of water. K_s specifies the softness of the substrate and is used to control the rate at which "bedrock" is converted to

sediment. K_d specifies the rate at which suspended sediment settles out of a unit of water and is added to the altitude of a vertex.

Through the above process water and, more importantly, substrate mass from higher points on the landscape is transported to and deposited in lower areas. This movement constitutes the communication necessary for modelling the global process of erosion. Unfortunately, it also involves finding a numerical solution to some challenging nonlinear partial differential equations, and current work is focusing on finding a reasonable solution thereto. (As the author's primary area of interest and expertise is computer graphics, not numerical analysis, the involvement of other researchers in this project is essential.)

The resulting features bear reasonable resemblance to natural erosion patterns (see Plate 2.11). Ongoing research is concentrating on constructing a more sophisticated, physically accurate model, based on the erosion laws of the literature of fluvial geomorphology. [3,135]

Plates 2.10 and 2.11 show a 200x200 terrain patch before and after 2000 time steps of fluvial and thermal erosion. The erosion simulation required approximately 4 hours of CPU time on a Silicon Graphics Iris 4D/70 workstation. In this simulation, $K_c = 5.0$, $K_d = 0.1$, and $K_s = 0.3$. Note the gullies, confluences, and alluvial fans that have appeared in the eroded patch, which is rendered as a dry wash, i.e., without water present.

The uneroded patch shown in Plate 2.10 demonstrates a reasonable first approximation to an eroded landscape with a central stream bed. The uneroded patch was created by weighting the addition of always-positive noise values by the distance d of the point from the diagonal of the patch, which diagonal is also "higher" at the far end. The stream bed is made non-linear by the addition of an fBm offset to the distance d.

This patch demonstrates the flexibility of the noise synthesis method for terrain modelling; it did not require much time to construct.

The distribution of rainfall on landscapes in nature is strongly influenced by *adiabatics,* or the behavior of moisture-laden air as it rises and descends. As air rises, it cools and the relative humidity rises. When the relative humidity becomes great enough clouds form; when the clouds become sufficiently dense, precipitation occurs. Wind blowing over mountains causes air to rise as it passes over the mountains, thus precipitation is much greater in the vicinity of mountain peaks. It is easy to include a rough approximation of adiabatic effects in our erosion model by making precipitation a linear function of altitude. This has a significant effect on the erosion patterns produced. Obviously, some complexity could be introduced by attempting to model prevailing winds, rain shadows, seasonal variations, etc.

In our use of the fluvial erosion model, we have simply allowed a fixed amount of rain (approximately one one-thousandth of the height of the vertex) fall at regular intervals (approximately every sixty to one hundred time steps). Mandelbrot [81] has shown that records of flooding of the Nile river show a $1/f$ noise distribution, i.e., large floods happen with low frequency. A correspondingly noisy distribution in the rainfall rate would constitute a more realistic simulation of nature. It is probable that it would have a long-term effect on the erosion features created: hundred-year and thousand-year floods may well, after all, exert a greater morphogenic influence than all the intervening years of less dramatic erosion. This is an idea that has yet to be explored.

2.4.2. Thermal Weathering

Another erosion process we model is "thermal weathering", which is our catch-all term for any process that loosens substrate, which subsequently falls down to pile up at the bottom of an incline. The thermal weathering process creates talus slopes of uniform

angle. Thermal weathering is a kind of relaxation process and is both simple and fast. At each time step t+1, we compare the difference between the altitude a_t^v at the previous time step t of each vertex v and its neighbors u to the (global) constant *talus slope*[*] (T). If a neighbor is lower than the talus slope, we then move some fixed percentage c_t of the difference onto the neighbor.

$$a_{t+1}^u = \begin{cases} a_t^v - a_t^u > T & : \quad a_t^u + c_t\left(a_t^v - a_t^u - T\right) \\ a_t^v - a_t^u \le T & : \quad a_t^u \end{cases}$$

With care taken to assure the equitable distribution of talus material to all neighboring vertices, the slope to the neighboring vertices asymptotically approaches the talus angle.

Plates 2.12 and 2.13 show a patch created with non-uniform lacunarity before and after slumping or thermal weathering. This process has created a rough approximation of sand dunes.

2.4.3. Diffusive Erosion

Diffusive erosion processes, also known as *dry creep*, essentially amounts to a progressive spatial low-pass filter over time. The dynamic of the process, caused in nature by bioturbation (as by the footfalls of grazing animals) and sediment transport by raindrop splashing, consists of the rounding-off of sharp features, both peaks and valleys. It is trivially implemented by transporting a quantity of substrate in each time step, as a function of local slope:

$$a_{t+1}^u = c_d\left(a_t^v - a_t^u\right)$$

---

[*] The *talus slope* or *angle of repose* is a fixed angle or slope which is characteristic of rubble of a given size, shape, and composition. Piles of rubble with sides steeper than this will spontaneously collapse under the influence of gravity; slopes at or below this slope are stable.

where c_d is the *coefficient of diffusion.* This process is exactly equivalent to repeated convolution with a low-pass filter kernel, as the diffusive transport mechanism may move material uphill as well as downhill. [13]

2.4.4. Discussion of Erosion Models

Again, this write-up represents a preliminary exposition of preliminary results. Many extensions of this model have already been implemented, but are not yet sufficiently developed to merit careful documentation. The models represent a rich area for future research. We now mention a few of the foreseen (and, in some cases, already realized) extensions, modifications, and applications of this work.

One extension accounts for the differing hardnesses of bedrock, silt, and talus. This is accomplished by adding appropriate fields to the vertex data structure and making the simplifying assumption that silt is on top of talus, which is in turn on top of bedrock. Hardness increases from silt to bedrock. Another simple and interesting extension is to modulate the hardness to the bedrock: this can be done in strata, as in sedimentary rock, or with a space-filling fractal field of hardness values. Either can be easily implemented with a procedural solid texture, as described in Chapter 5. (See Plate 1.2 for an example of a sedimentary strata texture.)

Creating animations of erosion will be trivial, as it is a dynamic process which need only be imaged at every n time steps and played back in sequence, to yield a visualization of the process acting through time. It is hoped that the process of orogenesis (mountain building) and perhaps the creation of features like the Grand Canyon could be produced using a combination of the techniques mentioned above.

Other problems, such as the realistic rendering of landscapes including running water, dry washes, deltas, alluvial fans, etc., may keep researchers occupied for some time to come, as they do not seem to admit to obvious, simple solutions.

2.5 Conclusions

We have demonstrated some novel methods for generating fractal terrain models of enhanced realism. These models include terrain generation-time models of erosion features and heterogeneity that enhances their resemblance to natural terrains. The noise synthesis modelling method derives most of its power from its point-evaluated, procedural nature. It can be utilized in procedural rendering of terrain models with adaptive level of detail, as described in Chapter 3.

Our generation-time erosion models have been supplemented by dynamic erosion models which create globally context-sensitive features which are difficult to include in generation-time algorithms. These erosion models represent work in progress, and offer rich potential for future research.

The net results of this work has been a substantial improvement in the fidelity and variety of "fractal forgeries of Nature" we can create.

Chapter 3: *Grid Tracing* for Rendering Height Fields

Chapter Abstract

In this chapter we present a fast and memory-efficient algorithm for ray tracing height fields. The algorithm takes advantage of the regularity and spatial coherence of the height field structure, which consists of a regular, square lattice of altitude values. It employs a modified Bresenham DDA* to traverse this two dimensional array. At each cell of the grid, the altitude of the ray is compared with the heights of the four corners of the cell; ray/object intersections need only be calculated when the altitude of the ray is in the range of those heights. The average number of ray/triangle intersections performed is about two per ray; the two triangles tested for intersection are located in $O\left(\sqrt{N}\right)$ time where N is the number of height values in the field.

This chapter is based on a technical report written in 1988 [94], thus some of the references to specific hardware platforms may be dated. As such specific references are always of ephemeral relevance, we have not brought them up to date at the time of this writing in early 1993. Subsequent progress by other researchers has clarified the place of this algorithm in the field of height field ray tracing algorithms: while other algorithms

---

* *DDA* stands for *digital differential analyzer*, which is the standard algorithm for line drawing in raster graphics. It is designed to efficiently identify, given two endpoints, which pixels a line between them will traverse. It does this by an incremental traversal method, moving from one endpoint to the other. This is exactly analogous to the problem of identifying which cells of a height field a ray traverses, proceeding out from its origin.

have been shown to be faster [82,115], grid tracing remains the most memory-efficient of published schemes.

3.1 Introduction

Grid tracing is a fast, memory-efficient algorithm for ray tracing height fields. A height field is a data set composed of a two dimensional array of altitude values, which are generally interpolated in two dimensions to describe a surface. Grid tracing is essentially a fast method for traversing this data structure and testing bounding volumes, which takes advantage of the spatial coherence and intrinsic ordering of a height field. The problem that motivated this work is that of ray tracing fractal terrains [34,35,77,154], specifically large height fields with huge number of triangles tessellating the surface.* In a naive ray tracing algorithm, the time required to ray trace a scene is a function of the number of primitives in the scene; most of the computing time in ray tracing is spent on calculating ray/object intersections. [127,158] Many schemes to reduce the number of ray/object intersection tests have been devised. [5,36,41,58,59,82,127,141,156] Grid tracing is similar to the 3DDDA scheme used by Fujimoto [36] to reduce the number of ray/object intersections, in its incremental traversal scheme, but grid tracing is a more efficient approach, albeit one limited to rendering height fields. With grid tracing, height fields become a new primitive object for ray tracing, joining the set of usually-simpler primitives such as spheres, planes, polygons, cylinders, cones, etc. As such, height fields can be incorporated into general ray tracing systems such as that of Snyder et al. [141]

There are at least two other published algorithms for ray tracing height fields: the quad-tree approach outlined by Kajiya [54] and Mastin et al [82], and the more recent

---

* We use triangles to tessellate the surface because A) planar primitives are the fastest to ray trace (the line/plane intersection being particularly easy to calculate), and B) two triangles can create a planar decomposition of the four non-coplanar points at the corners of a square height field cell.

"parametric" approach by Paglieroni et al. [115] The quad-tree scheme is similar to grid tracing in that it is a bounding-volume scheme for paring unproductive ray/surface intersection tests. It is based on a hierarchical decomposition of the height field into successively smaller squares, with successively tighter bounding volumes. It has been shown by Paglieroni et al [115] and Pharr [122] to be significantly faster than grid tracing. The parametric approach is based on the concept of *distance estimators* [114] borrowed from the computer vision literature. It relies on precomputed *parameter planes*, which in turn may require $O(n^2)$ comparisons of all height field altitude values to create. Once in place, they facilitate large incremental steps across the height field, which rapidly converge on the desired ray/surface intersection.

What distinguishes those two algorithms from grid tracing, is their overhead in memory. Memory constraints are a significant concern as the sheer size of the height field data set can easily overwhelm the memory capacity of any commonly available computer. Grid tracing incurs no significant overhead in memory, if a fixed-sized cache for the objects (e.g., triangles) tessellating the surface is used (a strategy that is of equal importance to the other schemes). Both the quad-tree and parametric algorithms require significant memory for storage of their auxiliary structures. Thus, for a given fixed allotment of real memory, we can render larger height fields with grid tracing than with the competing schemes.

An approach which eliminates the need for external storage of the height field data set is *procedural* rendering. [15,54,68,86] It applies only to synthetic terrain models, and cannot be used for measured terrain data such as USGS DEMs.* In a procedural

---

* USGS DEM ≡ *United States Geological Survey digital elevation map*. These are readily available elevation data sets corresponding to the USGS "quads", or topographic maps. They are the most commonly imaged height field data sets, perhaps followed by planetary data from Mars, Venus, and certain moons of the gas giant planets. (See Appendix 2.)

approach, the model is not computed until a ray "violates its airspace". As the algorithm determining the morphology of the surface is deterministic, no information describing the primitive polygons which compose the surface need be computed and stored in advance of rendering. Thus savings in external storage space is exchanged for increased computation time (evaluation of the model, which usually requires significant computation, may need be repeated at a specific location if the results are not stored or at least cached). An advantage of procedural rendering schemes is that the level of detail in the representation of the surface may be dynamically changed for the purposes of animation. Drawbacks of procedural schemes include increased computation time and difficulty of implementation. [149]

The grid tracing approach is designed to for rendering height fields of fixed spatial resolution. To date, all such height fields have been precomputed, though there are no significant obstacles to procedural creation of the height field data, on-the-fly, at rendering time. While grid tracing does not immediately accommodate changing height field resolution (i.e., level of detail) with distance or for a dynamically changing point of view,* rendering precomputed height fields is generally much faster than any published procedural scheme.

This idea for ray tracing a height field, as opposed to a procedurally generated surface, was generated out of necessity. Procedural definitions of terrain surfaces rely on some knowledge of the shape of the projected area covered by the polygons in the patch of terrain, for use in determining ray/surface intersections. As part of ongoing research with Prof. Mandelbrot, polygon subdivision schemes for generating fractal terrain models

---

* Recently Anderson [2] has modified the grid tracing code in Rayshade [67] to accommodate multiresolution rendering. In this scheme, height fields of the same terrain patch, at a variety of spatial resolutions, are loaded into the renderer. As rays proceed away from their origin, they automatically switch to lower-resolution grids as a function of distance. While "cracks" may appear in the surface due to discontinuities at the boundaries between of different resolution, striking results have nevertheless been obtained in imaging Magellan data from Venus. [109]

were developed [78] which, while starting with a simple convex polygon such as a triangle or hexagon, evolve into terrain patches with fractal edges. (See Plate 2.1) This happens because the polygon subdivisions used do not "nest" neatly*, as does the common scheme of subdividing an equilateral triangle into four equilateral triangles by joining the midpoints of the edges. Because the successive stages of subdivision do not nest, the shape of the area covered by a polygon after subdivision is difficult to determine. What that shape is, depends on the subdivision scheme being used and on the number of levels to which the recursive subdivision will be performed. For these reasons, a procedural definition of our terrain models was deemed impractical.

3.2 The Algorithm

This is how the grid tracing algorithm works: A two dimensional array of altitude values is traversed in an arbitrary direction by a ray, using a modified DDA algorithm. The array is thought of as composing a *grid* of small square *cells* (corresponding to the pixels being illuminated by a DDA algorithm). Each cell has associated with it the altitudes of the four corners of the cell. As the ray traverses the array of cells, the altitude of the ray at each cell is compared to the four altitudes associated with the cell. Ray/surface intersection need only be checked when the altitude span of the ray over the extent of the cell intersects the interval of altitude defined by the lowest and highest of the four altitudes associated with the cell, i.e., the ray altitude is checked against the bounding volume of the cell. For a ray traveling above the surface, the condition can be stated:

$$\min\left(ray_{z_{near}}, ray_{z_{far}}\right) \leq \max\left(h_{i,j}, h_{i,j+1}, h_{i+1,j}, h_{i+1,j+1}\right) \tag{3.1}$$

* For an explanation of the "nesting" issue, see Mandelbrot. [78]

where the ray_z represent the altitudes of the end points of the ray segment within the cell and the $h_{i,j}$ represent the altitudes of the height field *H* at the four corners of cell i, j. As a surface linearly interpolating the height values within a cell can be represented with exactly two triangles (splitting the square diagonally), the ray/surface intersection test consists of two ray/triangle intersection tests. Only rays grazing through the crease between these two triangles will fail the intersection tests; most rays will incline directly into the surface at the first cell where surface intersection is tested. Thus this relatively computationally-expensive ray/plane intersection calculation is performed sparingly and with an excellent success rate, and most of the work is pushed into the far more efficient grid traversal calculations.

Advantages of this algorithm are manifold. First, only the height field need be calculated and stored as the model. The actual polygon descriptions (e.g., the plane equations of the triangles) need only be calculated (and optionally, stored) when an intersection is tested for. This can save both time and space in the creation of the terrain model, as polygons which are not visible in the rendering are never fully described or stored. Second, the grid traversal can be accomplished with the use of a modified Bresenham DDA algorithm. The Bresenham algorithm is a highly optimized, fast algorithm which uses only floating point addition* in determining the height of the ray and the next cell along the path of the ray. Third, the algorithm is general. Any two dimensional array of scalar data (e.g., an image) may be interpreted as a height field and ray traced with this algorithm. Fourth, due to the incremental front-to-back traversal scheme, we are guaranteed that the first intersection found will be the closest; no further test need be performed. Fifth, the algorithm performs ray/object intersections in $O\left(\sqrt{N}\right)$

---

* The Bresenham DDA most widely known is an integer algorithm. The version used for our purposes is not the integer Bresenham DDA, but rather a slightly less optimal floating point version. A simpler alternate scheme could use an ordinary integer DDA for the traversal, but would need to check one cell to either side of the ray path for possible intersections due to imprecision in tracking that path.

time, with a very small constant coefficient. Sixth, this algorithm is nicely amenable to stepwise implementation and optimization. Thus the implementation can be tested and debugged incrementally.

3.3 Implementation

Implementation of this algorithm can be accomplished in the following stepwise fashion, where each step represents a distinct programming and verification task:

1) Calculate the x, y, z coordinates of the intersection of the ray with the global bounding box of the height field.

2) Implement a standard DDA to traverse the grid in x, y coordinates until the ray exits the bounding box.

3) Modify the DDA to identify all cells traversed by the ray. (Line drawing DDA algorithms do not do this, due to the requirements of maintaining approximately constant illumination along lines drawn in the raster. [31])

4) Modify the DDA to track the z (altitude) values of the endpoints of the ray within each cell.

5) Test the z values of the ray at each cell against the altitudes at the four corners of the cell. If this test indicates that the ray passes between those altitudes at that cell, proceed with the steps below; if not, go to the next cell.

6) Create two triangles (only the plane description is needed) from the four altitude values at the corners of the cell.

7) Intersect the ray with the two triangles which tessellate the surface within the cell.

8) [Optional] Include an inverse skewing and scaling transformation for the ray, so the triangles may be equilateral rather than right triangles.

These steps implement the basic algorithm; many optimizations can subsequently be made. Note that the optional step 8) turns the height field patch, in world space, from a square to a diamond shape, but does not affect the shape of the square cells in DDA-traversal space. We prefer to tessellate our height fields with equilateral triangles, as this introduces less bias due to tessellation into the resulting surface and tiling with equilateral triangles yields the best possible ratio of area covered per unit height field data.

3.3.1 Modified DDA Algorithm

The DDA is the heart of this algorithm. Traversal of the grid with the DDA is where most of the CPU time is spent. There are two fundamental modifications to a line-drawing DDA which must be made for the purposes of this algorithm: First, the DDA must identify *all* cells traversed by the ray; a normal DDA identifies only one cell per unit travel along the driving axis. Second, the DDA must produce floating point intersection points with cell boundaries, rather than just identifying the integer x,y coordinate label of a cell, and care must be taken that cells are identified in the order in which the ray passes over them in its traversal from point *A* to point *B*. The second required modification makes a pure integer Bresenham DDA unusable for our purposes. The DDA we have used is an extension of the first Bresenham DDA presented in Rogers. [125]

The modifications to the DDA are extensive; a sample of the inner loop for rays in one octant is provided at the end of the chapter. We note that the number of comparisons to be performed at each cell (i.e., in the inner loop of the DDA) should be minimized. Naively, at each cell we must check six exit-of-bounding-box conditions for the top, bottom, and the four sides of the bounding box, and compare the four altitudes of the cell

corners against the values of the two ray endpoints at the cell boundaries. Careful construction of the DDA can reduce the number of comparisons, based on information about the path of the ray (e.g., noting that the ray is traveling in the positive x and directions, and negative z direction can eliminate three exit tests). Such optimization, trading increased quantity of code to be written and executed for a faster running time, is typical of a DDA algorithm.

An optimization we have implemented involves forming a secondary $(n-1)$ by $(n-1)$ grid A:

$$\left[A_{i,j} \quad \middle| \quad a_{i,j} = \max\left(h_{x,y}, h_{x+1,y}, h_{x,y+1}, h_{x+1,y+1}\right), \quad 1 \le x,y \le n, \quad 1 \le i,j \le n-1 \right]$$

where $h_{i,j}$ is the altitude of the i,j^{th} position in the height field H and n is the size of one side of the square height field. This grid A stores the maximum of the four altitude values for each cell, at the minimum x,y address of the cell. By thus precomputing the right hand side of inequality (3.1) for all cells, we can reduce that inequality to:

$$ray_{z_{\min}} \le a_{i,j} \tag{3.2}$$

using *a priori* knowledge of the ray's z direction to track $ray_{z_{\min}}$ without comparisons. Simply stated, we need only check if the minimum z value of the ray is less than the maximum altitude of the cell. Note that this approach constrains our point of view to be above the surface of the patch; a similar approach with inverted *min* and *max* would suffice for rays coming from below the surface.

3.3.2 Intersection Tests

The two ray/triangle intersection tests performed at a cell consist of:

1) finding the line/plane intersection for the ray and triangle plane, and

2) clipping the intersection point to the extent of each triangle in the cell.

The first step is a standard operation in ray tracing; the second step is very fast. For the lower left hand triangle of the cell it can be expressed:

```
if (intersection_x >= cell_x_min ) &&
   (intersection_y >= cell_y_min ) &&
   (intersection_x + intersection_y <= cell_x_max + cell_y_max )
            then intersected = TRUE;
```

where $\texttt{intersection}_{[x,y]}$ is the x or y coordinate of the line/plane intersection point and $\texttt{cell}_{[x,y]_{[\min,\max]}}$ is the minimum or maximum of the x,y coordinates of the four corners of the cell. Note that the latter are integers. The first two tests check the intersection point against the extent of the (square) cell. The third test checks the intersection against the diagonal of the cell, which divides the square into two triangles. In this construction the diagonal (arbitrarily) runs from top left to bottom right.

3.3.3 Memory Requirements

This algorithm is memory intensive, as will be any algorithm designed to efficiently handle large height fields. Memory resources are the major limiting factor in rendering such height fields, as opposed to computational power or time. We have rendered scenes containing over 5 million *virtual triangles*[*] (a height field of dimensions 1600^2); there is simply no easy, compact way to store that amount of data. Several approaches have been pursued to limit memory use.

The height field is a vector of altitude values. This vector may be quite large. Therefore, the data type used for elements of the vector has a direct, linear impact on the memory requirements of a vector of given size. In the C language on the system we used, a variable of type `double` requisitions 8 bytes, a `float` takes 4 bytes, an `int` takes 4 bytes, and a `short` takes 2 bytes. To decrease memory requirements, one should use no

[*] We refer to the triangles composing the tessellation of the surface as *virtual triangles* because they do not exist as entities, until they are needed for ray/surface intersection tests.

more bits than are necessary to encode altitudes to the required precision. Our first implementation used the type `float` for storing the $h_{i,j}$ and $a_{i,j}$ altitude values; this is not an optimal use of memory space, but it is easy to implement. More recently, with the assistance of Matt Pharr, we have converted the altitude data type to `short`, for a savings of 50% in storage space. [121] For maximal accuracy, the data values are displaced and scaled to lie within the vertical span of the overall height field data range. The high and low values of this range must be stored with the height filed data for reconstruction.

The only essential data that must be stored is the height field *H* and, optionally, the *A* grid. When ray/surface intersections are tested, additional data describing the planes of the two triangles in a cell is generated. This data may or may not be stored. If all such data is stored, it may cause the host machine to run out of available memory; if it is not stored, it must be recomputed at each test. Recomputing this data at each test is undesirable, as some triangles (particularly those in the foreground when the eye is close to the surface) may be intersected by many rays, thus indicating storage of the triangle plane data.

Our initial implementation of the `plane` data structure used four `doubles` or 32 bytes (clearly not optimal) and dynamically allocated memory for all triangles tested for intersection, never freeing that memory until completion of the rendering. With approximately 3 million triangles, this implementation could exceed the memory capacity of an Encore Multimax machine, with 64 Mb (megabytes) main memory and 115 Mb total virtual memory. Our first solution to this problem has been to implement a circular queue of `plane` structures. Thus we store only the number of `plane` structures in this queue (two to four times the length of a scanline in our implementation). When a new `plane` structure is created it uses a pointer to a `plane` structure in this circular queue, and the pointer to the `plane` structure which previously occupied that position in the queue is set to `NULL`. Another approach, for ray casting, is to store just two temporary

`plane` structures with the x,y coordinates of the cell in which they reside. By checking the x,y coordinates before recalculating the `plane` data, coherence of a triangle on a given scanline can be used to eliminate repeated calculations; recalculation is only necessary for each scanline on which the triangle is tested for intersection.

The efficacy of these caching schemes is strongly affected by two factors: the eye point's position relative to the grid, and reflection, refraction, and shadow ray propagation. If the eye point is everywhere distant from the surface, all triangles may be pixel- to sub-pixel size and will therefore not be intersected by more than a few rays. Storing only one cell's `plane` data will often be futile if shadow, reflected, or refracted rays are propagated. Thus the hit rate of either caching scheme can vary widely. The best scheme to date is the LRU (least recently used) cache implemented by Craig Kolb in Rayshade. [67]

3.4 Time Complexity

The worst case time complexity of this algorithm is $O\left(\sqrt{N}\right)$ where N is the number of height values in the field.* Note that $N = n^2$, where n is the size of a side of a square height field grid. The time is linear in the number of rays cast at the virtual screen. Each ray will traverse the grid with a number of operations proportional to, in the worse case, $2\left(\sqrt{N}\right)$ for a ray crossing the grid diagonally without intersecting the surface. The average case is closer to $\frac{C}{2}2\left(\sqrt{N}\right)$ operations, where C is the worst case constant number of operations per cell traversed, and the division by two is due to the expectation that the ray will intersect the surface, on average, halfway across the grid.

---

* We ignore the $O(N)$ overhead in forming the array A as this is done just once, while grid traversals have a constant coefficient upwards of the order of 10^6, corresponding to the total number of rays traced in the scene.

The number of virtual triangles t in a height field is:

$$t = 2\left(N - 2\sqrt{N} + 1\right) = 2(n-1)^2$$

as the number of cells c in the grid is:

$$c = N - 2\sqrt{N} + 1 = (n-1)^2$$

and there are two virtual triangles per cell. The DDA algorithm insures that the first intersection found will be the closest, therefore the search for ray/object intersections can be terminated at the first intersection. As most rays will intersect the surface in the first cell for which the line/plane intersection test is performed, the average number of ray/object intersection tests for any N, t or c is approximately two. The worst case is $4\left(\sqrt{n} - 1\right)$ for a ray that skims through a trough that runs diagonally across the grid (quite unlikely for a stochastic surface).

The low number of ray/object intersection tests is the real advantage of this algorithm. Naive ray tracing algorithms have a time complexity proportional to the number of primitive objects in the scene; most of their time is spent in the relatively expensive ray/object intersection tests. In this algorithm the time spent calculating ray/object intersections is small and constant,* and the time complexity is dominated by the relatively inexpensive search for candidate objects to be intersected with the rays. Note that the preprocessing overhead of this algorithm is small as well. The height field is calculated just once, in contrast to some procedural algorithms. The spatial coherence of the height field obviates potentially costly preprocessing required by some spatial subdivision algorithms, for sorting primitives and assigning them to the appropriate cells in a grid or hierarchy. In a height field, the primitives come pre-sorted, spatially.

---

* This corresponds to what Kaplan [58] calls "constant time" for his spatial subdivision scheme.

It is the inner loop of the DDA traversing the grid which dominates the time complexity of this algorithm. As this inner loop involves a small number of inexpensive operations, we claim that the constant scalar C to the $O\left(\sqrt{N}\right)$ time complexity is small and that the algorithm is quite efficient. In our use of this algorithm to ray trace fractal mountain patches, the execution time of our renderer is dominated by the evaluation of the procedural solid textures (see chapter 5) used to simulate water, treelines and snowlines, and clouds. Thus a typical 1280 by 1024 resolution image of a fractal patch described by a height field of size 407^2 (329,672 virtual triangles) ray traced at 1 ray/pixel may take 24 hours of CPU time on a single National Semiconductor 32332 microprocessor with fast floating point support. Of this time, only about 8 hours are spent in ray tracing and determining lighting for the surface; most of the balance is spent evaluating the procedural textures at ray/surface intersection points. Generation of the height field by polygon subdivision typically takes only a few minutes on a machine such as a Sun 3/60, and while it may take some tens of minutes to generate a large height field, this time is still considerably less than the time required for rendering.

3.5 Future Directions

Problems with the grid tracing technique include clamping spatial frequencies in image space. Distant portions of the height field rendered with a perspective projection will have arbitrarily high spatial frequencies, leading to aliasing. This aliasing will be particularly objectionable in animated sequences where it will cause scintillation on distant surfaces. Anderson has solved this problem with the multi-resolution, multi-grid extension to the grid tracing algorithm mentioned above. [2]

The grid tracing algorithm is amenable to many fine optimizations, mostly in the DDA. Other more significant global optimizations are possible. For instance, Mastin and Watterberg [82] have suggested a quad tree approach to ray tracing height fields which is reminiscent of Kajiya's procedural algorithm [54] for stochastic surfaces. In

Mastin's approach, the ray altitude is checked against a quadtree of bounding boxes of decreasing volume which enclose a rectilinear set of subdivisions of the surface (i.e., a square grid is subdivided into four smaller square grids, recursively). The data structure required to store such a tree adds an $O(\log_4 N)$ overhead in storage to the cost of storing the simple height field; this would be prohibitive for the sizes of height fields which we render with grid tracing.

The idea of a hierarchy of bounding volumes may be borrowed from the quad-tree scheme, to speed up grid tracing. By creating a hierarchy of grids, the advantages of a hierarchical data structure can be combined with the efficiency of the DDA grid traversal. Such a scheme is similar to that required for an adaptive level-of-detail implementation. Note that by making the grids smaller through a hierarchy, page faults can be decreased. Also, the time complexity of the overall rendering algorithm may be affected favorably. If the height field were placed in a hierarchy of m by m grids, the time complexity would go to $O(m \log_m N)$,with $\log_m N$ time required to traverse the hierarchical structure, with cost m for tracing m by m grids at all levels. Note that we introduce an $O(\log_m N)$ overhead in memory for the hierarchical structure. Thus we are implementing, in effect, a continuous tradeoff between the speed of the quadtree scheme (where $m = 4$) and the memory efficiency of grid tracing (where $m = n$, the dimension of one side of the grid), parameterized by m. Craig Kolb has implemented this scheme, with m fixed at compile time. We have used it extensively at $m = 16$, with good results (i.e., it runs quickly and does not overburden system memory capacity, with large height fields).

Grid tracing could be implemented in integer arithmetic. For example, the bounding box of the entire grid could be scaled in z (height) to the full range of the integers, thereby facilitating tracking of the ray altitude with an integer DDA. As mentioned above, the traversal of the grid can be accomplished with an ordinary integer DDA, with

the added expense of extra ray/surface intersection checks along the path of the ray. These extra checks are necessary because of error in the calculation of the ray path intrinsic in an integer calculation. Due to the current state of computer hardware in which floating point arithmetic can be as fast as, or even faster than, integer arithmetic, in terms of CPU cycles required per operation, we have not undertaking an integer implementation.

The surface of the height field can be represented with a bicubic spline, rather than a triangular tessellation. This requires some analysis of the local behavior of the spline, as the altitude of the surface within a cell might pass above that of the highest altitude at the corners of the cell. The behavior of the surface is less easily defined, and therefore ray/surface intersections are more complicated to compute. Calculating the intersection of a ray with a bicubic polynomial using Newton's method, is expensive. Our experiments indicate that it is far more efficient, time-wise, to simply fit the spline to the height field, resample the resulting surface at higher spatial resolution, and render the resulting height field.

3.6 Conclusions

Grid tracing is an efficient technique for ray tracing a limited class of complex tessellated objects. This algorithm has been successfully used to render over five million unique triangles in a single scene. Such a scene can easily have all polygons appear at pixel size or below, thereby eliminating the flat polygonal surfaces from the image of a tessellated surface. The major limitation to the use of this algorithm is the real memory capacity of the host computer (i.e., page faults); this can be ameliorated by implementing a hierarchical data structure of smaller grids. The grid tracing technique is general to all regular height fields, and calls for creative applications.

3.7 Code Segment of DDA Inner Loop

```
/* traverse the 2D grid of surface height data */
/* uses a modified non-integer Bresenham DDA */

register double   nearZ, farZ, minZ, error, delta, deltaX, deltaY,
deltaZ;
register int      xCell, yCell;
double            signX, signY;

/* set initial position */
nearZ = intersection_point.z;
farZ = (/* z value of ray at far end of cell */);
xCell = (/* x index of cell at ray intersection with bounding box */);
yCell = (/* y index of cell at ray intersection with bounding box */);
/* set DDA variables */
error = (/* an octant-specific initialization dependent on
            exact (floating point) intersection point with cell */);
delta = (/* delta for error */);
deltaX = (/* delta for x */);
deltaY = (/* delta for y */);
deltaZ = (/* delta for z */);
/* SIGN() returns +1 for a positive argument; -1 otherwise */
signX = SIGN(ray->dir.x);
signY = SIGN(ray->dir.y);
/* inner loop: while still in the grid, traverse.driving axis of DDA is
x axis */
do    {
      minZ = MIN(nearZ,farZ);
      if ( minZ <= HighestAlt[xCell][yCell] )
            if ( Intersection(xCell, yCell, ray) )
                  return;
      if ( error > VERY_SMALL ) {
            /* check the cell that a normal DDA would skip */
            yCell += signY;
            if ( (yCell < 0) || (yCell == sideMax) )
                  /* bounding box has been exceeded */
                  break;
            else
                  if ( minZ <= HighestAlt[xCell][yCell] )
                        if ( Intersection(xCell, yCell, ray)
                              return;
            error--;
            }
      else if ( error > -VERY_SMALL ) {
            /* ray crosses at exactly the corner of the cell (unusual)*/
            yCell += signY;
            error--;
            }
      xCell += signX;
      error += delta;
      nearZ = farZ;
      farZ += deltaZ;
      } while     ( (nearZ >= BoxBottom) && (nearZ <= BoxTop) ) &&
                    (xCell >= 0) && (xCell < sideMax) &&
                    (yCell >= 0) && (yCell < sideMax) );
```

Chapter 4: Atmospheric Scattering Models

Chapter Abstract

In scenes on the scale of landscapes, atmospheric effects are generally salient. The most common and important of these effects is the change in color and contrast with distance known as *aerial perspective*. This effect is a critical perceptual cue for indicating large scale in a rendering, be it a computer graphic or a painting. Also striking, if less ubiquitous in Nature, are the rainbow and mirage. Each of these phenomena is the result of redirection and/or absorption of light in its passage through the atmosphere, thus we classify them as *atmospheric scattering* effects.

Aerial perspective and the rainbow are optical effects dependent on the wavelength of light, while mirages are not, to any significant degree. Each of these effects varies with spatial location. Thus we present geometric models of the density distributions which modulate aerial perspective and the mirage, and describe a rainbow model based on geometric optics. Spatial variations having been determined, we develop models of the interaction of light with the participating medium. This takes the form of a variation on Beer's Law [19] for aerial perspective, of the Fresnel equation for the mirage, and of both for the rainbow model. In Fournier's taxonomy [32] of models of natural phenomena, our models range from "impressionistic", in the case of the aerial perspective model, to "physical", or what Barr [7] might classify as "teleological", in the case of the rainbow model.

4.1 Some Methods for Aerial Perspective

In this section, we present our models for aerial perspective. Our Rayleigh scattering model is a simple rgb-space extension of Beer's Law. This is not a physical model, yet it reflects the most essential behavior of natural systems. We also present some geometric models for aerosol density distributions, which serve well for production image synthesis and could be incorporated in scattering models of greater physical veracity.

4.1.1 Introduction

A critical factor in the realistic appearance of synthetic terrain images is the sense of grand scale provided by *aerial* or *atmospheric perspective*: the change in color and decrease in contrast, or changing to blue-grey, of objects seen in the distance. Atmospheric perspective has been known for hundreds of years by landscape painters [40] to be as important a distance cue as the more-familiar *geometric perspective*, wherein size varies as the inverse of distance and parallel lines converge to so-called "vanishing points". Cartographers also have used aerial perspective for over a hundred years, as an aid to visualizing topographic data in map making. [50]

It is important to note that synthetic terrain models and digital elevation maps (DEMs) are simply abstract data sets which, when properly interpreted visually, may resemble the surface of a landscape; as such, they have no intrinsic scale. Thus in our renderings we must provide visual cues which indicate the intended grand scale of real terrain. For this reason, it is worth developing realistic, efficient models of atmospheric scattering which faithfully reproduce the observed behavior of Nature. (See Plate 4.1.) For the purposes of production image synthesis, we wish to derive elegant and computationally efficient models, and to avoid the over-modelling that sometimes accompanies physically-accurate simulations. [32]

As Blinn has noted [12], modelling scattering involves two distinct problems: devising geometric models of aerosol density distributions, upon which the optical density of scattering depends, and integrating the scattering & attenuation of light over arbitrary optical paths through these density distributions. The first problem boils down to deriving plausible density distributions which are efficiently-integrable. The second deals with modelling the effects of the participating medium on the propagation of light; these effects include extinction and scattering,* and the dependency of such effects on wavelength or color. For the purposes of production image synthesis efficiency is of paramount importance in both types of model, as we wish to maintain the elegance and accuracy of per-ray computation of atmospheric effects. (This may be contrasted to, for instance, the interpolation of sparse samples proposed by Klassen. [64])

It is well known that physically accurate models of atmospheric scattering are computationally impractical, for the purposes of production image synthesis. [12,18] We will show that reasonably good subjective approximations to the physical behavior of Nature can be had through some relatively simple constructions, which are sufficiently efficient to use on a per-ray basis in production image synthesis. These models may serve as a basis upon which to build models of greater physical veracity.

4.1.2 Problem Statement

Despite the rich supply of scientific literature on the topic, a general light-scattering model remains an open problem in computer graphics. It is conceivable that a relatively straightforward solution is available, however, that solution will most likely be computationally intractable for the purposes of production image synthesis, due to the

---

* For our discussion, we define extinction as the removal of light energy along an optical path by absorption or scattering out of the path, and scattering as the redirection of light energy towards the origin of the optical path, i.e., as (only) towards-the-eye scattering.

complexities of photon propagation and energy transfer in multiple and frequency-dependent scattering. Yet the appearance of terrestrial terrains on the large scale is strongly affected by scattering, particularly wavelength-dependent Rayleigh scattering: it is Rayleigh scattering which gives distant mountains their purple cast; makes sunsets red; gives direct sunlight a yellow hue; and makes the sky blue, thereby causing areas not in direct sunlight to be illuminated with a cool, bluish hue. [40]

Correct calculation of the illumination I at the origin (eyepoint) of optical path p in the presence of Rayleigh scattering requires solving the integral equation:

$$I = \int_{\lambda} \left\{ I_{endpt_{\lambda}} \int_{p} e^{-\tau_{\lambda}} + \int_{p} \left[\frac{3}{4} \left(1 + \cos^2 \theta \right) \right] \left[\frac{2}{3} \pi^2 \frac{\left(\eta^2 - 1 \right)^2}{N_u \lambda^4} \right] \left[e^{-\tau_{\lambda_u}} \right] \left[I_{0_{\lambda}} \int_{l} e^{-\tau_{\lambda}} \right] du \right\} d\lambda \qquad (4.0)$$

where τ is a function of aerosol density; the first term in the second integral over p is the *phase function* which is a function of θ, the angle between the optical path and the light source I_0; the second term is the Rayleigh scattering coefficient which is a function of the refractive index η, the molecular density N at u, and wavelength λ; the third term is the extinction occurring between point u and the origin of the optical path; and the fourth term represents the light reaching u from the light source, as extinguished over path l. This formulation is for a single light source and does not account for shadows. It is valid for single-scattering only; multiple scattering is more complex.

Wishing to avoid the complexities inherent in solving equation (4.0) and its more accurate descendants, we propose a non-physical, computationally-efficient, practical alternative to a physically- or empirically-based implementation of Rayleigh scattering. Our model is sufficient as a first approximation to Rayleigh scattering, and is surprisingly simple to implement. Its cost is little more than that of an ordinary "fog" function which features no frequency-dependent behavior, aside from the fog's intrinsic color. It may be argued that this model represents the distillation and generalization of physical scattering

models, making their essential behavior practically available for production image synthesis. It embodies the basic principle of frequency-dependence or color-dependence in scattering, abstracted from concerns with the geometry of scattering and atmospheric structure. Our model may be evaluated efficiently, freeing us from the need for image-space sample interpolation, lookup tables, or other such speedup schemes.

The atmosphere of the Earth and other planets is not homogeneous: its density varies exponentially with altitude, and optical paths not limited by intersection with some object are limited by the curvature of the atmosphere around the planet. Thus we will also develop some aerosol density distribution functions for use with the (independent) scattering models. Like our scattering model, these density models are designed to be simple, computationally-efficient approximations to the behavior of the Earth's atmosphere. These density models could be employed to improve published physically-based implementations of Rayleigh scattering. Coupling them with our scattering model, we obtain something that is in some ways superior to more physically accurate models: it is much faster, much easier to implement, and can actually *look* better than a physically-based single-scattering Rayleigh model. (We make the latter claim based on experience with several, independent implementations of such physical models.)

It should be stressed that the geometric aerosol density distribution models are entirely independent of the scattering model. Thus they may be employed in conjunction with more accurate scattering models, if desired, to improve both the accuracy of the simulation and the image results obtained. While we will develop our scattering model in the rgb color space, it is also worth stressing that it could be implemented using a vector of frequency samples in the CIE xyz color space [163], the resultant of which is subsequently transformed to an rgb value for display. Again, it is the goal of this work to present the simplest, most elegant possible distillation of the relevant effects. Complications and improvements are quite easy to add.

4.1.3 Previous Work

Effects of participating media -- primarily atmospheric effects -- have been modelled since Whitted's "Foggy Chessmen". [157] Scattering of light by a participating medium should be a primary term of the rendering equation [56]; this fact is sometimes overlooked. Researchers who have addressed scattering problems in the literature include Blinn [12], Kajiya et al [57], Klassen [64], Max [83,84], Musgrave [98], Nishita et al [111], and Rushmeier et al. [130] Some of these models include scattering effects which depend on the frequency (i.e., the color) of the light; most do not.

Klassen has described a detailed physical* model of Rayleigh scattering. [64] Unfortunately, this model is simultaneously too complex to admit to efficient implementation and too simple to model nature adequately: the functions involved are expensive to integrate; the assumptions about the distribution of atmospheric aerosol density are simplistic; and the single-scattering model is inadequate, particularly for short wavelengths and when the sun is close to the horizon. In a single-scattering implementation of Rayleigh scattering, sunsets appear to have a red horizon, fading to a dark olive-colored sky above -- clearly not a very realistic simulation of Nature. This appearance can be improved by adding an ambient term which varies with altitude to represent multiply-scattered light, particularly blue and violet light, but a physical basis for such an ambient term has not been described in the scientific literature, thus the physical legitimacy of the model is compromised.

Klassen sidesteps the issue of multiple scattering by making the assumption that it is acceptable to simply color the background a uniform shade of blue, the intensity of which

---

* We will refer to Klassen's model as a "physical" model as a matter of computer graphics convention, as it is based on a model from the scientific literature. However, Lord Rayleigh's scattering model is, in fact, an empirical model rather than a teleological model derived from first principles; thus Klassen's model may be argued to be "empirical", rather than "physical".

is not physically or empirically justified.* As we will require the flexibility of having arbitrary view points, including those outside of the atmosphere, this assumption would be unworkable even if it were deemed intellectually acceptable.

Many practitioners seem to have used versions of the exponential mist aerosol distribution function described here, for some years now. Nevertheless, other than simple descriptions of Beer's Law [19] for simulating homogeneous fog, there seem to be no descriptions of more complex geometric models of aerosol density distributions in the commonly-available graphics literature. Thus, while some of these results may not be unprecedented, this may represent the first publication of their full technical description.

4.1.4 Homogeneous Fog

The simplest atmospheric scattering function is a homogeneous, isotropic fog, as described by Beer's Law. [19] This function displays the same behavior at every place and in every direction, so the only parameters of import are the *extinction coefficient* and the color of the fog. The extinction coefficient ε (or *coefficient of absorption*) is defined to be the portion of light scattered per unit distance δ. The product $\varepsilon\delta$ of the extinction coefficient and the distance travelled is the dimensionless quantity τ, called the *optical depth* or the *optical thickness*. [144] The *transparency* σ of the fog along the optical path is an exponential function of the optical depth $\tau = \varepsilon\delta$:

$$\sigma(\varepsilon, \delta) = e^{-\varepsilon\delta} = e^{-\tau} \tag{4.1.1}$$

The transparency $\sigma{:}1 \rightarrow 0$ as distance $\delta{:}0 \rightarrow \infty$; the *opacity* along the optical path (in a single-scattering model) is $1 - \sigma$. The illumination sample I which a given ray

---

* The hue of such a background color should at least correspond to a λ^{-4} spectrum, λ being wavelength. Klassen, however, indicates [64] that the shade of blue for the sky thus derived in CIE XYZ color space may be out of gamut for the average monitor. This, as well as increased complexity, contraindicates the use of XYZ color in a Rayleigh scattering model.

represents is a function of the value calculated by the surface shader at the end point of the ray, I_s, and the contribution of the atmosphere I_a along the ray path:

$$I = \sigma I_s + (1 - \sigma) I_a \tag{4.1.2}$$

Note that this is not a physical model: there is no provision for shadows or *crepuscular rays* (i.e., beams of light) in the atmosphere and there is no attempt to balance energy transfer; and I_a is simply an rgb or frequency vector-valued color for the fog. It does, however, simulate nicely the decrease in contrast with distance that characterizes real fog, smoke, and haze.

We stated above that opacity equals $1 - \sigma$, i.e., that extinction exactly equals scattering in the direction towards the ray origin. This may not hold in Nature or for a multiple-scattering model. Furthermore, for aesthetic purposes in image synthesis we may wish to exert independent control over the scattering and extinction. Thus we may wish to calculate different values of σ for scattering and extinction. We may accomplish this by using different coefficients ε for scattering and extinction. While this increases flexibility, it also contributes to the undesirable proliferation of parameters for the user to deal with, which often characterizes models of natural phenomena. [87] Experimenting with separate ε parameters represents an area for future research.

4.1.5 Exponential Mist

Our next step towards increased realism is to make the density of the atmospheric effect vary exponentially with altitude. Lynch [75] has shown such exponential distribution of scattering aerosols to be responsible for the visual appearance of

successive mountain ridges receding in the distance; indeed, the construction presented here represents a clarification of his rather convoluted derivation of the same model.

In this model, the extinction coefficient ε varies with altitude; to get the optical depth we require the average value of ε over the optical path. We get this by integrating ε over the vertical interval traversed in the optical path and normalizing by the width of the interval. Thus we require an exponential function which is readily integrable. The function $f(x) = e^{-x}$ is ideal, as

$$\int e^{\alpha x} dx = \frac{1}{\alpha} e^{\alpha x} + c \tag{4.1.3}$$

for real $\alpha \neq 0$. The constant c introduced by the integration can safely be ignored for our purposes. For $\alpha = -1$ we have:

$$\int e^{-x} dx = -e^{-x} \tag{4.1.4}$$

To get the extinction coefficient ε we need the definite integral of the function over the vertical span of the ray. We obtain ε by taking that integral, and normalizing by the width of the span $z_2 - z_1$, the z_i being the altitude values of the ray endpoints. Substituting z for x in equation (4.1.4) we get:

$$\tau = \frac{1}{z_2 - z_1} \int_{z_1}^{z_2} e^{-z} dz = \frac{1}{z_2 - z_1} \left(-e^{-z_1} + e^{-z_2} \right) \tag{4.1.5}$$

The transparency σ is again given by equation (4.1.1).

Should the difference $z_2 - z_1$ be small enough to risk floating point division problems (as in the case of horizontal and nearly-horizontal rays), we assume that the difference in density between z_1 and z_2 is insignificant and substitute the expression:

$$\varepsilon = e^{-z_1} \tag{4.1.6}$$

Note that we could also use the mean of z_1 and z_2:

$$\varepsilon = e^{-\left[\frac{z_1 + z_2}{2}\right]} \tag{4.1.7}$$

but in practice we have found this unnecessary.

Plates 4.1 and 4.2 illustrate the effectiveness of this exponential atmosphere as a scale cue. Again, terrain models are of visually indeterminate size without atmospheric effects; addition of the mist effectively gives the visual impression of the grand scale intended. The realism of the atmospheric effect can be further enhanced by the inclusion of our Rayleigh scattering approximation, which makes the more distant peaks appear bluish in Plates 4.1 and 4.2.

4.1.6 Radial Fog or Planetary Atmosphere

Again, optical paths in Nature are limited by curvature of the atmosphere around the planet; the above models do not account for this. We now describe a model which does.

4.1.6.1. Motivation

The two atmospheric functions described above require that rays which miss everything in the scene be limited to a reasonably short path length, lest the atmosphere integrate to complete opacity, thus rendering the background in the fog's intrinsic color (e.g., white). The path length of stray rays can be limited by placing a limiting object such as a vertical plane behind the scene, or by varying the ray tracer's definition of the default path length of a ray which fails to intersect any object, but alas, both of those solutions are no more than overt kludges.

In Nature, integration of atmospheric effects along the path of a ray proceeding into space is limited by both the finite thickness of the atmosphere and the atmosphere's curvature around the earth. As in real life, we want the atmosphere to wrap around the sphere of the planet, to limit optical path lengths through the atmosphere without imposing some arbitrary limit to the length of "stray" rays which intersect nothing. A geometrically-correct atmosphere model will not only automatically limit the integration of atmospheric effects, it will additionally grant the possibility of accurately rendering planetary models from arbitrary points of view -- a useful capability for animations. (In fact, our entire foray into planetary models was motivated by the need for a "global" context for fly-bys of existing synthetic landscape models. [91,106,108]) Furthermore, such a function could be used to model a more realistic distribution of aerosols for a physical model of atmospheric scattering such as Klassen's. [64]

To realize this radial atmosphere we need to integrate, along an arbitrary line segment, a function that falls off exponentially by distance from a given point in space. We now describe such a function and the derivation of its integral.

4.1.6.2. Density Profile of Atmosphere by Radius

Consider an ideal gas-planet: a gravitationally-bound collection of gas of homogeneous composition and temperature, without internal structure or motion, in an equilibrium state where gravitational collapse is countered solely by internal pressure of the gas. The density γ of such an object might vary something like

$$\gamma(r) = e^{-r^2}, \tag{4.1.8}$$

that is, as an exponential function of radius r from the center of the planet. Figure 4.1.1 illustrates the graph of this function.

Figure 4.1.1: Graph of the function $density = \gamma(r) = e^{-r^2}$

Note that the slope is zero at $r = 0$; this is because the net gravitational force goes to zero at the center of the planet, as the gravitational attraction of all the mass in the object is exactly balanced at that point in space.

4.1.6.3. Density Profile Along an Arbitrary Line

Consider an arbitrary line $\bar{l}$, in relation to the center of the planet $\vec{c}$. The line and the point determine a plane, thus the analysis of density profile may proceed in two dimensions, which simplifies our problem (see Figure 4.1.2).

Figure 4.1.2: Geometric construction of the atmospheric function.

The density profile along this line will be similar to that in Figure 4.1.1: The density will increase up to the *point of closest approach* ($\vec{p}$), i.e., the intersection of the line with its perpendicular through the planet center. At that point, the rate of change of density will be zero, as in Figure 4.1.1. The exact value of the density at $\vec{p}$ is $\gamma = (r_p)$, where r_p is the distance from $\vec{p}$ to $\vec{c}$:

$$r_p = \|\vec{p} - \vec{v}\| = \sqrt{(\vec{p} - \vec{v}) \cdot (\vec{p} - \vec{v})} \tag{4.1.9}$$

The density will fall off exponentially with distance from $\vec{p}$ along the line $\bar{l}$; density along $\bar{l}$ will properly vary as:

$$\Gamma_{\bar{l}} = \gamma(\|\vec{t} - \vec{c}\|) \tag{4.1.10}$$

for $\vec{t} \in \bar{l}$.

For our purposes, we will approximate the density distribution $\Gamma_{\bar{l}}$ along $\bar{l}$ with the function $\Gamma_{\bar{l}}$, defined as:

$$\Gamma_{\bar{l}}(\vec{t}) = \frac{\gamma(r_p)\gamma(\|\vec{t} - \vec{p}\|)}{\gamma(0)} = \frac{\gamma(r_p)\gamma(r)}{\gamma(0)} \tag{4.1.11}$$

That is, we use the density distribution shown in Figure 4.1.1 (i.e., that of a line through the center of the planet) scaled by the normalized value at $\vec{p}$.

4.1.6.4. The Integrating the Atmosphere Function

What we require is the integral of the function $\Gamma_{\bar{l}}$. Unfortunately, there exists no closed-form solution to the integral of the exponential function in equation (4.1.8). The C and FORTRAN math libraries do, however, contain implementations of the so-called "error function" which supplies a numerical approximation to the semi-definite integral of the function e^{-t^2}:

$$\texttt{erf}(x) \cong \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt. \tag{4.1.12}$$

We can then approximate the definite integral

$$\tau = \int_{\vec{t}_1}^{\vec{t}_2} \Gamma_{\bar{l}} \cong e^{-r_p^2} \left[erf(\|\vec{t}_2 - \vec{c}\|) - erf(\|\vec{t}_1 - \vec{c}\|) \right] \tag{4.1.13}$$

to get the average value of the extinction coefficient ε along an arbitrary optical path (the optical path always being a finite line segment). The transparency σ is again given by equation (4.1.1).

There is a numerical problem with this model which indicates the use of the math library function `erfc()` rather than the function `erf()`, `erfc()` being defined to be 1.0 – `erf()`. This is because of the difference between two calls to `erf()` taken in the right hand side of equation (4.1.13). For large values of $\|\vec{t} - \vec{c}\|$, this difference may lead to *catastrophic cancellation*. [52] Problems with catastrophic cancellation have been observed in practice, when the atmosphere has been scaled down to be very thin, as is the Earth's atmosphere. In this situation, numerical underflow occurs around the area where we are looking straight down on the planet (e.g., in a circle concentric with the projected disk of the planet on the screen). The net effect is the creation of a sharp-edged "ozone hole" where the atmosphere simply (and discontinuously) disappears in an circle around the center of the planet. While the use of `erfc()` does not completely prevent this event, it does allow the atmosphere to be made thinner than when using `erf()`, before the "ozone hole" appears.

4.1.6.5. Atmospheric umbra

The radial fog model presented above is isotropic, from the point atmospheric origin $\vec{c}$. Thus when used in a quarter-lit rendering, the atmosphere has the same lightness above the night side of the planet as above the daylight. This effect is worse, in practice, than it might at first seem: the perceptual effect of light/dark contrast creates the net effect of an atmosphere that appears lighter on the night side of the planet than on the daylight side. (Note that in Plate 4.5 the atmosphere appears a bit lighter near the poles than at the equator on the sunlit side -- this is the same effect in action.)

What we need, then, is a simulation of the planet's shadow in its own atmosphere: a "planetary umbra" or shadow in the atmosphere. Fortunately, this is easily simulated by fitting a cylinder of about the same radius as the planet, to the planet at the terminator (the line of transition from day to night on the planet surface; i.e., where the sunlight is exactly tangent to the planet's spherical surface). Assuming a light source at infinity, the

terminator is a great circle on the planetary sphere, lying in the plane passing through the center of the planet and perpendicular to the direction to the light source.

The umbra cylinder is placed where that the shadow of the planet would propagate, i.e., to the dark side of the planet. The cylinder is then assigned a special 'texture' or surface attribute which simply toggles the radial atmosphere function to a different set of color and density parameters, which are appropriate to a nighttime atmosphere. The net effect is illustrated in Plate 4.5. While this model ignores the penumbra of an extended light source, it is simple, inexpensive, and an acceptable first approximation. Jittering the placement of the cylinder with the multiple samples of a simulated area light source could be used to get a Monte Carlo approximation of a penumbra, as simple extension to the repertoire of "distributed" ray tracing. [25]

Note that the cylinder radius should be that of the planet plus a small epsilon. This is because of the standard epsilon offset applied to refracted rays to prevent floating point imprecision from causing immediate reintersection with the object from which they are departing. If the umbra cylinder and the planet sphere are closer together than this epsilon, refracted rays can miss the planet surface and proceed inside of the planet. This generally leads to the appearance of a light-colored line along the terminator, which is quite undesirable. Close inspection of Plate 4.5 reveals this artifact in practice; note that adaptive antialiasing has not caused it to disappear.

Another solution to the umbra problem is illustrated in Plate 4.6. What we have done here is to "shade" the atmosphere itself with a Lambertian reflection model. That is, we've made the atmosphere darken as a function of the cosine of the angle its "normal" (i.e., the vector from $\bar{c}$ to $\bar{p}$ for a given ray) makes with the light direction. We will expand upon this at the end of the next section.

4.1.7 A Model of Rayleigh Scattering

There are two main types of atmospheric scattering: Rayleigh and Mie scattering. Mie scattering is frequency-independent, highly directional scattering by aerosol particles much bigger than the wavelength of visible light. [85] Rayleigh scattering is frequency-dependent scattering of light by aerosol particles of size smaller than or roughly equal to, the wavelengths of visible light.

The mathematical formulation of the Mie scattering model may intractably complex for procedural image synthesis purposes [65], but this is of little consequence as directional scattering of white light can be adequately modelled with a simple implementation of the phase function; Blinn [12] presented a generalized phase function suitable for simulating Mie scattering.

The empirical scattering model developed by Lord Rayleigh in the nineteenth century indicates that scattering is proportional to the fourth power of the frequency. [148] That is to say, blue light is scattered more efficiently that green and red light; this is the cause of the blue color of the sky. Let us return to equation (4.1.2), which calculates illumination sample I as a function of the value calculated by the surface shader at the end point of the ray, I_s, and the contribution of the atmosphere along the ray path:

$$I = \sigma I_s + (1 - \sigma) I_a \tag{4.1.2}$$

and recall that we said that I_a is simply an rgb or frequency vector-valued color for the fog. Equation (4.1.1) indicated that the transparency σ of the fog is an exponential function of the optical depth $\tau = \varepsilon\delta$:

$$\sigma(\varepsilon, \delta) = e^{-\varepsilon\delta} = e^{-\tau} \tag{4.1.1}$$

There was an unstated assumption that σ was invariant for each of the colors in the vector I_a. Note that the calculation in equation (4.1.2) must be performed separately for

each color component of the color vectors I_a and I_s, that is, at least once for each of r, g and b. But we can make the extinction coefficient ε a vector-valued quantity $\vec{\varepsilon}$, i.e., we can have different extinction coefficients for different colors. Then we still integrate to the given fog color I_a at the limit, but with differing rates of replacement for red, green and blue. Particularly, by increasing the extinction coefficient for the blue component, we can get a nice *ad-hoc* approximation to Rayleigh scattering, modulo the phase function (which describes non-isotropic scattering as a function of angle to the lighting direction).

Figure 4.1.3: Differing extinction coefficients for r, g, and b in Rayleigh approximation.

Plate 4.3 shows the scattering model applied in a planetary atmosphere. Note that the atmosphere appears blue at the edges and pale at the horizon due to scattering, and yellows the (intrinsically white) snow caps by extinction. The scattering/extinction color dichotomy of the scattering model is best seen in the vicinity of the earth/moon intersection (Plate 4.4): note how the atmosphere gracefully transitions from blue (due to Rayleigh scattering) against the black background of space, to yellow-orange (due to extinction) against the pale backdrop of the moon. In this image the underlying color of the fog is a smoggy orange-yellow, to provide the visual impression of Rayleigh extinction, and we use an rgb extinction coefficient vector with component ratios of

approximately $\vec{\varepsilon} \propto (1{:}3{:}20)$ which would be appropriate for fourth powers of red, green and blue primaries with (the rather exaggerated) wavelengths of $800, 500$ and 380 nanometers, respectively. Note that we have not used separate values of $\vec{\varepsilon}$ for scattering and extinction; the effect is achieved with one atmosphere color (orange-yellow) and a single $\vec{\varepsilon}$ tuned to emulate Rayleigh scattering.

We have found this atmospheric scattering approximation useful in most of our more down-to-earth landscape renderings, as they generally include atmospheric haze to provide depth cueing and realism - again, note the bluish cast on the distant peaks in Plates 4.1 and 4.2. The added cost of the scattering approximation is only two additional calls to the `exp()` function, and it adds significantly to the overall realism of the image.

Plate 4.6 shows yet another enhancement to the scattering model. What we have done here is to vary I_a and $\vec{\varepsilon}$ with orientation to the light source - a sort of Lambertian shading applied to the scattering parameters - to get a better approximation to the changes in color of the atmosphere by distance-travelled-through-the-atmosphere of the sunlight. Thus the atmosphere color I_a is a pale grey near the "high noon" areas, fading to red near the terminator. The extinction coefficient $\vec{\varepsilon}$ rolls off the high value for blue near the terminator, to avoid darkening the backdrop by filtering with an atmosphere color I_a which is deficient in blue.

Note that the umbra construct seen in Plate 4.5 is neither needed nor used in Plate 4.6, as the atmospheric density goes to zero on the night side of the planet. Inclusion of a phase function, to improve the Rayleigh scattering approximation and/or to provide a Mie scattering approximation, is similarly simple.

4.1.8 Conclusions

We have shown that the general behaviors of Nature which are responsible for the effects causing atmospheric perspective may be emulated with a few relatively simple

and efficient models. The geometric models of aerosol density distributions may be used in more accurate physical models of scattering. A truly accurate model of scattering should solve for the equilibrium state of radiative energy transfer, as in radiosity models. Such a solution might be had in less-than-geologic computation time by using a volumetric version of the Greengard-Rokhlin algorithm [42], as described for surfaces by Hanrahan et al. [46] Short of such an ambitious solution, the models presented here suffice for the purposes of visual realism.

The models presented here may also be used for other purposes. In Plate 4.7 we see the radial fog employed in the service of visualization. The DLA (diffusion-limited aggregation) depicted here is a very complex object. In an ordinary rendering, where the 3-D object is simply projected onto a 2-D image plane, the resulting image has a visual complexity which renders it rather indecipherable: it is hard to tell what is in front, and what is behind. The radial fog centered on the DLA helps to disambiguate foreground from background, and serves (perhaps perversely) as the only illumination in this rendering-with-no-light-source. In Plate 4.8 we see the same atmospheric model used artistically. In this case, we have found a use other than emulating Rayleigh scattering for the independent extinction coefficients for red, green, and blue: by carefully assigning the values of the fog color and these coefficients, we have obtained a visual approximation of a range of black body temperatures, as if the cloud of gas were glowing white-hot at the center and cooling to a cherry red towards the edges. It is to be hoped that these models can find other such creative uses.

4.2 A Physical Model of Refraction and the Rainbow

In this section we present our models of dispersion and rainbows. The dispersion model was developed for, and presented in, the author's Masters thesis. [101] It was subsequently employed in the development of the physical model of the rainbow presented here. [98] To properly motivate that result, the dispersion model is described below.

4.2.1 Introduction

Treatment of refraction in computer graphics generally lacks *dispersion*, or the spreading of refracted light into its component colors or *spectrum*. While convincing simulations of transparent objects can be had without taking dispersion into account, the inclusion of dispersion makes available additional realism and beauty. We will present a dispersion model, within the ray tracing paradigm, and develop a physical model of the rainbow based on that dispersion model.

Modelling of dispersion entails the solution of at least two distinct problems: the sampling and reconstruction of the power spectrum of light by frequency, and the display of the spectrum of *monochromatic* colors on a standard graphics display device. The first problem may be treated as another aspect of the distributed ray tracing model of Cook et al [25] or as an enhancement to Kajiya's *rendering equation*. [56] The problem of reproducing monochromatic colors is in the realm of color science [44,163] and an approximate solution can be had through the use of *metamers*, though this problem remains an open area of research.

Perhaps the most striking example of dispersion at work in Nature is the rainbow. The arc of the rainbow is a result of the geometry of the reflection and refraction of light in raindrops; the wonderful colors of the rainbow are the result of dispersion of sunlight in refraction through water. With a working dispersion model and some geometric

optics, we can produce an efficient rainbow model for use in ray traced and Z-buffered rendering schemes. We will present two rainbow models, one empirical or impressionistic [32] and another purely physical and thereby quite true to Nature.

4.2.2 Problem Statement

The Cook-Torrance [26] shading model takes into account the frequency of light waves in reflection from surfaces, as a function of the index of refraction. What has been missing from the generally available literature is a model of refraction which takes into account the frequency of light. Such a dispersion model has been called for in previous research. [56,71] Some dispersion models have apparently been developed, but not published. [33,159] Thomas [150] published a brief description of a dispersion model, but did not develop atmospheric rainbows; unfortunately, Thomas' article remains obscure. The work presented here was developed independently of Thomas, and differs in most important respects.

The model of dispersion developed here is an extension of distributed ray tracing [158] and thereby uses the Monte Carlo integration techniques of Cook. [23] Integration of a continuous function by a finite number of point samples can lead to two types of aliasing, that of the frequency content of the signal being sampled and that introduced in the reconstruction of the signal from the samples. It is important to note that we are not concerned with the former type of aliasing, which is the result of sampling the signal at a rate below the Nyquist limit. Color *metamerism* generally obviates the need for very accurate reproduction of the exact curve of the power spectrum; nuances of the power distribution are important only in the interaction of light with attenuating media and reflecting surfaces and can safely be ignored in our model.

What *is* important is our reconstruction of the spectrum from the point samples taken. As our approximation of the integral of the power spectrum will be a set of discrete

samples, our reconstruction will be prone to appearing as a set of discrete, overlapping colors. This situation is analogous to that of *temporal aliasing*, where a moving ball may be sampled (imaged) at several points in time in an attempt to get motion blur and, upon reconstruction, appear as several overlapping, translucent circles. In the case of dispersion, if we were to render a white disk on a black background through a prism, we might see several overlapping disks of different colors. We call this effect *spectral aliasing*, and use the jittering technique of stochastic sampling, or Monte Carlo integration, to defeat it. Jittering is random placement of the actual sample points within fixed sample intervals, which intervals may themselves be regularly spaced. Jittering adds noise to the image and turns the distinct overlapping images into a speckled blur, which looks a bit like spray paint.

The advantage of this noisy reconstruction of the image is that the human visual system tends to blur the noise together into a smooth continuum, whereas it actually enhances the sharp edges in the non-noisy images for a most displeasing effect. Such sharp discontinuities in intensity or color, or the rate of change thereof, manifest the phenomenon known as *Mach banding*. Mach bands are an artifact of the edge-enhancement caused by *lateral inhibition* in the retina. [27] When constructing and sampling our representation of the spectrum we must be aware of the potential for trouble with color Mach banding. The practical significance of this problem will be addressed in section 4.2.4.1.

Whatever colors we choose for representation, we will fail to accurately reproduce the spectrum of monochromatic colors of light. The graphics monitor has three primary colors with which to work, none of which is fully *saturated*. Even if we have three fully saturated or monochromatic primaries (as are available with laser raster projection systems), all other monochromatic colors can only be approximated, with varying degrees of *desaturation*. Our task, then, is to represent the entire visible spectrum of

monochromatic colors as best we can, using three desaturated primaries and avoiding Mach bands. Furthermore, the sum of the samples chosen to represent the spectrum must, at full intensity, be the value of full-intensity white. If not, image samples involving dispersion will be tinted and/or shifted in intensity.

Given a working model of dispersion and an acceptable representation of the spectrum, one looks for applications. One striking application is a physical model of the rainbow. Rainbows are the result of the interaction of sunlight with very large numbers of raindrops in the atmosphere. The sheer number of particles (raindrops) involved, multiplied by the number of samples required to integrate the spectrum, makes a direct simulation of nature quite impractical. By modelling of the interaction of light with a single ideal raindrop, we can acquire a table of data which represents the situation in nature. This table may be used subsequently in the rendering process to replicate the effects of a rainbow in nature, with very good computational efficiency. We will describe such an approach in Section 4.2.3.

4.2.3 Previous Work

4.2.3.1. Physics of Refraction

Refraction is an effect of the differing speed of light in dissimilar materials. The speed of light in a material determines its *optical density* which, surprisingly, is not exactly proportional to its mass density. As light slows down upon entering a medium of greater optical density, the wave trains are compressed. Thus, while frequency is preserved, wavelength is not. It thereby behooves one to be careful not to use "frequency" and "wavelength" interchangeably when discussing refraction and dispersion.

The *angle of refraction*, or the angle of the change in path for light, was related mathematically to the net change in index of refraction by Willebrord Snell in 1621 (and,

independently, by René Descartes at nearly the same time). It is codified in *Snell's Law* [9,125]:

$$\eta_1 \sin\theta_i = \eta_2 \sin\theta_t \tag{4.2.1}$$

where η_1 and η_2 are the indices of refraction of the two transmissive media, θ_i is the angle of incidence and θ_t is the angle of transmission. As the refractive index η is a function of the frequency of the light ray, the angle of refraction is also a function of frequency. Thus arises dispersion.

4.2.3.2. Physics of Dispersion

The proportion of change of index of refraction with frequency in a material is termed *dispersive power*. The dispersive power w of a material is defined as the ratio of the dispersion between the F and C Fraunhofer lines* to the mean deviation, i.e., the deviation for the D Fraunhofer line. [139,143,161] Thus

$$w = \frac{\eta_F - \eta_C}{\eta_D - 1} \tag{4.2.2}$$

where η_F, η_C, and η_D are the refractive indices of the material at the frequencies of the F, C, and D Fraunhofer lines, respectively. In the optical industry, the reciprocal of the dispersive power ν (the "ν-value" or "Abbe number") is more commonly used [9]:

$$\nu = \frac{1}{w} = \frac{\eta_D - 1}{\eta_F - \eta_C} \tag{4.2.3}$$

The Abbe number can range from about 16, for methylene iodide, to over 95, for calcium fluoride. (The respective dispersive powers are approximately 0.06 to 0.01.) For optical

---

* The Fraunhofer lines are *emission lines* of hydrogen. They represent monochromatic light at various visible wavelengths: the C line is at 656.3 nm (red), D is at 589.3 nm (yellow), and F is at 486.1 nm (violet).

glasses the Abbe number can range from 19.7, for the densest silicate flint glass, to 70.0, for light phosphate crown glass. The Abbe number for water is 55.7.

Just as optical density is independent of mass density, dispersive power is independent of optical density. The reason is that dispersion is modulated by *absorption bands* in materials, not by optical density.

Figure 4.2.1 The dispersion curve at an absorption band.

Note also that the plot of refractive index vs. frequency is not perfectly straight, but curved. This is an important factor in the development of a model of dispersion.

There have been many attempts to formulate a quantitative relation of refractive index η to frequency or wavelength λ, none entirely successful. The best known and most general is that of Sellmeier [9]:

$$\eta^2 = 1 + \sum \frac{b\lambda^2}{c^2 - \lambda^2} \tag{4.2.3}$$

where b is a constant characteristic of the material, c is an idealized absorption wavelength of the material (corresponding to a spectral absorption band) where the index of refraction is infinite, and the summation is over all absorption bands in the material. Simpler equations which are suitable for limited extents within the spectrum are [9]:

$$\eta = \frac{a}{\lambda^0} + \frac{b}{\lambda^2} + \frac{c}{\lambda^4} + \ldots \qquad \text{(Cauchy)}$$

$$\eta = 1 + \frac{b}{c - \lambda} \qquad \text{(Hartmann)}$$

$$\eta = a + \frac{b}{\lambda} + \frac{c}{\lambda^{\frac{7}{2}}} \qquad \text{(Conrady)}$$

$$\eta = a + b\lambda^2 + cL + dL^2 \qquad \text{(Hertzberger)}$$

where $L = \left(\lambda^2 - 0.028\right)^{-1}$, and a, b, c, and d are constants. These equations are all nonlinear, and values of the constants for various materials are not easily found in the literature. This will be a consideration in our development of a dispersion model.

4.2.3.3. Rainbows

René Descartes worked out the first physically accurate model of the rainbow in 1637. [43,88] To do this, he assumed the raindrops to be spherical and traced rays through a circular, two dimensional representation - proof that ray tracing is hardly a new technique. Descartes' simulation is illustrated in Figure 4.2.2.

With his simulation, Descartes was able to accurately explain the angular size and position of the primary rainbow arc and some of the supernumerary arcs. (The supernumerary arcs which sometimes appear immediately inside of the primary rainbow arc are due to *diffraction* effects arising from the wave nature of light, and thus cannot be modelled using the geometric optics of a particle transport ray tracing paradigm. For more on this topic, see Nussenzveig. [112]) Interestingly, an explanation for the color in the rainbow had to await Newton's discovery of dispersion some decades later. Aside from the supernumerary arcs inside the primary rainbow arc, Descartes' raindrop remains an accurate and sufficient model of the rainbow.

To recreate Descartes' simulation, we trace rays into the raindrop from the optical axis (ray 1 in Figure 4.2.2) to the edge of the circle. This corresponds to a range of zero to one for the *impact parameter*; the value of this impact parameter uniquely determines the path of the ray through the raindrop. Upon impinging the raindrop, the ray is refracted,

Figure 4.2.2 Descartes' raindrop.

reflected once for the primary arc or twice for the secondary arc, and refracted again upon exiting the drop. Arcs formed by higher-order internal reflections are deemed unimportant as they are too dim and/or appear close to the sun in the sky, and are therefore not visible in Nature.

Note that all rays with an impact parameter greater than or less than that of ray 7 in Figure 4.2.2, the *Descartes ray*, emerge at an angle closer to the optical axis than that ray. Thus the Descartes ray marks a point where the rate of change of emergence angle with impact parameter is zero, and there is a concentration of light energy being returned at this angle, which is approximately 42 degrees. This gives us a bright feature 42 degrees from the optical axis; it is dispersion which spreads the bright feature into the spectrum of colors. Note also that the fact that all rays which are reflected exactly once inside the raindrop emerge at 42 degrees or less, makes the sky appear lighter inside of the primary arc of the rainbow. Rays reflected exactly twice inside the raindrop emerge with a peak power at approximately 52 degrees, with the excess light emerging at greater angles. Thus the secondary arc appears at about 52 degrees; between the two arcs is a zone of darkness known as Alexander's band. Alexander's band occurs naturally in our simulation: see Plates 2.5, 4.12, and 4.13.

To perform an accurate simulation of energy transfer in Descartes' raindrop, the Fresnel equation should be used to modulate the quantities of reflected and refracted energy. With an *extinction coefficient*[*] of 0, the Fresnel equation for reflection can be written [14]:

---

[*] The extinction coefficient is a physical quantity specific to each material [144] which varies with frequency. The specific values of this coefficient are often unknown for a given material, and it is generally set to 0, for the purposes of computer graphics lighting models.

$$r_{\parallel} = \frac{\eta_2 \cos\theta_i - \eta_1 \cos\theta_t}{\eta_2 \cos\theta_i + \eta_1 \cos\theta_t} \tag{4.2.5}$$

$$r_{\perp} = \frac{\eta_1 \cos\theta_i - \eta_2 \cos\theta_t}{\eta_1 \cos\theta_i + \eta_2 \cos\theta_t} \tag{4.2.6}$$

$$R = \frac{r_{\parallel}^2 + r_{\perp}^2}{2} \tag{4.2.7}$$

where $r_{\parallel}$ is the reflection coefficient for the component of light which is polarized parallel to the surface, $r_{\perp}$ is the reflection coefficient for the component polarized perpendicular to the surface, η_1 and η_2 are the refractive indices of the two materials, θ_i is the angle of incidence, θ_t is the angle of refraction, and R is the total reflectivity. Light not reflected is refracted in quantity $1 - R$.

Figure 4.2.3 The cone of a rainbow.

The rainbow phenomenon exists as a cone in space which is unique for each point of view (and indeed for each eye of the individual observer); Figure 4.2.3 is intended to illuminate this. Inspect it carefully for the following argument. Since the geometry of reflection and refraction as discussed above gives us a spectrum appearing at an angle the same as that of the Descartes ray from the straight back direction to the light source, we would expect to see that spectrum in all (sunlit) raindrops viewed from that angle. The sun's rays can be assumed to be parallel, thus this effect appears to the observer as a circle of angular radius 42 degrees, since the observer is, by definition, at the apex of the cone.

Naturally occurring rainbows actually constitute a cone of half-angle 42 degrees around the *antisolar point* and have an angular width of approximately 2 degrees. The secondary arc appears at a half-angle of 52 degrees. Plate 4.11 demonstrates that, in the absence of a landscape which clips the bottom half (at least) of the rainbow, it appears as concentric circles around the antisolar ray.

4.2.3.4. Computer Graphics

As mentioned above, the Cook-Torrance shading model relates reflection to index of refraction and frequency through the Fresnel equation. [144] A model of refraction relating index of refraction to frequency has been developed by Thomas [150] and more recently by the author [101,102]; that work is extended here to include a physical model of the rainbow. [98] The problem of integration and reconstruction using point samples has been addressed by Cook [23] in his discussion of the distributed ray tracing model. [25] The dispersion model developed by the author is a straightforward application of Cook's techniques, as an extension to the repertoire of effects available through distributed ray tracing. A model of atmospheric rainbows has been alluded to in the literature [22] and demonstrated in an image [24], but was not described. It was not, however, derived through a physical simulation. [21] A physical model of the rainbow requires a fair amount of development work. Fortunately, the simulation need only be run once; the results may then be included a rendering program or programs for all future imaging applications.

4.2.4 Solution

We now describe our solutions to the problems of modelling dispersion and the rainbow.

4.2.4.1. Sampling in the Frequency Domain of Light

To model dispersion, we must integrate the power spectrum of light at each sample point in the image where there occurs dispersive refraction, such as on the surface of a glass prism. The integral of the power spectrum can be expressed:

$$I_T = \int_{380}^{800} I(\lambda)\, d\lambda \tag{4.2.8}$$

where I_T is the total illuminance at the given point in space and $I(\lambda)$ is the illuminance at wavelength λ at that point. As we need only integrate the power spectrum of *transmitted* light at dispersive surfaces, since only transmitted or refracted light is dispersed, the integral we are interested in can be stated

$$I_T = \int_{380}^{800} T(\lambda)\, d\lambda \tag{4.2.9}$$

where I_t is now the illuminance by transmitted light at a point in space on the boundary of a change in refractive index, and $T(\lambda)$ is the illuminance by the transmitted light at wavelength λ.

As previously stated, we will approximate this integral using a set of point samples. We perform stochastic antialiasing of our integral by jittering [23] the samples. If a sample f at frequency λ represents the power in the spectrum over an interval of width Δf, the jittering consists of adding a random offset $\Delta f(X - 0.5)$ where X is a random variable of uniform distribution in the range $[0,1]$. The net effect is to randomly place the sample f somewhere within the interval $\left[\lambda - \frac{\Delta f}{2}, \lambda + \frac{\Delta f}{2}\right]$.The fact that we take point samples in the frequency continuum of light implies that we are also taking point samples of the continuum of the dispersion curve, as index of refraction is a function of frequency. Thus we face the choice of whether to jitter the frequency (and therefore the color) of the rays or the refractive index of the material, or both. Given that the jittered sample at

frequency f needs to be translated into $R(f)$, the value of the refractive index function R at f, we will prefer to jitter a linear function R over a nonlinear function for reasons of computational efficiency, as linear interpolation is in general quicker to evaluate than nonlinear interpolation.

This may motivate us to contrive piecewise linear approximations to the spectrum and the dispersion curve. It is unlikely that the viewer of the final image will be able to discriminate between a physically accurate nonlinear model and a computationally efficient linear approximation; furthermore, since the dispersion curve is specific to a given material, to be true to nature one would need to tabulate data for every distinct material to be rendered. We therefore employ a linear approximation to the dispersion curve for our rendering dispersion model.

The refractive index and dispersive power for surfaces can be input parameters. Thus one can specify a polygon with an associated refractive index of, for example, 4.2 and a dispersive power of perhaps 0.5, both of which are outlandish in terms of the "real" world, but viable within our model. It is interesting to create situations and materials which cannot exist in our everyday experience; this is part of the power of computer graphics.

The issue of which quantity to jitter, refractive index or color, or both, should be evaluated in the light of computational efficiency. The reason for jittering samples is to avoid spectral aliasing, however, it has been our experience that spectral aliasing is not a significant problem in any but deliberately pathological scenes. That is, the distinct overlapping images of different colors are simply not readily visible unless the dispersive power is unrealistically high. When jittering is deemed desirable, we jitter the frequency of the ray and derive, in a pre-rendering operation, a constant c_s for each refractive surface s in the scene:

$$c_s = \frac{w(\eta - 1) - \eta}{0.76} \tag{4.2.9}$$

where w is the dispersive power, η is the refractive index at the far red end of the spectrum, 0.76 is the proportion of the visible spectrum that lies between the C and F Fraunhofer lines. This constant c_s, when multiplied by the frequency of a sample, gives the refractive index at that frequency for use in calculations of propagation of refracted light. (Note that this assumes that frequency is specified in the range $[0,1]$.) Thus the cost of jittering is reduced to one floating point multiplication per surface encountered, plus the negligible preprocessing cost of evaluating c_s for each relevant object in the scene and the cost of interpolating the color of the final sample.

4.2.4.2. Representing the Spectrum

To reproduce the spectrum, we must simulate the entire gamut of monochromatic colors using only the three desaturated primaries of the graphics monitor. Furthermore, the integral of each of the red, green, and blue curves of our simulated spectrum must be unity, or the reconstruction of an image from our samples will be tinted, darkened, or overdriven. We refer to this as the *summing to white* criterion. As we work within the *rgb* color space, we should restate equation (9) in terms of the rgb vectors:

$$I_{t_R} = \int_{380}^{800} R(\lambda) T(\lambda)\, d\lambda \tag{4.2.11}$$

$$I_{t_G} = \int_{380}^{800} G(\lambda) T(\lambda)\, d\lambda \tag{4.2.12}$$

$$I_{t_B} = \int_{380}^{800} B(\lambda) T(\lambda)\, d\lambda \tag{4.2.13}$$

where $\mathrm{R}(\lambda)$, $\mathrm{G}(\lambda)$, and $\mathrm{B}(\lambda)$ are the values of the R, G, and B *tristimulus* functions for the metameric color used to represent the color of monochromatic light of wavelength λ. [163] When sampling at a particular frequency then, we are actually taking three (red,

green, and blue) samples of $T(\lambda)$. The distribution of the samples should be tailored to the shape of the tristimulus curves used in the representation of the spectrum, with care taken to assure that:

$$\sum_{i=1}^{n} R(\lambda_i) = \sum_{i=1}^{n} G(\lambda_i) = \sum_{i=1}^{n} B(\lambda_i) \tag{4.2.14}$$

where λ_i is the wavelength of the i^{th} sample, and $R(\lambda_i)$, $G(\lambda_i)$, and $B(\lambda_i)$ are the red, green, and blue values, respectively, of sample λ_i. This equality is necessary in order to have the samples (at their maximum intensity values) sum to white in the rgb color space of the graphics monitor.

4.2.4.2.1. Linear Spectrum Model

Figure 4.2.4. The rgb curves of the linear spectral representation.

A simple representation of the spectrum, given these constraints, is shown in Figure 4.2.4. This model has the advantage of being piecewise linear, for fast interpolation of color, and it provides a reasonably good perceptual representation of the spectrum. It has the disadvantage of using a significant portion of the power available to the red primary,

in the approximation of violet with magenta. Violet is of higher frequency than is available with an rgb monitor and therefore cannot be directly reproduced; magenta is a visually acceptable substitute. A problem with the magenta representation of violet is that edges which are blurred by dispersion such that they should appear with the color sequence yellow-orange-red-black, actually appear greenish-yellow-red-black. (See Plate 4.9.) This is because in a white-to-black transition of this sort, the first color to be subtracted out from the sum is violet. When violet is represented as a sum of equal quantities of red and blue, the subtraction of violet leaves a surplus of green. This is a subtle effect, and escapes the notice of most viewers.

Another potential drawback of this representation of the spectrum is the pronounced discontinuities in the first derivative of the rgb curves. While this has the potential for causing color mach banding, such an effect has only been observed in deliberately pathological scenes. Yet another problem found is that the red band in the spectrum appears too narrow, again because some of the red energy is used to display violet. The final problem is that the rolloff of red and violet to black is too steep and short; the entire curve bears no resemblance to the response curve of the human visual system.

Despite the above drawbacks, we have found this to be a viable representation of the visible spectrum.

We sample the representation of the spectrum at 13 intervals centered on the vertical lines in Figure 4.2.4. This provides a good basis for reconstruction of the spectrum and preserves the summing to white property. However, when jittering we encounter the problem that the samples may longer sum to white. The noise added by uncorrelated jittering of the 13 samples will generally skew the sum; in practice this appears as a faint colorful noise, faint enough to not be objectionable or even usually noticeable. (This problem could be defeated by correlating the jittering of the 13 samples to balance total color content, but this is computationally expensive.) Furthermore, about half the time

Figure 4.2.5. The rgb curves of the CIE spectral representation.

the sum of jittered samples of a full intensity white point will exceed unity. If the sum is not clamped to unity at the high end, overflow will occur and the color of the summed samples is likely to wrap around to black. This problem is defeated by clamping the sum, at minimal computational cost.

4.2.4.2.2. CIE Spectrum Model

A more rigorous approach to the construction of the representation of the spectrum involves taking the xyz coordinates of the monochromatic spectral colors in the CIE XYZ color space [163] and performing the appropriate linear transformation into rgb values. Construction of the transformation matrix requires information about the chromaticity coordinates of the specific monitor on which the spectrum is to be displayed. [125] We use as input the xyz coordinates of monochromatic colors weighted by the spectral radiant power distribution of the CIE standard illuminant B, which is designed to emulate direct sunlight (the light source for rainbows). The following graphs are piecewise linear between samples taken at 10 nm (nanometer) intervals from 380 to 770 nm. [163]

As our rgb primaries are not fully saturated, we expect that at all points in the spectrum at least one of the rgb values will be negative. This is indeed what we see in the curves of Figure 4.2.5. The sum of these curves, both with negative values included and with negative values clamped to zero, are shown in Figure 4.2.6. A more accurate approximation to the spectrum, without negative values, could be attained be limiting the xyz input values to the color gamut of the monitor.

Note that the curves in Figure 4.2.6 have a local minimum in the cyan area of the spectrum. These curves do not give an acceptable representation of the spectrum on a monitor calibrated for perceptually linear contrast response; the cyan and yellow colors appear far too dark. When adjusted with a gamma correction of 2.5 to 3.0, the zero-clamped curve gives a less objectionable representation of the spectrum, but one which is still far from an accurate representation of what we see in Nature (and not nearly as visually pleasing as the *ad hoc* model described above). Note also that the area under the zero clamped curves should be normalized to meet the summing to white requirement.

It would seem that the only "correct" solution to representing the spectrum, would be to desaturate all of the monochromatic colors (by the same amount) until the horseshoe of such colors on the CIE chromaticity diagram fits within the triangle defined by the

Figure 4.2.6. Summed rgb values, with and without negative values.

chromaticity coordinates of the rgb display device primaries. [125,163] This would lead to a highly desaturated world in the resulting images, and for that reason this theoretically-correct solution is rejected out of hand.

4.2.4.3. Rainbow Models

4.2.4.3.1. Empirical Rainbow Model

We have developed two models of the rainbow, one empirical* and relatively simple, the other comparatively complex and purely physical. The former model entails using the 13 colors of our samples of the linear spectrum model to create 13 different colors of fog which compose a rainbow. The fog function is simply an asymptotic replacement of some percentage r of the color value computed at the end of the ray, with the color value of the fog, based on the distance that the ray has traveled. This is again given by Beer's law [19] :

$$r = e^{-hd/t} \tag{4.2.14}$$

where h is a constant, d is the distance, and t is the *transmittance* constant; note that t has red, green, and blue components, usually equal. As that distance goes to infinity, the percentage of replacement goes to 100. The 13 colored fogs are invoked in concentric rings (cones, actually) around the *antisolar vector*, e.g., the vector from the light source to the eye point. This vector corresponds to the ray from the observer to the antisolar point in Figure 4.2.3. Each ring is a band of some angular width, at some angular offset from the antisolar vector. We construct the rainbow by taking the dot product of each ray traced, with the antisolar vector; this dot product gives us the cosine of the angle between

---

* Interestingly, Fournier contends [33] that this first model is an empirical model, as parameters such as angular width and position are taken from measurements of the rainbow in Nature. The author, in contrast, maintains that "empirical" implies too strong a basis in such measurements, and that model is better classified as impressionistic, i.e., based on a visual impression of Nature. After all, those empirical parameters may be changed arbitrarily by the user in our implementation.

the two vectors. This cosine is then used as an index into a table of the 13 colored fogs. The indexing function can be parameterized to vary the width and angular placement of the rainbow. The following C code segment implements this parameterized rainbow:

```
index = ( DOT(ray_direction, antisolar_ray) - rainbow_angle)
        * rainbow_width;

if (jitter_option)
      index += jitter(delta);
if ((index >= 0) && (index < FREQUENCIES))
      Fog = Rainbow[(int)index];
else Fog = NULL;
```

where ray_direction and `antisolar_ray` are vectors, `Fog` and `Rainbow[]` are pointers to structures for the `fog` type, and the other variables are floating point type. The constant `FREQUENCIES` is equal to 13; the function call `jitter(delta)` returns a random value of uniform distribution in the range `[-delta/2,delta/2]`.

The jitter option turns a rainbow composed of concentric bands of color to a more attractive "fuzzy" rainbow. This jittered rainbow can look fairly realistic, particularly when supersampling is employed to soften the noise introduced by the jittering. (See Plate 4.9.) Note that this scheme only jitters the index to the table of colored fogs, and not the color of the fog itself; an improvement would be to add such color jittering.

4.2.4.3.2. Physical Rainbow Model

The above approach is *ad hoc* and is not really based on a dispersion model, but only uses the spectral representation of our dispersion scheme. A more rigorous and complex approach, yielding a more realistic result, is to recreate Descartes' simulation using dispersion. We will have to integrate Descartes' raindrop over the visible frequencies of light; this entails ray tracing Descartes' raindrop at a variety of frequencies and summing the results. Clearly it is inefficient to ray trace Descartes' raindrop for every ray spawned in the process of rendering a scene; fortunately we can do much better than this. We need only perform the integration over frequency of Descartes' raindrop once, in a

preprocessing step, to build a table of fogs similar to that used in the our simpler rainbow model. This table will need to have a relatively large number of entries, as a real rainbow is an illumination effect that covers most of the sky, though mostly to a very subtle degree. Thus we have entries for a large number of angular displacements, over a 180 degree range. (In practice, no fog might be required in the 10 degree interval of Alexander's band, as no light is returned there by refraction.)

The first step in implementation of the physical model is to generate an algorithm for ray tracing Descartes' raindrop. This means calculating the angle of emergence and energy attenuation factor for rays which are reflected once and twice inside the raindrop, as a function of the impact parameter. The angle of emergence of a given ray is determined by the geometric optics of reflection and refraction in a sphere, while the energy transfer is determined by the physics of reflection and refraction of light as it interacts with air/water boundaries.

The geometric optics of Descartes' raindrop are illustrated in Figure 4.2.7.

Note that we can take advantage of the equality of angles θ_1 and θ_2. Once this geometry is established, it is straightforward to program an algorithm to trace the required rays.

For the purposes of computer graphics, we are generally not concerned with the polarization of light, and generalizations of the Fresnel equation for non-polarized light are usually employed. For this simulation, however, we are more interested in physical veracity than computational efficiency, so we choose the formulation of the equation as it appears in equations (4.2.5-7). Note that the orientation of polarization to the surface is preserved through reflections and refractions in a spherical raindrop.

Also in the interest of physical accuracy, we use a nonlinear approximation of the dispersion curve of water in our rainbow simulation. Using actual measurements of the refractive index of water at various frequencies [60] we derive constants a and b of Cauchy's equation for refractive index, getting $a = 1.3239$ and $b = 3116.3$. The first two elements of the Cauchy series:

Figure 4.2.7 The geometric optics of Descartes' raindrop.

$$\eta = \frac{a}{\lambda^0} + \frac{b}{\lambda^2} = a + \frac{b}{\lambda^2} \tag{4.2.16}$$

give a good approximation to the dispersion curve of water with the derived values of a and b: over the wavelength range from 405 to 670 nanometers, the calculated values of η agree with measured values to within plus or minus 0.0001, or 0.8 of one percent. We use a refractive index of 1.0003 for air.

Our first implementation of the physical rainbow model uses samples taken at 13 fixed, evenly spaced frequencies or wavelengths. (We relax our rigor in the use of "frequency" and "wavelength" here, as the visible spectrum is usually specified by wavelengths of light in a vacuum.) We trace 50,000 rays per wavelength, over the range of impact parameters. For each wavelength sampled, the intensities of the emerging rays are summed by angle of emergence in a linear array of 1800 buckets. The intensities in each bucket are then multiplied by the rgb vector of the representative color for that wavelength and added to buckets of a similar array of rgb intensities by angle. After all wavelengths have been sampled, the results in the rgb array are normalized and inverted for use in the fog function. Unlike the *ad hoc* rainbow model, the fogs used are not themselves colored, but rather their transmittances, t in equation (4.2.15), are unequal in red, green, and blue. Thus the fogs have no intrinsic color, but red, green, and blue values at ray endpoints are replaced at independent rates per unit distance, as in the Rayleigh scattering model of section 4.1.7. This prevents unnecessary filtering by attenuation of colors behind the rainbow, and is appropriate because the rainbow, in Nature, acts by adding color energy along the optical path, not by filtering it out.

Our first approach evidences significant spectral aliasing. Spectral aliasing is accentuated in the rainbow model, as the bright feature at the Descartes ray is quite narrow and pronounced for a point light source, resulting in thin concentric rings of color in the rainbow (see Plate 4.10). The rings are more widely spaced and therefore more

evident in the violet end of the spectrum, as the dispersion curve is steeper at shorter wavelengths.

A second implementation employs spectral antialiasing. Again we sample at 13 distinct frequency intervals, but we jitter the samples within the intervals. This approach requires that we multiply the intensity of the ray by the interpolated rgb value for its specific frequency, and store that vector in the rgb array immediately, rather than using an intermediate storage array, as the colors of individual rays will vary. This has the effect of blurring and merging the rings produced by discrete sampling.

Again, the process described above yields the rainbow produced by a point light source, thus the rings of color produced by spectral aliasing are quite narrow and distinct (Plate 4.10). In nature rainbows are produced by the sun, which has an angular diameter of approximately one half of one degree. Convolution of the final rgb tables with a (one dimensional) kernel which represents the disk of the sun spreads each of the rings over one half a degree of angle. The kernel we use is five entries wide, corresponding to the fact that our fog samples are taken at $1/10^{th}$ degree intervals. Since the entire angular width of the rainbow is approximately two degrees, this blurs the rings together well enough to provide very good spectral antialiasing. See Plate 4.11; compare to Plate 4.10. If the area under the curve of the semicircular kernel is normalized, there will be no net change in the density of the fog tables after the convolution.

Plate 4.12 illustrates another feature of our rainbow models. In nature, rainbows are rarely perfect arcs, in fact one most often sees only a portion of the full rainbow arc. Rainbows are modulated by three factors: the horizon, which clips the bottom portion of the rainbow; the shadows through space of the clouds from which the rain originates; and the spatial distribution of the falling rain itself. In an effort to make our rainbows look more natural, we modulate intensity of the rainbow with a procedural fBm texture (see chapters 2 & 5). This is a texture function which takes a vector as its argument and

returns a stochastic scalar quantity with a $1/f$ power spectrum. The vector we pass to the texture is the ray direction; we use the scalar value returned to modulate the transmittance of the rainbow fogs. The frequency content of the fBm function can be parameterized for varying effects (see chapters 2 and 5), and the texture can be scaled on a vertical or slanted axis to simulate sheets of falling rain.

4.2.5 Conclusion

A model of dispersive refraction within the distributed ray tracing paradigm has been implemented, with good subjective results. The problem of representing the spectrum of monochromatic colors within the rgb color space has been addressed, but not solved to final satisfaction; further work is called for here. Physical and empirical/impressionistic models of the rainbow have been developed, using the above results. In contrast to the dispersion model, the rainbow models are relatively efficient to render, because of their table-lookup implementation. The rainbow models are suitable for Z-buffer rendering schemes, as well as ray tracing.

4.3 A Physical Model of the Mirage

4.3.1 Introduction

In the May, 1990, edition of IEEE Computer Graphics and Applications featured an article "Ray tracing Mirages" by Marc Berger, Terry Trout, and Nancy Levit. [8] It contained a line-drawing figure, and an explanation of it, which we saw as essentially misleading. The misleading illustration was traced back to their reference sources. [30 ,69] Some other small inaccuracies in the article were noted as well. In response, we drafted a letter to the editor. Never expecting the letter to be published, we were astonished when we were asked for permission to use it as a short "research note" article in the same journal. [95]

Having been unexpectedly called to go public with this comment, we felt compelled to "put our money where our mouth is" and develop a mirage model of our own. This lead to the image seen in Plate 1.3, which was developed as a verification test for the model. (It remains, to date, the one and only image we have created using the mirage model.) The image has received wide recognition as an artwork in its own right, having appeared on the cover of IEEE Computer Graphics and Applications [91], as a two-page spread in the Communications of the ACM [63], in both the SIGGRAPH '91 Art Show and the SIGGRAPH '91 Technical Slide Set, and in numerous international publications. It is particularly interesting in that it may be the first example of a mirage appearing in a fine art landscape rendering. [40] As an artwork which was born as an illustration of an original scientific model, it embodies all the best and most significant aspects of this work.

4.3.2 A Comment on "Ray Tracing Mirages"

The article "Ray Tracing Mirages" [8] by Berger, Trout and Levit describes a certain model for mirage formation. This model is based on the existing physics literature

[30,69]; we will refer to it as the Khular/Fabri model below. We would like to make some perhaps picayune points about Boyer et al's interpretation of this model and their description of their implementation of a version of it. These points have important implications for the resulting images.

The Khular/Fabri model asserts that light rays may be caused to follow a parabolic path by a continuous decrease in the index of refraction of air with altitude. The change in refractive index η due to heating of air near the ground is modelled as an exponential function of altitude. In the words of Fabri:

> A light ray forming angle ν with the (vertical) axis undergoes a series of infinitesimal refractions; $\eta \sin \nu$ being always constant... Therefore when $\eta = \eta_A \sin \nu_A$, it follows that $\sin \nu = 1$ and the ray cannot penetrate at lower values of n; it is a concave curve, pointing at both ends towards the positive sense of the (vertical) axis.

What Fabri is pointing out is the phenomenon of *total (internal) reflection* in which light rays which impinge upon a boundary of decrease in refractive index at an angle greater than the so-called *critical angle* are reflected with 100% efficiency. The critical angle is that where Snell's Law [9,125]

$$\eta_1 \sin \theta_i = \eta_2 \sin \theta_t \tag{4.3.1}$$

requires the sine of one the angles to be equal to one; above this value the sine would be required to be greater than one, thus no light is refracted, all is reflected. [161] Unfortunately, Khular, Fabri and Berger et al all have nice illustrations of the resulting ray path which indicate that path is a (continuously differentiable) paraboloid or hyperboloid. Here the mischief is made: while the path of the ray *will* be "a concave curve, pointing at both ends towards the positive sense of the (vertical) axis", it will be a curve that is not differentiable due to the discontinuity in the first derivative at the point of total reflection. The continuous change in the refractive index of the transmissive medium (the air) will indeed cause the path of the ray at either side of this point to follow

a nonlinear curve. The entire path of the ray cannot, however, be modelled as a continuously-differentiable paraboloid curve; it is, in fact, a reflected hyperbolic curve. While the authors appear to have correctly included total reflection in their model (see point R in figure 5 on page 40 and the third sentence in the first paragraph on page 39), they make the misleading statement on page 38:

> As a ray enters the mirage box, it strikes the different air layers, bending at each level and thereby approximating the parabolic ray equation.

Why is this seemingly minor point of any importance? First, if the formation of a mirage were by some continuous deformation in the ray path, one would expect to see a different image than if it were formed by some discontinuous process such as total reflection. Second, if refraction were the primary engine in mirage formation, one would expect to see the effects of dispersion in a mirage, smearing the image into its component colors by a prismatic effect (an important factor in the generation of the famous and elusive "green flash" of the setting sun). This author's subjective experience indicates little-to-no dispersion in mirages observed in nature. Furthermore, as the index of refraction of air at room temperature and pressure is approximately 1.00027, versus 1.0 for a vacuum, there is precious little refractive power to work with via density changes caused by atmospheric heating. By contrast, water has a refractive index of 1.33, glass can range from 1.5 to 1.9. This would indicate that the primary "bending agent" in mirage formation is total reflection, not refraction, which in turn would indicate that a purely reflective model, without refraction, might well be sufficient. This would be a very good thing, as dispersive ray tracing is not particularly easy or inexpensive to implement. [98,150] It would also obviate the statement on page 38:

> The use of layers allows a digitized approach to the continuous spectrum.

This obviation is also a good thing, as this author claims that the problem of representing the spectrum with fully saturated (as opposed to partially desaturated) rbg colors is an open problem in the field of computer graphics [98,101] thus calling into

question schemes which involve taking a number of monochromatic, or tightly band-limited, samples rather than just three samples in each of red, green and blue. A series of monochromatic samples may only be displayed in proper chromatic relation to one another, by substantially desaturating them so that they are reproducible with the color gamut of the display device. [147,163] A highly color-desaturated world is a boring place to render; thus we question the wisdom of frequency-sampling techniques, except when exact simulation of the interaction of color of materials with specific, known spectra is important.

4.3.3 Total Internal Reflection in Ray Tracing

A final note on the article "Ray Tracing Mirages": we hope that it will mark the beginning of the end for a faulty assumption built into many ray tracers, since the very inception of ray tracing as a rendering technique in computer graphics. This assumption is that all rays spawned by total internal reflection should immediately be squelched, their contribution set to that of the background color. This practice of squelching is attributable to Whitted [158] who did this, and published it, in his seminal paper on ray tracing. While the author cannot claim access to the thought processes of Mr. Whitted at the time, we can guess that he made the assumption that this squelching was a good idea because: a) any ray spawned by total internal reflection inside of a sphere will propagate inside that sphere forever by total reflection, b) ray tracing was already a very expensive algorithm to execute on the hardware of the day, therefore excess and non-contributing rays were best culled, and c) once having ray traced glass spheres, people might never want to ray trace anything else, therefore one might as well optimize the algorithm for spheres.

This author has reaped some unpleasant artifacts from Whitted's assumption - see Plate 4.9 where the bottom of the (dispersively refracting) prism, which could be (totally) reflecting something in the scene, is rendered in dark blue, the background color. We

therefore advocate the abolition of this banishment of totally reflected rays in ray tracing programs. If the potential performance cost it in ray tracing glass spheres is irksome, we suggest quashing totally reflected rays only when they originate inside of spheres.

4.4 Chapter Conclusion

Atmospheric effects are essential to the realistic rendering of terrain models. The complexity of photon propagation in such participating media is a much harder problem than we dare attempt to solve, in the context of production image generation. For such purposes, we require efficient models which reflect the gross behavior of Nature fairly well, in subjective visual assessment. We have developed and demonstrated the visual efficacy of, such models.

The atmospheric effect of chief importance is aerial perspective, as it is a critical distance cue in landscape renderings. Modelling aerial perspective requires geometric models of the spatial distribution of scattering aerosols and a model of frequency-dependent scattering and extinction. We have demonstrated novel and effective models of both.

Not so essential, but nevertheless of interest, are models of the rainbow and mirage. We have shown models of these effects as well. Variations on the rainbow model may be used to simulate other effects, such as halos, glories, sun dogs, and other spectral phenomena caused by transparent particles (such as ice crystals) suspended in the atmosphere, in future research.

As most of our models are designed to be simple expedients to gain first approximations to desired effects, it is not surprising that the area of modelling atmospheric effects has much room for fruitful future research. The true complexity of the behavior of Nature in this area will no doubt elude our simulation capabilities, at least in image synthesis, for some time to come.

5. Chapter 5: Procedural Textures

5.1 Introduction

In this chapter will describe a very small part of our work in developing procedural textures as models of natural phenomena. As these functions are developed and evaluated in an entirely subjective manner, there is relatively little of scientific or technical depth here. These functions are, however, critical to the visual success of our images. As one of the foremost practitioners in this area, the author has been called upon to share his methodology [99,100 ,105] that others may profit from whatever insights may be obtained from such an exposition. The main point of interest, in the context of this document, is the overall strategy of encapsulating complex visual behavior reflecting that found in Nature, in comparatively terse functions or algorithms -- the process of *proceduralism.*

5.1.1. Proceduralism as Paradigm

Proceduralism is a powerful paradigm for image synthesis. In a procedural approach, rather than explicitly specifying and storing all the complex details of a scene or sequence, we abstract them into a function or an algorithm (i.e., a *procedure*) and evaluate that procedure when and where needed, i.e., via lazy evaluation. We gain a savings in storage space, as the details are no longer explicitly specified but rather are implicit in the procedure, and we shift the time requirements for specification of details

from the programmer to the computer. We also gain the power of parametric control with its conceptual abstraction (e.g., a number which makes mountains "rougher" or "smoother") and the serendipity inherent in an at-least-semiautonomous process: we are often pleasantly surprised by unexpected behaviors, particularly in stochastic procedures.

Some aspects of image synthesis are perforce procedural, i.e., they can't practically be evaluated in advance: view-dependent specular shading and atmospheric effects, for example. It is implausible, for example, to evaluate and store a realistic model of atmospheric scattering in advance of rendering; rather, the atmospheric effects are more readily evaluated along the interval between points of interest during the rendering process.

In an essential sense, *anything* done with a computer can be thought of as being "procedural", but we in computer graphics have a somewhat more specific idea of what constitutes "proceduralism", though the term may defy exact definition. It may (or may not) be safe to say that when we computer graphicists think "procedural" we are usually thinking of "modelling that is done at rendering-time", as opposed to being done in a previous, separate modelling step.

The process of developing a procedural model embodies the basic loop of scientific discovery [107,110]: A formal model is posited, observations and comparisons of the model and Nature are made, the model is refined accordingly, and more observations are made. The process of observation and refinement proceeds in an iterative loop. Hanrahan has observed [45] that a fundamental difference between the way this loop proceeds in traditional sciences versus the development of a procedural model for computer graphics, is the time required for a single iteration: for a traditional laboratory scientist this period may be on the order of years or even a lifetime; when developing a model for computer graphics it is typically more like minutes.

5.1.2. Proceduralism and Fractal Models

We noted in chapter 1 that fractals are inherently procedural, as they are specified by recursive or iterative algorithms, and that computer graphics have always been required to grasp the complexity of their behavior. Thus it is not surprising that much of what is visually complex in computer graphics, has fractal underpinnings. There is simply no known way of specifying complexity, more simply.

Our work has largely involved developing fractal models of natural phenomena for computer graphics. Being very taken with this concept of simple specification and encoding of complexity, proceduralism has been our chosen *modus operandi*. There are powerful capabilities to be gained through this approach.

Fractals contain potentially unlimited high and low spatial frequency content. The former can play merry hell with the point sampling schemes of computer graphics, due to the effects described by the Nyquist sampling theorem. The high frequency content can, and should, be parameterized in procedural fractal models. We should be able to vary the high frequency content with the screen resolution of the rendering, e.g., higher-resolution images need more small details. Optimally, we would also like to be able to vary the frequency content adaptively in rendering, as the perspective projection makes feature sizes on the screen vary as the inverse square of their distance from the eye.

Procedural approaches can readily accommodate these needs, simply by parameterizing the number of iterations in the fractal construction loop, and assigning that parameter appropriate values at the various locations in the scene. As accomplishing this represents work currently in progress, we will not describe it here, but refer the interested reader to the literature. [117]

5.1.3. Procedural Solid Textures

Perhaps the best-known form of proceduralism in computer graphics is in *procedural textures*, as introduced in 1985 by Gardner [39], Peachey [118], and Perlin. [119] A procedural, or "solid" texture, rather than existing *a priori* as a two-dimensional image which must be mapped or "wallpapered" onto surfaces, is implemented as a function defined throughout three-space and evaluated when and where needed, e.g., on visible surfaces of objects. Such functions can also be evaluated throughout a volume, as Perlin has suggested in his "Hypertextures" paper [120], but this remains an exotic application -- in fact, these wonderful procedural textures are so computationally expensive that many practitioners may hesitate to use them even on surfaces, in everyday production image synthesis.

Currently it takes on the order of hours for several high performance workstations, working in parallel, to create a single high resolution image using the (sometimes particularly elaborate) procedural textures described below. Inevitably (and hopefully within five to ten years) we will have improved the efficiency of these procedures, and have sufficient computational power at our disposal, to evaluate such models in real-time. Then we will have the capability of interactively exploring the worlds we create, in a virtual reality setting.

5.1.4. Proceduralism and Parallel Computing

How do we currently overcome the high cost of evaluating these procedures? In a word, *parallelism*. Perlin uses an AT&T Pixel Machine to evaluate his hypertextures [120]; Sims has harnessed a Connection Machine 2 to evaluate his genetically-derived texture expressions [137,138]. These are both *SIMD* (single-instruction, multiple-data) implementations; the underlying architectures allowing access to massive, if inflexible, parallel processing power. Our work has been made possible by C-Linda [16], a minimal

extension of the C programming language (by exactly six statements) which allows transparent access to multiple processors in a *MIMD* (multiple-instruction, multiple-data) parallel environment. Our Linda strategies for parallel computation are described in Appendix 1.

5.1.5. Overview

Below is our exposition on procedural textures. It consists largely of segments from notes for courses we have participated in, on the topic. [100]Musgrave, ,#122; Musgrave, ,#127; Musgrave, ,#218 It is a relatively small segment thereof, because this area is more at practice than research, and we feel that the basic flavor of such work can be conveyed by a small number of examples. It may be seen as a continuation of discussion of procedural methods begun in chapter 2, the main difference being that now we are interpreting the functions as surface textures, rather than as height fields. The examples we give come from our current efforts at planetary modelling, culminating in a model of an Earth-like planet which incorporates many of the elements described in this dissertation, upon the face of which we seek to situate our more local renderings. This model marks an important step towards the creation of a viable "virtual world", to be explored with future interactive technologies.

5.2 Planetary-Scale Clouds

We have developed a reasonable procedural model of the coloring of the surface of the Earth (presented in section 5.5) and an atmosphere in which to ensconce the planet (section 4.1.6). But the most salient visual features of the Earth, as seen from space, are the bright white clouds, and the wonderful shapes they form in the global circulation of the atmosphere. We need, then, to develop a model of global cloud cover. While the model we present is inherently two-dimensional (though it could conceivably be implemented in three dimensions as a hypertexture [120]) and has visually obvious

shortcomings,* it is very simple and gives a reasonable first approximation to nature. We use it as a transparency map applied to a sphere concentric with, and of radius slightly greater than, the earth.

We start with a special variety of fBm. We have found that an fBm constructed using the noise shown in Plate 2.7b as a basis function has a stringy, wispy character, somewhat more cloud-like than fBm constructed using the conventional Perlin noise function -- compare Plate 2.7c and d. We refer to this noise `VLNoise()`, for "variable lacunarity noise". It is constructed by composition of noise functions: a vector-valued noise function, `DNoise()`, is used to displace the texture-space coordinates of the argument vector to a scalar-valued noise function. Again, this has the net effect of expanding the band limits off the noise function, while also changing its visual character significantly.

Note how simple pseudo-code for this is:

VLNoise($\vec{p}$)
    return (Noise($\vec{p}$ + DNoise($\vec{p}$ + 0.5))

Note that the argument vector $\vec{p}$, when passed to `DNoise()`, is displaced by a factor of 0.5 in all axes, to intentionally misregister the underlying integer lattices of the two noise functions.

C code implementing this function looks like:

```
double VLNoise( point, scale )
      Vector      point;
      double      scale;
{
      Vector      temp, DNoise();
      double      Noise();

      temp = DNoise( point );

      temp.x += scale * point.x;
      temp.y += scale * point.y;
```

* The major shortcoming of this model, as well of every other procedural model involving turbulent fluid flow, is the lack of *vortices*. Short of full dynamic solution of the Navier-Stokes equation [164], there is currently no good, continuous model of fractal vortices in the computer graphics literature.

```
        temp.z += scale * point.z;

        return ( Noise(temp) );
} /* VLNoise() */
```

Here we employ the variable `scale` to modulate the magnitude of the texture-space distortion.

For simulating the swirling and streaming of the large-scale flow of global weather systems, we can apply the same concept used in the construction of `VLNoise()`, that of distorting texture space by composition of noise functions. In this case, we add vector-valued noise to the evaluation-point vector passed to the fBm function `VLfBm()`, which in turn creates fBm from the `VLNoise()` basis function. The magnitude of the texture-space distortion is greater than that used above; the exact value used being determined simply by qualitative evaluation of the results.

Pseudo-code for the function might look like:

```
Planet_Clouds( p⃗, distortion, rescale, cutoff )
        q⃗ := distortion * DNoise( p⃗ )     /* get scaled distortion vector */
        result := VLfBm( rescale * p⃗ )    /* get fBm cloud texture, in rescaled texture space */
        if result < cutoff                 /* clamp the result, to provide cloud-free areas */
                return( 0 )
        else                               /* return a normalized value */
                return ( (result - cutoff) / ( 1 - cutoff ) )
```

The weather-system texture is generated by the following C function:

```
double Planet_Clouds( point, distortion, rescale, H, lacunarity,
                      octaves, offset, cutoff )
        Vector      point;
        double      distortion, rescale, H, lacunarity, octaves,
                    offset, cutoff;
{
        Vector      p, s, DNoise();
        double      result, fBm(), VLfBm();

                /* get "distortion" vector */
        p = DNoise( point );
                /* scale distortion */
        SCALAR_MULT( distortion, p );
                /* insert distortion */
        s = point;
        SCALAR_MULT( rescale, s );
        result = VLfBm( s, H, lacunarity, octaves );
                /* adjust zero crossing (where the clouds disappear) */
```

```
        result += offset;
        if ( result < 0.0 )
              return ( 0.0 );
        else  /* normalize density */
              return( result / 1.0+ offset );

} /* Planet_Clouds() */
```

This function, called with the argument vector (3.0, 0.7, 2.0, 9.0, 3.0, 0.0) created the clouds over the planet seen in Plate 4.3. The computed value in `result` is used to represent the density of the clouds. The cloud texture modulates the transparency of the surface to which it is applied, in a manner similar to the fog in Plate 5.1, and it should have the ability to cast shadows, as seen in Plate 5.2. This requires evaluation of the texture for shadow rays intersecting the cloud-sphere and attenuation of illumination by the computed cloud density.

This stretched cloud texture is also useful for more prosaic landscape renderings [91,95], as seen in Plates 1.2 and 1.3.

The texture-space stretching described above varies smoothly, with the value of `DNoise()`. It is also possible to apply an fBm-valued distortion; indeed this might seem to be a logical way to emulate the fractal character of turbulence. Plate 5.3 shows this in practice. Unfortunately, the result looks more like raw cotton than turbulent fluid flow, again due to the lack of vortices.

5.3 A Cyclonic Storm

One of the most salient features of the Earth's clouds at the global scale, is the spiral eddy structures of the cyclonic and anticyclonic flows. Models of cyclonic storms are under development, as illustrated in Plate 5.4. A number of features similar to this (but less exaggerated), scattered with a Poisson disk distribution through the cloud texture described above is foreseen to be a plausible approach to procedural modelling of atmospheric eddies.

Below is a function designed not to create fractal eddies, but rather one large cyclonic storm with fractal cloud distributions that vary with scale. It is quite preliminary and *ad hoc*, but marks an interesting experiment in the modelling of natural phenomena with procedural texture.

```
void Cyclone( texture, intersect, colour,
              max_radius, twist, scale, offset, omega, octaves )
            /* texture & intersection coords */
      Vector      texture, intersect;
            /* surface color */
      Colour      *colour;
            /* arguments specified in vector given in text below */
      double      max_radius, twist, scale, offset, omega, octaves;
{
      double      radius, dist, angle, sine, cosine, eye_weight, value;
      Vector      point;

            /* rotate hit point to "cyclone space" */
      radius = sqrt(intersect.x*intersect.x+intersect.y*intersect.y);

      if ( radius < max_radius ) {  /* inside of cyclone */
                  /* invert distance from center */
            dist = max_radius - radius;
            dist *= dist * dist;
            angle = PI + twist*TWOPI*(max_radius-dist)/max_radius;
            sine = sin( angle );
            cosine = cos( angle );
            point.x = texture.x*cosine - texture.y*sine;
            point.y = texture.x*sine + texture.y*cosine;
            point.z = texture.z;
                  /* subtract out "eye" of storm */
            if (radius < 0.1*max_radius) {       /* if in "eye" */
                              /* normalize */
                  eye_weight = (.1*max_radius-radius)*10.;
                  eye_weight = 1.- eye_weight; /* invert */
                  eye_weight *= eye_weight; /* make nonlinear */
                  eye_weight *= eye_weight; /* make nonlinear */
            }
            else  eye_weight = 1.;  /* not in "eye" */
      }
      else  point = texture;  /* not in storm radius */

      if ( eye_weight ) {     /* if in "storm" area */
            value = eye_weight *
                  (offset + scale*VLfBm(point,omega,2.,octaves));
            if ( value < 0.)  value = -value;
      }
      else  value = 0.;

            /* thin the (default == 1) density of the clouds */
      colour->red *= value;
      colour->green *= value;
      colour->blue *= value;
} /* Cyclone() */
```

This function, called with the argument vector (1.0, 0.5, 0.7, 0.5, 0.675, 4.0, 0.7), created the cyclone shown in Plate 5.4.

Note that in Plate 5.4, the large-scale features are distorted by stretching, while the small-scale cloud features are not. As the dynamics of the processes governing cloud formation and dissipation vary with scale, a single fractal description is not sufficient. (Our model may be seen as a crude first approximation to viscous damping in turbulent flow.) The particular formulation used in Plate 5.4 is indicated by observations of nature (see Kelley [62] for striking photographs of the Earth's weather systems, seen from space). Incorporating variation of features with scale in our global cloud models may increase their realism; this is currently under investigation. Clearly much background work could be done to link the statistical distribution of our clouds, at a variety of scales, to that of clouds in nature.

5.4 Venus

Venus is a planet which can be modelled remarkably well with a procedural texture. Venus has a particularly simple, fractal appearance: its primary markings are the huge streaks in the clouds, as distorted by the Coriolis effect. The Coriolis effect amounts to a twist of the clouds as the square of the radius from the axis of rotation; see Plate 5.5. The following function creates the entire effect when applied to a pale yellow sphere.

```
Coriolis( texture, intersect, colour, scale, twist, offset, omega,
          octaves )
      Vector      texture, intersect;
      Colour      *colour;
      double      scale, twist, offset, omega, octaves;
{
      double      radius_sq, angle, sine, cosine, value;
      Vector      point;

      radius_sq = intersect.x*intersect.x + intersect.y*intersect.y;
      angle = twist*TWOPI*radius_sq;
      sine = sin( angle );
      cosine = cos( angle );
      point.x = texture.x*cosine - texture.y*sine;
      point.y = texture.x*sine + texture.y*cosine;
      point.z = texture.z;

      value = offset + scale*VLfBm( point, omega, 2., octaves );
      if ( value < 0.)  value = -value;
      colour->red *= value;
      colour->green *= value;
      colour->blue *= value;
} /* Coriolis() */
```

5.5 Jupiter and Saturn

Gas giants such as Jupiter and Saturn are particularly easy to model, modulo the eddy structure in the clouds. For this we use a function much like Perlin's "marble" texture [119]. This function perturbs appropriately-colored horizontal strata, which represent the cloud bands on the planet. Enclosing a sphere so textured in a fairly dense, hazy atmosphere can give a good approximation to the appearance of these planets. The image in Plate 5.6 is the result of a few hours of work, emulating the Voyager image in Plate 5.7. Note that we can readily get a close approximation, but that the eddy structure that characterizes turbulent flow on all scales, is conspicuously missing.

We can emulate Saturn-like rings using a disk concentric with the planet, with an appropriate texture applied. (See Plate 5.8) Here we have used a one-dimensional fBm as a transparency map on the disk, indexed by radius and suitably rolled off at the appropriate inside and outside radius values. Again, the model is entirely stochastic and subjective. Any resemblance to Saturn's rings, or the gaps therein, is entirely fictitious.

5.6 Terran Procedural Texture

The planet we geocentric humans are certain to be most interested in modelling is the Earth. In this section we develop such a model. Note that in our planetary-scale renderings the solid earth is represented, for reasons of rendering efficiency, with a smooth sphere to which an elaborate procedural texture is applied. There are two drawbacks to this approach, which might be ameliorated by using displacement maps in a non-raytracing renderer: the "mountains" cannot cast shadows, and they do not rise up through the atmosphere. The latter is a significant effect in the appearance of the Earth from space - high mountain peaks rise above a significant portion of the atmospheric haze and scattering. See Plates 36, 77 and 113 in Kelley [62] for a view from space of the island of Hawaii, and see how Mauna Kea rises from hazy sea level right through much

of the atmosphere. (Cartographers are well aware of this effect, and its usefulness as an altitude cue in topographic maps. [50]) Modelling this effect is an area for future work; for a first approximation, the smooth sphere is acceptable.

We now describe the step-by step development of a "terran" procedural texture, largely as an illustrative example of how such complex texture functions come into being. In the process, we hope to illuminate the source of the "parameter proliferation" that tends to plague complex procedural textures, as well as comprehensive and more-scientific models of natural phenomena.* Plate 5.9 documents various steps in the development. C code for the completed texture is provided at the end of the chapter..

5.6.1. Continents and Oceans

The first step in creating an earth is to create continents and oceans. This can be accomplished by quantizing a fractal (fBm) bump map as follows:

```
bump = VfBm(point);
if ( dot(bump, surface.normal) < threshold )
      surface.color = ocean_color;
else  surface.normal += bump;
```

where `point` is the ray/earth intersection point, `VfBm()` is a procedural vector-valued fBm, and `threshold` controls the "sea level". (Note that in our code fragments henceforth we assume that operators such as `+=` are valid for vectors as well as well as scalars.) This quantization, with very simple blue/grey coloring, gives us the effect seen in Plate 5.9a.

---

* It is precisely the problem of managing this huge n-space of parameters, that Sims' genetic algorithms [137 ,138] so elegantly address.

5.6.2. Climatic Zones by Latitude

Next we provide a color lookup table to simulate climatic zones by latitude; see Plate 5.9b. Our goal is to have white polar caps, barren grey sub-Arctic zones blending into green, temperate zone forests, which in turn blend into buff-colored desert sands representing equatorial deserts. (Note that this is not necessarily the most accurate color scheme for emulating the Earth, where the major deserts generally bracket green tropical equatorial zones, and are more ruddy than buff-colored.) The coloring is accomplished with a 256-entry color lookup table, which is indexed by the latitude of the ray/earth intersection point.

5.6.3. Fractally Perturbing the Climatic Zones

This rough coloring-by-latitude is then fractally perturbed, as in Plate 5.9c. We accomplish this perturbation by adding a random component to the latitude value when determining the color, and taking into account the bump map, so that the "altitude" of the terrain may affect the climate. This can be accomplished with a code fragment similar to this:

```
index = point.z + c1*fBm(point) + c2*DOT(bump, surface.normal);
surface.color = colormap[index]
```

where `fBm()` is a scalar-valued procedural fBm routine and `c1` and `c2` are scaling parameters for adjusting the influence of latitude and terrain altitude. The dot product term represents the magnitude of the bump map in the direction normal to the surface; this quantity should be computed and stored prior to applying the bump map. Note that altitude and latitude represent two independent quantities that could be used as parameters to a two-dimensional color map; to date we have used only a one-dimensional color lookup table for simplicity.

Next we add an exponentiation parameter to the value of `index` computed above, to allow us to "drive back the glaciers" and expand the deserts to a favorable balance, as in Plate 5.9d.

5.6.4. Adding Depth to the Oceans

We now modify the oceans, adjusting the sea level for a pleasing coastline and making the color a function of "depth" to highlight shallow waters along the coastlines (Plate 5.9e). Depth of the water is calculated in exactly the same way as the "altitude" of the mountains, i.e., as the magnitude of the bump vector in the direction of the surface normal. This depth value is used to darken the blue of the water almost to black in the deep areas; the blue provided by atmospheric scattering will bring the color up to a realistic value. Note that, while we have not yet implemented it, it would also be desirable to modify the surface properties of the earth-sphere object in the ocean areas, specifically the specular highlight, as this significantly affects the appearance of the Earth from space (again, see Kelley [62]).

5.6.5. Increasing Realism by Fractal Color Perturbation

Finally, we note that the "desert" areas about the equator in Plate 5.9e are quite flat and unrealistic in appearance. The Earth, by contrast, features all manner of random fractal mottling of color. By interpreting a vector-valued fBm as an rgb tuple [108], scaling it appropriately and adding the result to the color given by the index to the color lookup table, we can add significantly to the realism of our model - compare Plate 5.9e and f.

5.6.6. Result

The resulting texture provides the surface for an Earth-like planet, the realism of which is designed to be enhanced by the clouds described in section 5.2 and the

atmosphere model described in section 4.1.6. The ensemble is seen in Plates 4.3 and 4.4, with another similarly-complex procedural model of a moon. Note that in those plates the fractal function used to create the continents is the heterogeneous fBm described in section 2.3.2.5, thus the coastlines have a heterogeneous fractal dimension and the land masses have an interesting variety of detail. This is our preliminary model of a synthetic planet devised to contain enough complexity and variety to merit extended investigation, as a complete world unto itself.

5.7 Conclusions

The complexity of natural scenes is, in general, far greater than what we are currently capable of modelling, geometrically. As long as this is an obstacle, surface textures will be useful for providing visual complexity beyond that in the actual shapes of the objects in synthetic scenes.

The textures we have demonstrated here have proven usefulness in this application, as seen in the color plates from throughout this dissertation. They embody rich visual expression, in relatively compact, formal, and deterministic procedural specifications; therein lies their power.

The flexibility of the procedural approach to modelling fractal natural phenomena will be demonstrated in the animation "Spirit of Gaea" which is in production at the time of this writing. This animation, which is being made with the help of Robert Cook, Matthew Pharr, and Gordon Palumbo (senior undergraduate students in the Yale Department of Computer Science), consists of a logarithmic zoom from space down to the terrain of a fractal planet. The range of scales will be extreme, starting with the entire planet appearing pixel-sized on the screen, and moving in to a close-up of details of the procedurally-rendered fractal terrain. To demonstrate the scaling properties of the procedural models, the zoom will be continuous, employing the same models (planet

texture, atmosphere, cloud texture, and procedural height field) at all scales. While many such "Powers of Ten" zooms have been produced before, none has been made without changes of models at different scales. Thus the "Spirit of Gaea" animation will be an unprecedented technical achievement.

5.8 Sample Texture Code with Auxiliary Functions

```
/* Below is an example of an elaborate procedural texture which,
 * when applied to the surface of a sphere using the right 20-something
 * magic parameter values and a just-so color map, can do a nice imitation
 * of an Earth-like planet, literally creating (part of) a "virtual world".
 * Note that this function is surface-coloring and bump-map effect,
 * thus the sphere remains geometrically smooth.
 *
 * There are 22 arguments to this procedure, stored in the textArg[] array.
 * The following vector of parameter values (plus a color map) created the
 * planet seen in the Color Plate 1 of these course notes:
 *
 * .25 2 -.48 10 0 .45 300 2.6 .7 20 .75 220 170 20 .6 4 1.125 1.2 -.085 1 .3 0
 *
 * The code is in the form of one of the (many) cases in an enormous "switch"
 * statement in our ray tracer (John Amanatide's and Andrew Woo's
 * "Optik" from the University of Toronto's Dynamic Graphics Project).
 */

case PLANET:
/*
 * Perform initialization of fractal (spectral exponent) parameter:
 *    textArg[1] is lacunarity, or the gap between frequencies
 *    textArg[2] is the fractal codimension parameter
 *    textArg[4] serves as static storage for the computed exponent
 *            (note that this is an idiosyncrasy of our implementation)
 *    firstPlanetCall is a static flag variable, initialized to TRUE
 */
      if(firstPlanetCall) {
            textArg[4] = pow(textArg[1],(-0.5-textArg[2]));
            firstPlanetCall = FALSE;
      }

/*
 * Choose between fractal bump functions, based on the value in textArg[19]
 *    (note that the high parameter number -- 19 -- indicates that this was
 *     added quite late in the development of the texture).
 */
      if ( !textArg[19] )     /* use a "standard" fBm bump function */
            bump = VfBm(texture, textArg[4], textArg[1], textArg[3]);
      else {                  /* use a "multifractal" fBm bump function */
                  /* get "distortion" vector, as used with clouds */
            distort = DNoise( texture );
                  /* scale distortion vector */
            SMULT( textArg[20], distort );
                  /* insert distortion vector */
            texture = VADD( distort, texture );
                  /* compute bump vector using displaced point */
            bump = MVfBm(texture, textArg[4], textArg[1], textArg[3]);
      }

            /* get the "height" of the bump, displacing by textArg[18] */
      chaos = -DOT(bump, hit->normal) + textArg[18];
            /* set bump for land masses (i.e., areas above "sea level") */
      if( chaos > 0.)  {
            chaos *= textArg[5];
            hit->normal.x += textArg[0] * bump.x;
            hit->normal.y += textArg[0] * bump.y;
            hit->normal.z += textArg[0] * bump.z;
```

```
	Normalize(&hit->normal)
}

	/* if there's a colormap associated with the texture, use it */
if( cmap ) {
		/* use a scaled "z" value for offset in color map */
	temp = ABS(hit->intersect.z)*textArg[16];
		/* fractally perturb color map offset using "chaos" */
		/* textArg[7] scales perturbation-by-z */
		/* textArg[17] scales overall perturbation */
	temp = chaos*(textArg[7]*(1.-temp) + textArg[17]) + temp;
	if ( temp > 0.)   /* if above "sea level" */
			/* "mountains" appear too "chunky", */
			/* so exponentiate the color map offset */
		offset = (int)(textArg[6]*pow(temp,textArg[15]));
	else  offset = 0; /* (don't mess with oceans) */

		/* now do oceans; textArg[11] sets polar ice caps */
	if ((offset < 0.)  || ((chaos <= 0.)  && (offset < textArg[11])))
		offset = 0;

		/* clamp color map offset to upper bound */
	if ( offset > 255 ) offset = 255;
		/* set surface color to calculated color map entry */
	color = cmap[offset];

		/* darken the "deep waters" */
		/* (note that "chaos" is less than 0 here) */
	if ( offset == 0 ) {
			/* scale the effect */
		chaos *= textArg[9];
			/* make the effect nonlinear, according to */
			/* the whim encoded in textArg[21] */
		if ( textArg[21] )
			chaos *= 1.-texture.z*texture.z;

			/* limit how dark deepest waters get */
		if ( chaos < -textArg[10] ) chaos = -textArg[10];
			/* now darken the color of the deeper waters */
		color.red += chaos * color.red;
		color.green += chaos * color.green;
		color.blue += chaos * color.blue;
	}
		/* else we are in the landmass areas, where the color */
		/* is ...boring, so we'll mottle it with a vector-valued */
		/* fBm, interpreted as an RGB value */
	else if ( offset < textArg[12] ) {  /* don't mottle snow! */
			/* scale size of color-bumps */
		SMULT(textArg[13], texture);
			/* get the vector-valued fBm */
			/* (note that we've hard-coded some constants */
			/* in a feeble effort to fight parameter */
			/* proliferation.)  */
		bump = VfBm( texture, textArg[14], 2., 8.);
			/* using only bump.x is a "feature", */
			/* not a bug (don't ask me why!); */
			/* more hard-coded constants used, */
			/* to lessen parameter proliferation */
		color.red += color.red * 0.5*textArg[8]*bump.x;
		color.green += color.green * 0.175*textArg[8]*bump.x;
		color.blue += color.green * 0.5*textArg[8]*bump.x;

			/* now clamp errant color values */
		if ( color.red < 0.)  color.red = 0.;
```

```
                if ( color.green < 0.)  color.green = 0.;
                if ( color.blue < 0.)  color.blue = 0.;
                if ( color.red > 1.)  color.red = 1.;
                if ( color.green > 1.)  color.green = 1.;
                if ( color.blue > 1.)  color.blue = 1.;
            }
        } else {          /* no color map, so just use mottled texture */
            color.red *= chaos;
            color.green *= chaos;
            color.blue *= chaos;
        }
        break;      /* whew! */


/*
 * And now for some of the auxiliary functions and data referred to above...
 */

/* fBm constructed with VLNoise() */
double VLfBm( point, omega, lambda, octaves )
        Vector      point;
        double      omega, lambda, octaves;
{
        register double   l, a, o;
        register int      i;
        register Vector   tp;
        double VLNoise();

        l = lambda; o = omega;
        a = VLNoise( point, 1.0 );
        for( i=1; i<octaves; i++ ) {
            tp.x = l*point.x;
            tp.y = l*point.y;
            tp.z = l*point.z;
            a += o * VLNoise( tp, 1.0 );
            l *= lambda;
            o *= omega;
            if (o < VERY_SMALL) break;
        }
        return( a );
} /* VLfBm() */

/* vector-valued fBm */
Vector VfBm( point, omega, lambda, octaves )
        Vector      point;
        double      omega, lambda, octaves;
{
        register double   l, o;
        register int      i;
        register Vector   tp, n, a;

        l = lambda; o = omega;
        a = DNoise( point );
        for( i=1; i<octaves; i++ ) {
            tp.x = l*point.x;
            tp.y = l*point.y;
            tp.z = l*point.z;
            n = DNoise( tp );
            a.x += o * n.x;
            a.y += o * n.y;
            a.z += o * n.z;
            l *= lambda;
            o *= omega;
            if (o < VERY_SMALL) break;
```

```
	}
	return( a );
} /* VfBm() */

/* vector-valued multifractal fBm routine */
Vector MVfBm( point, omega, lambda, octaves )
	Vector	point;
	double	omega, lambda, octaves;
{
	register double	tmp, lacunarity, o, weight;
	register int	i;
	register Vector	tp, tv, result;
	double		Noise(), VLNoise();
	Vector		DNoise();

	lacunarity = lambda; o = omega; tp = point;
	result.x = 0.0; result.y = 0.0; result.z = 0.0;
		/* get intitial value */
	weight = VLNoise( tp, 1.5 );
	if ( weight < 0.)  weight = -weight;
	tv = DNoise(tp);
	result = SMULT( weight, tv );
	for( i=1; i<octaves; i++ ) {
		tp = SMULT( lacunarity, tp );
			/* get subsequent values, weighted by previous value */
		weight *= o * ( N_OFFSET + Noise(tp) ) ;
		if ( weight < 0.)  weight = -weight;
		if ( weight > 1.0 ) weight = 1.0;
		if ( (weight<VERY_SMALL) && (weight>-VERY_SMALL) ) break;
		tv = DNoise(tp);
		tmp = MIN( weight, omega );
		tv = SMULT( tmp, tv );
		result = VADD( tv, result );
		o *= omega;
	} /* for */

	return( result );
} /* MVfBm() */

/*
 * And now for the maniacs (we know you're out there) who'd type this in,
 * here's the color map used to create the planet seen in Color Plate 1:
 */
char planet_map[256][3] =
  {{1,14,81}, {176,134,80}, {170,123,72}, {164,113,64}, {158,103,56},
  {153,93,48}, {153,92,46}, {153,90,44}, {154,88,42}, {154,86,40},
  {154,84,38}, {154,83,36}, {155,81,34}, {155,79,32}, {155,77,30},
  {156,76,28}, {155,74,26}, {154,72,24}, {153,70,22}, {153,68,21},
  {152,66,19}, {151,64,17}, {150,62,15}, {150,60,14}, {149,58,12},
  {148,56,10}, {147,54,8}, {147,52,7}, {146,50,5}, {145,48,3},
  {144,46,1}, {144,45,0}, {142,45,1}, {141,46,2}, {139,47,3},
  {138,47,4}, {137,48,5}, {135,49,6}, {134,49,7}, {133,50,8},
  {131,51,9}, {130,51,10}, {128,52,11}, {127,53,12}, {126,53,13},
  {124,54,14}, {123,55,15}, {122,56,16}, {121,57,17}, {120,57,17},
  {119,58,18}, {118,59,19}, {117,59,20}, {116,60,20}, {114,61,21},
  {113,61,22}, {112,62,22}, {111,63,23}, {110,63,24}, {109,64,25},
  {108,65,25}, {107,65,26}, {106,66,27}, {105,67,28}, {103,68,28},
  {101,68,26}, {99,69,24}, {97,69,23}, {95,70,21}, {93,70,19},
  {92,71,18}, {90,72,16}, {88,72,14}, {86,73,13}, {84,73,11},
  {83,74,9}, {81,75,8}, {79,75,6}, {77,76,4}, {76,77,3},
  {74,76,3}, {73,75,4}, {71,74,5}, {70,74,5}, {68,73,6},
  {67,72,7}, {65,71,7}, {64,71,8}, {62,70,9}, {61,69,9},
  {59,68,10}, {58,68,11}, {56,67,11}, {55,66,12}, {53,65,13},
  {52,65,14}, {51,64,14}, {49,63,15}, {48,62,16}, {46,62,16},
```

```
{45,61,17}, {43,60,18}, {42,59,18}, {40,59,19}, {39,58,20},
{37,57,20}, {36,56,21}, {34,56,22}, {33,55,22}, {31,54,23},
{30,53,24}, {29,53,25}, {28,52,25}, {27,51,25}, {25,50,24},
{24,50,24}, {22,49,24}, {21,48,23}, {19,47,23}, {19,47,23},
{19,47,23}, {19,47,23}, {19,47,24}, {19,47,24}, {19,47,24},
{20,47,25}, {20,47,25}, {20,47,25}, {20,47,26}, {20,47,26},
{20,47,26}, {21,47,27}, {21,47,27}, {21,47,27}, {21,47,28},
{21,47,28}, {21,47,28}, {21,47,28}, {21,47,28}, {21,48,28},
{21,48,28}, {21,48,28}, {21,48,28}, {21,49,28}, {21,49,28},
{21,49,28}, {21,49,28}, {21,50,28}, {21,50,28}, {21,50,28},
{21,50,28}, {21,51,28}, {23,52,30}, {26,53,32}, {28,54,34},
{31,55,37}, {34,56,39}, {36,57,41}, {39,58,43}, {41,59,46},
{44,60,48}, {47,61,50}, {49,62,53}, {52,63,55}, {54,64,57},
{57,65,59}, {60,66,62}, {62,67,64}, {65,68,66}, {68,69,69},
{69,69,69}, {69,69,69}, {69,70,70}, {70,70,70}, {70,70,70},
{70,70,70}, {71,71,71}, {71,70,70}, {72,70,69}, {72,69,69},
{73,69,68}, {74,68,68}, {74,68,67}, {75,67,67}, {76,67,66},
{76,66,66}, {77,66,65}, {78,65,65}, {78,65,64}, {79,64,64},
{80,64,63}, {81,64,63}, {82,63,62}, {84,62,61}, {94,74,73},
{104,87,86}, {115,100,99}, {125,113,112}, {135,125,125}, {146,138,138},
{156,151,151}, {166,164,164}, {177,177,177}, {196,196,196}, {216,216,216},
{235,235,235}, {255,255,255}, {255,255,255}, {255,255,255}, {255,255,255},
{255,255,255}, {255,255,255}, {255,255,255}, {255,255,255}, {255,255,255},
{255,255,255}, {255,255,255}, {255,255,255}, {255,255,255}, {255,255,255},
{255,255,255}, {255,255,255}, {255,255,255}, {255,255,255}, {255,255,255},
{255,255,255}, {255,255,255}, {255,255,255}, {255,255,255}, {255,255,255},
{255,255,255}, {255,255,255}, {255,255,255}, {255,255,255}, {255,255,255},
{255,255,255}, {255,255,255}, {255,255,255}, {255,255,255}, {255,255,255},
{255,255,255}, {255,255,255}, {255,255,255}, {255,255,255}, {255,255,255},
{255,255,255}, {255,255,255}, {255,255,255}, {255,255,255}, {255,255,255},
{255,255,255}, {255,255,255}, {255,255,255}, {255,255,255}, {255,255,255},
{255,255,255}};
```

Chapter 6: Conclusions

6.1. Summary of Results

Our thesis is that there are several necessary elements for creating realistic synthetic landscape images: convincing terrain models, fast methods for rendering them realistically, atmospheric perspective for a proper sense of scale, surface textures to add visual detail efficiently, and a global setting in which to situate them. We have presented original results in each of these areas.

6.1.1. Terrain Models

We have demonstrated an essentially new terrain generation method, termed *noise synthesis*. This procedural or functional method is more flexible than many of its predecessors, in that it naturally facilitates local control of terrain character. This has allowed the development of more convincing, heterogeneous fractal terrain models. These new terrain models can resemble ancient, heavily eroded mountains (Plate 4.2), terrains on the very large scale (Plates 2.8 and 2.9), and can reflect more accurately than pure fBm, the morphology of mountains (Plates 1.2, 2.5, 4,1, and 4.13).

We have generated the globally context-sensitive patterns of fluvial drainage networks, as well as fluvial deposits, talus slopes, and diffusion effects in synthetic terrain models through dynamic simulations of physical erosion processes (Plates 2.10-2.13). The simulations of fluvial erosion will serve as the basis of experimental

verification of the published laws of sediment transport in research currently underway with Prof. Bolton of the Yale Department of Geology and Geophysics.

6.1.2. Ray Tracing Height Fields

We have developed the *grid tracing* algorithm for efficient rendering of detailed terrain models. This algorithm is general to all height field data sets.

Grid tracing is not the fastest of published ray tracing schemes for height fields; quad-tree methods and the so-called *parametric* ray tracing algorithm are faster. Grid tracing remains, however, the most memory-efficient published algorithm for this purpose, and is still the method of choice for memory-bound applications.

6.1.3. Atmospheric Scattering

Atmospheric perspective is a critical element of convincing landscape renderings; landscape painters have known this for hundreds of years. Atmospheric perspective is a result of atmospheric absorption and scattering of light, primarily the wavelength-dependent Rayleigh scattering. Constructing a general solution to the problem of modelling atmospheric scattering of light is a difficult problem, and accurate models are bound to be computationally expensive, due to the complexities of photon propagation in the atmosphere.

Due to the need for atmospheric perspective, and the difficulty of obtaining it through a physically correct model, we have devised and documented some simple, efficient alternatives. These models are based on a distillation of the primary physical behaviors of Nature, yet remain elegant and computationally efficient. They include both geometric models of scattering aerosol density distributions (Plates 1.2 and 4.1) and of Rayleigh scattering (Plates 4.2-4.4, and 5.1). These models provide effective atmospheric perspective at acceptable computational costs.

6.1.4. Procedural Textures

It is generally too expensive to provide geometric detail on the order of that found in Nature, in models for computer graphics. Fortunately, we can fool the eye rather effectively by providing surface detail on models of limited geometric complexity. *Procedural* or *solid* textures provide a flexible method for generating such detail in a rendering, at acceptable computational cost (a cost which is, in general, considerably less than that of providing a comparable measure of geometric detail).

We have demonstrated the development of a wide variety of such textures, mostly in service of modelling natural phenomena. These textures range from models of sedimentary rock strata to clouds to entire planets (see most color plates).

6.1.5. The Ensemble of Models

Taken together, these results comprise a significant advancement of the state of the art in image synthesis, particularly of fractal landscapes. We may now generate models and images of significantly greater realism and fidelity to Nature, than was previously possible. This work also comprises significant progress in the development of a new medium and process for the creation of fine art. It is hoped that some of the images created in the course of this work qualify as such.

6.2. Future Directions

The work done so far lays the foundations for future endeavors in all the areas addressed. This overall effort is foreseen to culminate in a virtual universe, populated with procedural fractal worlds which may be explored at will, along the user's chosen path, at any distance, scale, or location. Due to their procedural random fractal nature, such worlds would embody as much serendipity for their "creator" as for the "casual visitor". It is our experience that, while we "control" the creation of such places and (in

principle, at least) their visual manifestations, their stochastic nature imbues them with what almost amounts to a life of their own, and thereby with a high surprise factor and a rich store of serendipity.

Such a detailed and beautiful "virtual universe" should have significant monetary potential in entertainment products. The technology will first have to be advanced substantially: hardware and software will be required to speed up rendering by several orders of magnitude, and display and interaction devices should be improved both in resolution and ergonomics. Such advances are inevitable; in the coming years they will occur, it is only a question of when. When this work comes to fruition in that context, it will be Big.

6.2.1. Terrain Models

Fractal terrains represent a class of ontogenetic models of certain forms of naturally occurring terrains. There are, however, many types of terrains which have yet to admit to convincing synthetic modelling. Indeed, the fractal paradigm is of limited usefulness in describing Nature, as not all naturally occurring forms are marked by self similarity of a significant range of scale. Yet there remains a wide array of essentially fractal terrains which have yet to be modelled. We now describe some foreseeable progress in this area.

6.2.1.1. Heterogeneous Terrain Models

One of the primary insights driving our work is that introducing heterogeneity into terrain models can broaden our repertoire of descriptive capabilities. Given our bias towards variations on fBm which preserve, to the greatest degree possible, the underlying elegance of the fBm paradigm, we have implemented some of the most straightforward of such models which have occurred to us. There remain many relatively simple variations which have yet to be implemented.

For instance, consider the model of large scale terrains in which higher frequencies are damped by the values of previous, lower frequencies in the fractal summation. Given that the underlying idea is that valley floors should be smoother than local maxima (peaks), we should actually base our damping on the local value of the derivative the terrain function, rather than the magnitude of the previous frequency signal: the latter may occur on a steep hillside, which does not in any way qualify as a "valley floor". While conceptually straightforward, we have deferred implementation due to the increased computational complexity and the inelegance that entrains. Nevertheless, this approach should be evaluated experimentally.

There is another variation on that model which will be necessary in the foreseen application of creating procedural planets which may be explored interactively, at any level of detail. In the current model, amplitude of higher frequencies is a monotonically decreasing function, relative to that of a uniform fractal dimension, equal to the highest fractal dimension in the heterogeneous function. This means that once the terrain has become smooth locally, it will never get any rougher, at any scale. In practice, we should allow the amplitude of higher frequencies to rise again after being damped by a lower frequency. Valley floors are not, after all, uniformly smooth; they may become rough again at smaller scales. This is foreseen to be another relatively straightforward ontogentic adaptation which should prove useful in realistic terrain modelling. It may also prove useful in overcoming the problem of ever-increasing slopes in close zooms into self-affine fBm terrain models. [133]

6.2.1.1.1. Modelling Specific Features

There are many features of natural terrains which are common and salient, yet which do not necessarily fit into the fractal paradigm particularly well. These include terraced features (as with mesas), ridgelines, volcanoes, batholiths, sedimentary stratigraphy, and some large scale tectonic features (such as orogenetic belts), to name a few. Other terrain

features do not fit into the constraints of height fields: cliff faces, caves, arches, sedimentary rock strata outcroppings, and the like. Some of these features, such as orogenetic belts, should admit fairly readily to straightforward ontogenetic description. Others, such as non-height field features, will require an expensive paradigm shift.

Certain of these features, such as coherent ridgelines and strata features, are common enough to call for concerted effort to model them. Others, such as non-height field features, are perhaps not worth pursuing until a conceptual breakthrough occurs which makes their implementation more plausible. It is foreseen that many of these features could be generated through a simulation of orogenesis, as discussed below.

6.2.1.1.2. Increasing Variety in Terrain

Fractal terrain models reflect some of the complexity of Nature. They express this, however, through what seems to be the simplest conceivable form of complexity: repetition of a single form over a variety of scales. Real terrains generally feature complexity which does not admit to such simple description. There may be a wide variety of seemingly unrelated surface morphologies present at various nearby locations. This could be addressed by simply compiling a variety of independent geometric models and methods for transitioning between them, but there is little elegance in such an approach. This may remain necessary for full description of natural terrains, however, until a fundamental paradigm shift (as occurred with fractal geometry) reveals hitherto unrecognized commonalties.

Our bias in favor of simplicity in models, as prescribed by Occam's Razor, will continue to lead us to focus on more elegant solutions, as long as they are seen to be available.

6.2.1.1.3 Using Novel Basis Functions

A simple, straightforward approach to increasing the descriptive power of fBm-based terrain models is to use a variety of basis functions with varying shapes. Our experiments in this area have been very limited to date. The approaches described by Mandelbrot [74], Lewis [72], and van Wijk [160], termed *fractal sum of pulses* by Mandelbrot, allow the use of any finite basis function, or set of functions as with *wavelets* [131], through convolution with a sparse distribution of impulse functions. For best results, however, the distribution of the impulse functions should be Poisson-hyperdisk, i.e., there should be a minimum radius between adjacent impulses. At this time, we know of no efficient method for procedurally generating such a distribution in n-space; all known schemes (due to the context-sensitive nature of the problem) require pre-generation and storage of a fixed distribution. This directly trades storage space against the scale of the inevitable periodicity in the resulting function.

Nevertheless, this approach deserves additional attention. Simplifications such as using a Poisson distribution can make the method more efficient, and the gain in expressive power should be considerable.

6.2.1.2. Erosion Models

Our work on erosion models is still embryonic. Once the problems of numerical stability have been solved, we can begin to extend our results in both terrain modelling and in experimental verification of the published models of fluvial geomorphology.

6.2.1.2.1. Geophysical Simulation

By including the published erosion laws in an implementation of fluvial transport, we can test the validity of these models in simulations. Such simulations are foreseen to be of interest to geologists and hydrologists. [13] This is a benefit which was not foreseen

when we originally implemented erosion models simply to generate certain morphological features in fractal terrains.

Work in this area is ongoing, and is expected to increase in pace after the completion of this dissertation.

6.2.1.2.2. Animation of Orogenesis

One of the most exciting prospects for the erosion models is animation of the process of orogenesis through time. It is fairly easy to generate a plausible function describing, procedurally, the variations in friability (as with sedimentary strata) of underlying bedrock. This, coupled with a dynamic model of tectonic deformation and the erosion models, could conceivably generate some striking animations of the process of formation over time of mountain ranges and perhaps even features such as the Grand Canyon.

6.3. Rendering Terrain Models

The grid tracing algorithm has served us well, making feasible the ray tracing of large, highly detailed terrain models. There are many improvements and advances which can and should be made, however.

6.3.1. Parametric Ray Tracing

The *parametric* method for ray tracing height fields [115] should be added to our repertoire of rendering algorithms, due to its greatly improved efficiency. As this represents a significant coding effort, it has not as yet been undertaken. It will undoubtedly prove useful when it is available. It will not, however, entirely supersede the grid tracing code currently in use, due to its overhead in memory space. For very large height fields, where the application becomes memory-bound, grid tracing remains the rendering algorithm of choice.

6.3.2. Procedural Ray Tracing

Most algorithms for ray tracing height fields assume a precomputed height field of fixed, uniform spatial resolution. This assumption may be valid for many datasets, such as USGS DEMs (digital elevation maps) and data from planetary probes. But the character of discrete sampling and reconstruction, and the perspective projection, makes necessary adaptive level of detail in the terrain model, for high levels of realism. That is, feature size in model space should vary as the square of the distance from the eyepoint, to maintain detail in the foreground and avoid aliasing in the distance.

6.3.2.1. Promise and Limitations of Procedural Terrain Rendering

Adaptive level of detail is a stickier problem than it seems at first glance. Simply undersampling a random fractal function will lead to a reconstruction that varies with the sampling rate. Thus, as one zooms in to a landscape, the morphology and illumination values may change dramatically with change of scale. There may be no way to overcome this, other than to precompute the integral of the function over the entire range of scale of its geometric detail. This assumes that the function is band-limited, which in turn limits the detail available, which in turn (the self-affine character of fBm notwithstanding) seems to violate the appeal of fractal models: the availability of potentially unlimited detail. At any rate, precomputing, storing, and accessing such integral information is likely to be impractical, if not simply inelegant.

Nevertheless, procedural rendering has great appeal: we can imbue our images with pixel-sized geometric detail at all ranges and locations, and we can use terrain models which cannot be rendered properly otherwise. Plate 4.1 is an example of this: this terrain is fBm constructed with a basis function with a discontinuous derivative, which yields coherent ridgelines in the resulting terrain. Such ridgelines would become saw-toothed,

if their geometry were undersampled, and (as with any stochastic terrain) alias in the distance if oversampled geometrically. Thus the procedural rendering has granted us the ability to render a striking terrain model which would be otherwise unrenderable, and has proved useful despite the drawbacks outlined above.

For our ultimate purposes of exploring procedural planets in an interactive setting, we will require such procedural rendering, with or without its flaws. It is hoped that artful use of atmospheric perspective can mask the most egregious shortcomings sufficiently to make the net result acceptably realistic.

6.3.2.2. Quad-Tree Method

Procedural ray tracing with adaptive level of detail calls for a quad-tree data structure [54,55], as that naturally accommodates the change in spatial resolution with distance. The grid tracing algorithm is inappropriate in its naive form, as it assumes a fixed resolution in the grid. In practice, this has been overcome by changing to different resolution grids at appropriate distances. [2] The parametric algorithm may admit to similar treatment, this is yet to be investigated.

At any rate, our implementations of procedural ray tracing currently under development utilize quad tree data structures for spatial subdivision.

6.3.2.2.1. Calculating Bounding Volumes

Efficiency in procedural schemes is largely dependent on accurate determination of bounding volumes. In the absence of such information, all terrain traversed must be explicitly generated and tested for intersection the ray. This is costly, and can be avoided if the terrain function can be determined to be out of the path of the ray by some other means. Determining tight bounding volumes is a nontrivial task [15,54], and is an area of current investigation [162].

6.3.2.2.2. Memory Management

Models with adaptive level of detail become large, in high resolution images. Furthermore, a geometric element (i.e., primitive triangle) of the surface is likely to be visited by only a few rays, in close temporal proximity in the rendering process, and never referenced again. Thus dynamic memory management is essential to the algorithm. This is another area currently under investigation.

6.3.2.2.3. Parallel Implementation

As with all our work, we use C-Linda to speed our procedural renderings. We can use the distributed computation model of network Linda not only to increase the cycles applied to the problem, but to facilitate memory management as well. When a single processor is used to ray trace a procedural terrain at high resolution in the usual horizontal-scanline-at-a-time paradigm, it may have to free parts of the model computed at one end of the scanline before it reaches the other end, due to memory constraints. Thus parts of the model may end up being evaluated repeatedly, which is highly inefficient. Any given part should be evaluated and stored once, referenced as many times as necessary, then freed. By managing the tasks assigned to workers in the Linda paradigm, we can assure that each worker is constrained to work on a subsection of the horizontal extent of the image. Hopefully this extent can be kept small enough that this reevaluation is unnecessary. At the time of this writing, experiments are about to begin in this area.

6.3.2.3. Analytic Method

It is possible to use information about the derivatives of the cubic spline of the noise function to devise a fairly efficient scheme for analytic ray tracing of noise-based height fields and hypertextures. [162] This is another area that merits further investigation. Procedural geometric models with adaptive level of detail could then be devised by band-

limiting the fractal function as a function of distance, which is quite straightforward. [117]

6.4. Atmospheric Scattering

Atmospheric scattering will remain a challenge for the foreseeable future. This is an area which has received much attention in the scientific literature, as it reduces to the more general problem of particle transport in the presence of scattering media. [6] Such problems are of importance to the design of nuclear reactors, radar, etc., and in the theory of stellar evolution and heat exchange. [18] Thus, dramatic improvements in the simulation of scattering are unlikely to come from the field of computer graphics. We have demonstrated, however, that by respecting the peculiar goals of computer graphics, we can devise novel models which are useful for image synthesis.

6.4.1. The Multiple Scattering Problem

Single scattering can be simulated fairly readily and accurately. [64] Unfortunately, in most interesting systems multiple scattering is an important factor. This is much harder to deal with. Some believe that the multiple scattering problem may be effectively addressed by Monte Carlo ray tracing [4], but it is the author's considered opinion that it will probably only be addressed satisfactorily through solving for the equilibrium state of energy transport, as with the radiosity method. [130] This approach is facilitated in turn by application of the Greengard-Rokhlin algorithm [42] as a three-dimensional extension of the method described by Hanrahan et al. [46] At any rate, there clearly remains much work to be done in this area.

6.4.1.1. Rayleigh Scattering

Experiments with several implementations of single-scattering Rayleigh models indicate that multiple scattering is essential to realistic results in emulating the Earth's

atmosphere: in single-scattering models, the sky simply isn't blue enough when the sun is low to the horizon, indicating that short wavelengths are subject to extensive multiple scattering. This in turn indicates the development of a multiple-scattering Rayleigh model for added realism in renderings of outdoor scenes.

6.4.1.2. Clouds

Clouds are obviously an essential element in landscape scenes. They might admit fairly readily to geometric modelling with fractal hypertextures. [120] But shading the geometric model is another, perhaps more difficult problem: illumination in clouds is largely a function of multiple scattering by high-albedo particles. It is plausible a striking (if not wonderfully efficient) cloud model could be had by combining a procedural geometric model with a volumetric radiosity illumination model, as mentioned above. This is, therefore, a rich area for future research.

6.4.2. Monte Carlo Single Scattering

Given that multiple scattering is a hard and largely unsolved problem, what *can* we do effectively? Single scattering can provide a useful first approximation for image synthesis purposes, and can be implemented in a fairly straightforward manner with Monte Carlo methods, similar to those used in distributed ray tracing. [23,25]

6.4.2.1. Ray Marching

As in the distributed ray tracing paradigm, a Monte Carlo variant of the midpoint rule can be used to approximate the illumination integral along an optical path by "marching' the ray along intervals, and firing secondary rays towards light sources from random points within each interval. This sample is then used to represent the illumination for the entire interval, and subsequently the scattering towards the ray origin. For non-homogeneous aerosol density distributions, such as those described in Chapter 4, the

width of these intervals might be normalized by aerosol density, rather than spatial extent; this represents yet another problem currently under investigation.

6.4.2.2. Atmospheric Shadows

The ray marching algorithm sketched out above would immediately yield atmospheric shadows, as cast by mountain peaks into the mists below, and crepuscular rays (sunbeams) cast by clouds. As is demonstrated by many photographs and paintings of landscapes [40,49], these may be highly desirable aesthetic effects. Thus some such scheme should be implemented and added to our ray tracer.

6.5. Procedural Textures

Procedural textures will remain mostly an application, perhaps pursued more in commercial software development than in academic research. The topic is still of vital interest to practitioners of computer graphics, as the recent acceptance of our second course in the area, proposed for SIGGRAPH '93, indicates. There also remain some interesting challenges in the area.

6.5.1. Clouds

As the color plates illustrate, we have had some success in developing elegant, two-dimensional models of clouds. Again, clouds are essential to landscape renderings, and the potential for extension of existing models is almost unlimited -- there are a great variety of cloud types in Nature, and we have effectively emulated but a few.

6.5.1.1. Geometric Models

It is foreseen that creating models of certain cloud types -- most notably, cumulus -- might prove relatively easy, using fractal sum of pulses or hypertexture approaches. Unfortunately, while such models might be straightforward to specify, they may prove

difficult to render efficiently. The sheer geometric complexity of most clouds challenges the development of efficient rendering algorithms. Add to this the fact that clouds do not have surfaces as such, only density distributions, and the complexities of illumination described above, and we see that we have a truly significant challenge in modelling and rendering. No immediate breakthroughs are foreseen in this area, other than the application of the Greengard-Rokhlin algorithm to radiosity illumination calculations. The sheer importance of clouds in terrain renderings insures that work in this are will continue, however.

6.5.1.2. Modelling Turbulence

A closely related topic is procedural modelling of turbulence. This has applications in modelling clouds, smoke, galaxies, and indeed almost any common instance of fluid mixing. A convincing, continuous (i.e., non-particle system) model of the transition from laminar to turbulent flow in smoke rising from a cigarette, using something significantly more computationally efficient than a full Navier-Stokes solution, would be a significant accomplishment in computer graphics.

Turbulence is composed of a hierarchy of eddies. It seems plausible that successive application of annular-twist distortions to some underlying function, over a range of scales, might produce an emulation of turbulence in two dimensions. Toroidal distortions might achieve the same effect in three dimensions. Wavelet analysis, with similar wavelet functions, might be used to predict the success of such models and to tune the basis function shapes. These are ideas which deserve to be tested and evaluated at least for subjective effectiveness in modelling turbulence.

6.5.2. Planets

As we eventually intend to populate a synthetic universe with procedural worlds to be explored interactively, we will need to develop and extend our models of planets. This

will be largely an artistic undertaking, but an increased variety of procedural models will be required to maintain a high level of interest for explorers of such places.

6.5.2.1. Heterogeneous Virtual Planets

Interest in such models will be largely a function of their heterogeneity -- "variety is the spice of life", after all. We have developed some preliminary heterogeneous models which provide greater variety in landforms. Relatively little work has been done in this area to date, and more results are there to be obtained. As greater complexity in the result is sought the underlying algorithms, the elegance of which we have worked so hard to retain, is bound to be ever more compromised. Thus this work, too, is likely to be pursued more in the realm of commercial product development than in academic research. There may yet remain some characterization and classification work to be done in that context, though.

6.5.2.2. Refining Parameter Space

As was demonstrated in Chapter 5, complex procedural models, and indeed all models of complex natural phenomena [32], tend to inflict upon the user an undesirable proliferation of picayune numerical parameters with obscure effects on the model. Each parameter increases the dimensionality of the n-space which the user must search for the desired result. Thus an essential part of any concerted effort to make such models usable, is to determine the principal components of the relevant parameter space, and then to provide some meaning to the resulting parameters, which will be in some way intuitive to the user. This is not an easy task.

6.6. Modelling Vegetation

A salient feature of most landscapes, which has been almost entirely missing in our models and renderings, is vegetation. There are good reasons for this omission: first, the

dearth of convincing geometric models of bushes and trees (arguably the most important plants in landscape images) and second, the substantial difficulties in dealing with the spatial frequency content of vegetation. If the first were available, the second might make it unusable for the purposes of production image synthesis. It is almost certainly not feasible to derive illumination integrals for pixels depicting distant, geometrically-modelled vegetation, with point samples as in ray tracing. Such an integration would fail to converge with a reasonable number off samples, in general.

The most promising approach is a hierarchical model such as that proposed by Kajiya [53], in which a geometric model is employed up close, a texture at mid range, and a surface lighting model in the distance. It is not immediately apparent how any of these models should be formulated, or how to handle transitions between them. The need for such models of vegetation assures that work in the area will, however, continue. [124]

6.7. Virtual Reality

The ultimate application of this work may be in a virtual reality entertainment setting. The scenes we have devised, and the capability of creating complex, intriguing procedural planets should prove rich in entertainment value, once the capability is in place to explore such scenes interactively with sufficient detail and realism. Such creations may serve as settings for games, exploration, or meditation. Their rich intrinsic beauty should be able to hold the attention of "visitors" for a significant period of time. (Humans are invariably most interested in the doings of other humans, however, so the most popular applications will undoubtedly include representations of people, something entirely outside the scope of this work.)

6.7.1. Displays and Input Devices

Current technology for virtual reality is crude, to assess it charitably. Displays generally feature unacceptable resolution and color, and are too bulky for comfortable

use. Input devices often feature poor spatial resolution, range, latency, and reliability. But these things merely represent engineering problems, problems which will be solved in years to come. In five to ten years, those solutions will be implemented in mass-produced, low-priced, ubiquitously available devices which every average Western family will have in their homes. Such progress is inevitable, and we computer graphics researchers need not concern ourselves with the issues; they will be solved for us, and soon.

6.7.2. Real-Time Rendering

Another problem that *is* the concern of graphics researchers, is real-time rendering. Currently, realistic rendering cannot be done at a sufficient rate by anything less than the absolute state of the art graphics engines, and even these do not approach the level of realism achieved in our work. Three factors will contribute to the solution of this problem: hardware improvements, more efficient algorithms, and code of improved efficiency. The major improvements will be made in first two categories; graphics programmers are generally already in the habit of writing efficient code. Algorithmic improvements are difficult to predict, as they tend to take the form of unforeseeable conceptual leaps. The parametric method for rendering height fields is a good example: the algorithm is non-obvious, and may have gone unthought of for an arbitrary period of time. There is simply no telling when and where the next conceptual discontinuity will occur.

What is predictable, is the continued exponential advance in the speed of computation devices. This alone might eventually effect the orders-of-magnitude speedup we currently require. At the time of this writing, the time required to render one of our image at high (e.g., HDTV) resolution is on the order of hours. We need to get this down to about 140 frames per second (two 70 Hz channels, one for each eye in a stereoscopic display). This amounts to a gap of about six orders of magnitude, base 10. It would

wasteful to simply wait for hardware advances to close such a large gap. Fortunately, those interested in simulators of various sorts are constantly working on efficient solutions to the problems of adding realistic visual details to real-time rendering systems; we may be able to borrow some of their solutions, such as Gardner's table-lookup schemes for fractal texture generation. [37,38]

6.7.3. Forecast

When this technology matures into a virtual reality product, the economic opportunity will be enormous. As a next-generation video game, the market will comprise just about every young person who can afford it. This will require the convergence of several technologies: displays, interaction devices, and rendering. We see our role in this development process as that of refining aesthetics and realism in virtual worlds, so that when the technical capability matures, we will have a virtual universe worth exploring already "in stock".

Perhaps less lucrative, but far more interesting, are the prospects of this peculiar medium and process as fine art. Our capabilities in image synthesis are currently insufficient for creation of what could stand comparison to established media such as painting and sculpture. We need to discover potentials and develop craftsmanship, to levels orders of magnitude beyond our current capabilities. It is not immediately evident what the "right" solutions are, particularly for issues of the physical manifestation of the art object itself. Ours is a highly conceptual process, the import of which lies mainly in the machinations through which the image comes into being. But marketable artworks are either objects or performances, and we do not yet have the means to create either, with quality comparable to other media in the fine arts. Thus the process is reasonably advanced, but the medium is embryonic.

The most exciting future challenge of this work is the development of a fully mature capacity for the creation of fine artworks. This opportunity is unbounded, and will occupy the author for the foreseeable future.

Appendix A. Parallel Computation

7.1. The Linda Paradigm

C-Linda [16] is a *coordination language* [17] which greatly facilitates marshaling the computational resources in asynchronous MIMD (multiple instruction, multiple data) parallel computations. C-Linda extends the C language by exactly six statements. These six statements handle, transparently, the machinations of multiple process control and communication.

The beauty of Linda, for researchers uninterested in the picayune details of parallel strategies, is its simplicity. Processes are spawned by calling the `eval()` statement. Communication is handled with the `out()`, `in()`, `rd()`, `inp()` and `rdp()` statements. These statements move data in and out of *tuple space*, a shared memory space with a relational database-style reference mechanism. Data is removed from tuple space with `in()` and referenced with `rd()`. These are blocking operations; `inp()` and `rdp()` provide equivalent non-blocking operations. The Linda user confronts parallelism at this very high level; all of the considerable complications of memory management, interprocess communication, and process control are taken care of without the need for direct orchestration by the Linda programmer. This can simplify immensely the task of writing a parallel program.

The fact that C-Linda is an extension of the C language facilitates parallelization of existing C programs. It also ensures portability of the resulting code, as there are no architecture- or platform specific constructs. C-Linda also facilitates maintenance of the capacity for sequential execution in a given program, by coding Linda portions with conditional compilation flags.

Our experience has been that we can readily achieve a near-linear speedup, per processor, in ray tracing applications using C-Linda. This can be accomplished with a minimum of effort (e.g., a day or two of design and coding time) while maintaining code portability and sequential execution capacity.

Our initial C-Linda implementation was the parallelization of the Optik [1] ray tracer of the University of Toronto's Dynamic Graphics Project. Subsequently, Rayshade [67] was parallelized. Next, Rob Bjornson parallelized our erosion code [10]. We are currently working, with Robert Cook and David Kaminsky of the Yale Department of Computer Science, on parallelization of the Raypaint interactive ray tracer shell and on Linda schemes for procedural height fields. These last two applications represent the first we have worked on where the parallelization was not a fairly trivial operation, as the former requires empirical tuning to find optimum task granularity, and the latter involves dynamic memory management.

We now describe our work with C-Linda. Please keep in mind that the most interesting aspects represent work in progress, and therefore reporting of definitive strategies and results is not possible, at the time of this writing.

7.2. Ray Tracing

Ray Tracing qualifies as an "embarrassingly parallel" application, in the MIMD (multiple-instruction, multiple-data) paradigm. The fact that each ray represents an independent thread of computation (as there is, in general, no predictable coherence

among adjacent rays) indicates MIMD parallelism over SIMD (single-instruction, multiple-data). Each processor should be free to follow the computational thread dictated by the ray's path through scene-space, and there is no requirement for communication with, or dependency on, other rays in the scene. Furthermore, ray tracing is CPU-bound: it will typically spend the bulk of its execution time in floating point operations, as opposed to, for example, memory accesses.

All of these factors indicate an ideal Linda application.

7.2.1. Screen-Space Subdivision

Much research has been devoted to parallelizing ray tracing algorithms. None of it has, to the author's knowledge, produced results worth referencing here. The obvious strategies, such as SIMD parallelization, subdivision of scene-space among processors, assigning a processor per ray, etc., fail in the general case, each from its own foibles. Due to the generality of the problem of image synthesis by the point sampling scheme of ray tracing, most schemes designed to optimize any aspect of the algorithm can be defeated by pathological scenes. In light of this observation, it is the author's view that the most sensible approach is to seek simple and elegant solutions which work well most of the time, and to not bother too much with exceptional situations.

The aspect of ray tracing that we are out to optimize here, is speedup per processor in a parallel environment. The simplest, most elegant and effective approach we have found to speeding this process is parallelization by *screen-space subdivision*. In this scheme, a parallel task granularity is defined by an area of the screen to be rendered as a distinct task. This scheme, too, may be foiled by pathological scenes: imagine, for example, a scene where almost all the imaging computation time is expended in one tiny area of the screen. In such a scenario, one processor is likely to end up doing almost all the work, and there will be little significant speedup from parallelism. Experience shows that such

scenarios are rare*, however, and thus screen-space subdivision is a viable general strategy.

7.2.1.1. Single Scanline Tasks

Given that we are going to subdivide screen space to produce our parallel tasks, we need to decide how to do so. The first idea might be one-pixel tasks. This turns out to have too small a granularity for Linda computations; communication overhead is significant at this level.

The next idea is one-scanline tasks. The scanline approach has the advantage of easy checkpointing: as each processor completes a scanline, the "supervisor" process collects that data and writes it to nonvolatile storage (i.e., a file on disk) as soon as all previous scanlines are available and have been collected and written to storage. Thus if the fifth scanline is the first to be completed, it is cached in tuple space until the preceding four scanlines are completed; then all five are written consecutively to the image file. If the rendering process is interrupted when the image is partially completed, all that is lost is those scanlines upon which work has been started, but which have not yet been written to nonvolatile storage.

Given this checkpointing scheme, there is little point in using a granularity smaller than one scanline. Furthermore, one-scanline tasks are big enough that communication overhead is negligible. Thus scanline granularity is suitable for our purposes.

---

* Such scenarios are not unheard-of, however. In a scanline-block parallel rendering of the image seen in Plate 9.1, the worker which gets the task of rendering the water just below the horizon usually holds up the computation significantly, due to the fact that the oblique view of the ripples there triggers a high rate of supersampling, with adaptive antialiasing enabled (as it always is, in our final renderings).

7.2.1.2. Watermarks

In the scanline scheme, the supervisor process may naively put all scanline task-specification tuples into tuple space at once, at the beginning of the rendering. Indeed, we did this in our first implementation. Workers then remove the scanline task tuples in sequence (e.g., from bottom scanline to top), render the scanline, and place the result back in tuple space for the supervisor to collect.

A serious problem was encountered with this scheme: when one or more workers fell well behind, for whatever reasons, the other workers could eventually overflow the capacity of tuple space. This may be unavoidable, due to the fact that final image file sizes range up to 100 Mb and more. Our experience indicates that Linda does not degrade gracefully upon tuple-space overflow: the worker who's attempted `out()` caused the overflow simply dies and becomes a zombie process, without any notification of other related processes. Thus useful computation comes to a halt, but the other processes continue to use up resources, blindly working away on a lost cause. This is not a desirable situation.

This problem can be alleviated with a *watermark* scheme. In our implementation, the supervisor only outputs task tuples up to a fixed number of scanlines above the last finished scanline retrieved from tuple space. Thus potential tuple space usage is limited to a fixed upper bound. This scheme may result in decreased efficiency due to idle workers when no task tuples are available, but again our experience indicates that this does not often happen. (We know this, because our code is written to inform us with a message to the terminal every *n* seconds that a worker is so idled, and we do not often see these messages.)

7.2.1.3. Multiple Scanline Tasks

The single-scanline task is fine for non-antialiased images. It would also be fine for images antialiased by fixed-rate supersampling per pixel. But the most efficient way to antialias, in general, is to employ *adaptive supersampling*. In adaptive schemes, adjacent samples are compared by some measure of contrast. [89] If a given threshold is exceeded, then the area (e.g., pixel) being sampled is subdivided into smaller areas, upon which the adaptive sampling routine is recursively called, until either the contrast drops below the given threshold or a given number of levels of subdivision is reached.

This adaptive antialiasing causes the ray tracing algorithm to become context-sensitive; there is now dependency among rays and communication may be required.

When we employ adaptive antialiasing, we typically limited it to a maximum of 64 or 128 rays per pixel. Our renderings typically average about four to six rays per pixel. To assess the adaptive refinement criterion, we require one sample per adjacent pixel. Thus a worker assigned an area of the screen to render, will need to sample outside the border of this area, at a rate of one sample per pixel (we refer to these as "external" samples). As the ratio of internal to external sampling rates is generally high, we may elect to simply compute the external samples for every task area, as opposed to sharing data via interprocess communication, which is an option for the overlapping borders of adjacent areas. In other words, we simply take the performance hit of repeating computations in overlapping areas, in favor of maintaining task independence.

This in turn indicates assigning task blocks of several scanlines each, to minimize overhead due to repeated computation of samples. The upper bound to the size of these scanline blocks is modulated by the need to avoid overflowing tuple space with results, and the desire to efficiently checkpoint computations done so far. That is, is the block size is too large, tuple space may overflow before all *n* processors are able to place their

results there, as the supervisor is constrained to remove scanlines in order (not being able to store a large image file in its local memory). Furthermore, even if this were possible, we wish to write as much of our results to non-volatile storage as possible, as soon as possible, to checkpoint the rendering in case of (the not-infrequent) interruption of the rendering process.

We have employed this strategy with good results. Overhead due to repeated computation is typically low, on the order of a few percent for ten-scanline tasks.

7.2.1.4. Postage Stamps

Given that we are going to take the hit for repeated computation of external samples, we wish to minimize this penalty. The way to do this is to maximize the ratio of internal to external samples. That is, we should maximize the screen area of the task, while minimizing the total length of its border. For reasons of simplicity in programming, it is really only reasonable to consider rectilinear quadrilateral screen areas. Given these constraints, square or nearly-square "postage stamp" screen areas comprise the optimal task.

Postage-stamp screen subdivision adds significant complexity to our Linda code; this is undesirable. It does, however generally speed the computation by a few percent. Its greatest advantage lies in its capacity to load-balance our landscape renderings. In a landscape rendering, high spatial frequency content tends to center around the horizon, due to models receding into the distance there. In a scanline-block task implementation, the worker or workers which receive the task or tasks around the horizon tend to fall behind, due to the heavy sampling rates called for by adaptive antialiasing there. In a postage-stamp decomposition scheme, work in this difficult area is more equitably distributed among processors.

7.2.1.5. Boundary Tuples

The optimal solution may be to precompute the external samples, and place them into tuple space for reference by all interested parties, without repeated computation. This has not been attempted to date due these concerns: A) Time dependence: if the shared data is not available in tuple space, what to do? Compute it, and place it in tuple space? Wait for it to appear? Either may entail performance hits. B) Communication overhead: it may turn out that the overhead for communication of such relatively small data sets may be comparable to the time required to recompute them. C) Complexity in programming: implementing such a scheme will further complicate our once-simple Linda code.

Work is in progress to assess the legitimacy of these concerns, and the potential efficacy of this approach. In general, we feel that there are more fruitful areas of endeavor, than implementing elaborate schemes to shave a few percent off our rendering times.

7.2.2. Shared-Memory Shared-Bus vs. Distributed Architectures

Our Linda implementations have been for both shared-memory, shared-bus architectures and distributed workstation environments. There are different assumptions which may or may not be made with each. As our original Linda work was on the Encore Multimax, an architecture of the former type, we began by making assumptions which hold only for such an architecture and which later needed to be amended -- though not necessarily *corrected* -- for the distributed computing environment.

The chief non-portable assumption which worked for the Multimax and the Apollo DN 10000, but not for distributed (i.e. "tsnet") computations, was the "fork" model for `eval()`. In a UNIX "fork", an exact copy of a process' image is created, with only a flag variable to tell the parent process from the child. This image duplication allows the ray tracer to first read in the scene specification file, then spawn pre-initialized workers by

`eval()`. A tsnet `eval()`, however, starts a new process by calling the executable file from non-volatile storage. Thus each worker must individually initialize itself by reading the input file or files.

A typical landscape rendering includes a height field of substantial size. On a shared-memory machine, the height filed data may be placed in shared memory and need not be replicated, whereas in a distributed computation, each process must read and store the height field data. For a large number of workers, this repeated reading of the input data can lead to contention over height field file access, and generates a significant overhead in startup time (typically on the order of tens of seconds).

A concern in the use of shared memory is contention among the various processors for access to the data stored in shared memory. We have not carefully analyzed the memory contention situation, but we have seen no evidence that this has been a significant impediment to the efficiency of the algorithm. The fact that the greatest proportion of the rendering time for our images is spent on independent work such as evaluation of procedural textures may help to alleviate potential memory contention problems, but subjective evaluation of simple tests of rendering without such textures still indicates no significant contention problem.

7.2.3. Procedural Height Fields

In procedural rendering of terrain models, the model is only evaluated when a ray "violates its airspace". The model is stored in a hierarchical data structure, typically a quad-tree, and evaluated with adaptive level of detail by distance from the eye. As the goal of the procedural method is to provide pixel-sized detail at all visible locations, the structure containing the model can get very large (up to gigabytes) in high-resolution images. Procedural terrains are very expensive, computationally, to render (on the order

of days on an single IBM RS/6000 640, for a 1280x1024 pixel antialiased image). They are therefore prime candidates for parallelization.

7.2.3.1. Memory Management

The procedural model must be evaluated not only at the places where it is ultimately visible, but also in non-visible areas as well, as these areas must be checked for ray/surface intersection in order to rule them out. These areas include the foreground below the edge of the scene, areas masked by features between them and the eye, and off-screen features checked for intersection with shadow rays. As detail is a function of distance, there may be a particularly substantial overhead involved in computing and storing the non-visible portions of the height field in the foreground, below the edge of the screen Also, low lighting angles cause a greater overhead in of-screen model evaluation for shadow calculations.

The procedural terrain itself is, in our applications, generated by the noise synthesis method described in chapter 2. Recall that this is not a particularly computationally-efficient modelling method; therefore each portion of the model represents a substantial investment of computing time, once evaluated. We therefore wish to avoid recomputing the model; rather we would prefer to store it, once computed, for as long as it will be referenced in ongoing computations.

Due to the size of the highly-detailed model, we cannot usually save all such data, in a uniprocessor implementation. Due to constraints in the size of volatile memory, we find that the requisite dynamic memory management scheme in the rendering code often has to recycle memory storing the model at one end of a scanline, before it reaches the

other end. Thus the entire model may be recomputed at each scanline,* an event which causes rendering time to go right through the roof, so to speak.

In a Linda implementation, we may constrain workers to preferentially render horizontal screen extents where they have already worked. This gains two advantages: First, these extents will be width *r/n* where *r* is the horizontal screen resolution and *n* is the number of workers. Thus for *n* of reasonable size, they should be small enough to avoid this model-recomputation problem, as the entire foreground model may fit into local memory for that smaller horizontal extent. Second, the storage required is more likely to be small enough to fit in real memory, thus avoiding costly page faults.

7.2.3.2. Load Balancing

Suppose one or more workers should fall behind, causing another worker to reach the high water mark in task tuples in its preferred vertical swatch of the image. This worker may then search for the lowest (by scanline number) available tuple, and begin working in this horizontal extent to help that worker which is farthest behind. This may cause model storage in the helping worker's original extent to be overwritten. But, by keeping track of first and second choices for horizontal extents, any given worker should be able to successfully minimize its horizontal extent of concern, while helping to maintain load balance.

This has not yet been implemented, thus we have no experimental evidence of the efficacy of this scheme.

---

* One might immediately suggest rendering the image using vertical scanlines, but this A) violates standard image-handling conventions, which nearly always assume horizontal scanlines in adherence to raster scan conventions; B) therefore necessarily creates a sideways image which must subsequently be rotated by 90°, which is not easy for a file that may be several tens of megabytes in size; and C) is not a general solution: consider animations, where a camera may roll 90° or more, thereby inverting the problem.

7.2.3.3. Piranha Implementation

The load balancing problem becomes more interesting still in a Piranha scenario, where the absolute number of processors available to the parallel computation is not known, and is expected to be constantly changing. (Piranha is a version of "tsnet", or network Linda, which is designed to start a process on any available node on a network if that node has been idle for some specified period, and to "retreat" when the node becomes active again, as when the owner of the workstation returns to his or her keyboard.) Here the optimum width of the horizontal extents cannot be known *a priori*, as the number of processors available is not known in advance. Furthermore, optimal extent size changes as the number of processors "on the job" changes.

Optimum load balancing in this complex scenario is a nontrivial issue. We are currently researching this area with the help of Robert Cook and David Kaminsky, of the Yale computer science department.

7.2.4. Interactive Ray Tracing

Yet another nontrivial parallel implementation is that of Raypaint, the interactive ray tracing shell which has been fitted to both the Optik and Rayshade ray tracers.

7.2.4.1. Raypaint description

Under Raypaint, the ray tracer performs adaptive subdivision of the screen, by the same contrast-comparison mechanism used for adaptive antialiasing on the pixel scale. Raypaint then uses the fast polygon filling routines of the X window system or the Silicon Graphics "gl" graphics library, to fill the entire display screen with smoothly (i.e., Goraud) shaded quadrilaterals. The image then appears to dynamically "come into focus" as larger quadrilaterals are subdivided into ever-smaller ones, gradually increasing detail seen on the monitor.

That is the back end of the shell; the front end consists of a mouse-driven interface whereby the user specifies areas of interest for preemptive refinement work.

The net result is a tool of great use, for rapid prototyping of images. It facilitates greatly decreased turnaround time in the loop of iterative aesthetic refinement of artistic images, for instance.

7.2.4.2. Granularity

Image-space refinement takes place with the aid of a quad-tree data structure. Task size could easily be as small as one screen-sampling ray per task; it is readily increased by factors of additional levels in the quad tree (i.e., by powers of four).

Optimum task size must be determined empirically, and may vary from among network environments. Preliminary results with a network of IBM RS/6000 servers, rendering over a gateway to an X window on a Sun Sparc 1, indicate a minimum task size of three levels in the quad tree. As a 512x512 image represents a quad-tree depth of eight levels, the resulting effective tree is only three tasks deep. Image rendering is greatly speeded by the parallel computation; but the coarse granularity obviates much of the interaction in refinement specification.

This parallel implementation is foreseen to be far more interesting, when run directly on a shared-memory shared-bus graphics computer such as the new Silicon Graphics multiprocessor machines. Linda latency should be less, allowing finer granularity, greater speed, and better interactivity. Also, overhead in process startup time should be substantially less, which greatly facilitates interactivity (no user wants to sit and wait for a good part of a minute for an interactive program just to start up).

7.2.4.3. Piranha Implementation

We are currently developing, again with the help of Robert Cook and David Kaminsky, a Piranha version of Raypaint. Foreseen problems include startup latency and graceful process retreats (both potential killers in an essentially real-time system such as Raypaint), and determining optimum granularity with an unknown number of processors.

7.2.5. Results

With the shared-bus and tsnet Linda ray tracer parallelizations, we have achieved essentially linear speedup by number of processors, to within a few percent of optimal linear time. [66] With the procedural terrains, we may achieve a highly superlinear speedup, due to efficiency wins outlined above (i.e., avoiding repeated model evaluation). Piranha schemes should maintain these results, with the added overhead of starting and stopping processes.

The efficacy of network-parallel Raypaint schemes is, however, highly questionable if only because of the generally long startup latency. Linda and Piranha Raypaint on a shared-bus Silicon Graphics multiprocessor should be a real win, in terms of speed. We look forward to having the opportunity to try it out in the near future.

7.3. Erosion Simulation

The other part of our work that has involved Linda parallelization is the erosion code. For this application, the Linda coding was performed by Robert Bjornson in consultation with the author. The code produced was designed to run on the Intel iPSC/2; for details of the implementation, see Bjornson. [10] Also, David Kaminsky is commencing a Piranha implementation at the time of this writing.

The erosion code is an iterative computation on a regular grid (i.e., a height field). The finite differences, as described in section 2.4, reference values at the nearest

neighbors in the grid at each time step. If we employ the simplest efficient spatial decomposition -- subdividing the square height field into smaller squares -- it is readily apparent that we will need to communicate values at adjacent edges of such sub-patches at every time step of the simulation, to provide boundary conditions for the adjacent patch or patches.

A typical simulation runs on the order of 10^4 to 10^6 time steps, though much longer runs may prove desirable once a stable transport scheme has been implemented.

This parallel scheme represents a fairly high bandwidth of communication, as compared to the ray tracing application. This in turn indicates that this Linda application may be better suited to a closely-coupled parallel architecture, where communication is rapid, than to a network-distributed environment. On a shared-memory machine the need for Linda communication might be obviated, as workers could directly access the requisite data in shared memory. Nevertheless, Linda implementation might still be preferred, for ease of coding and portability.

7.4. Conclusions

Computer graphics is an applications intensive research area. As such, we tend to put substantial investments of time and energy into fairly large computer programs. Thus we would generally prefer to maintain portability so that these programs need not be rewritten as hardware platforms come and go, as they so regularly do. Also, the efficacy of graphics programs in general is limited, at least in part, by the computational resources which can be brought to bear in their execution. Furthermore, by convention of the field, the majority of graphics applications are written in the C language. Finally, graphics researchers may generally be inclined to work on problems more directly germane to their field, than issues of parallel computation. Yet, few work in an environment where there is no more than one processor available to execute their code at any time.

Each of these factors indicates the use of C-Linda for computer graphics applications. We have found that Linda readily facilitates simple parallelization of preexisting C language programs, regardless of size, *while maintaining both portability and sequential-execution capacity* . Those italicized points add inestimable value to the C-Linda language, as we simply cannot afford to spend significant time in writing device-dependent code, and we are guaranteed to we lose nothing in the Linda-fication of our precious code.* What we can gain is speedup of our code by a factor nearly linear in the number of processors used, or even substantially better, in exceptional cases such as the procedural height fields. Realism in synthetic imagery is our research goal; success is partly a function of the number of cycles expended in a given rendering. Thus Linda is a boon to our research. Without the power that Linda has made available, it is safe to say that much of the research presented in this dissertation would not have been undertaken, having been ruled out *a priori* as being computationally impractical.

Many computer graphics applications are as embarrassingly parallel as ray tracing. Due to our emphasis on proceduralism, which implies that modelling functionality should be built into the renderer, we have found that parallelization of one code -- our renderer -- automatically parallelized most of the rest of our software applications, such as atmospheric and procedural texture code (which is built into the renderer). And this for just few days of programming time, spent some years ago: we have found that the Linda portions of our code rarely need be revisited, usually only when a paradigm shift occurs, such as moving from shared-bus to distributed computation.

---

* Anecdotally: Having run, and extensively developed, our ray tracing code for almost three years under Linda, the author found it necessary to return to sequential-execution mode during his summer spent at NASA Ames. We were astonished when the (10^4-line) program compiled and ran sequentially immediately, after simply changing the `-LINDA` conditional compilation flag in the makefile.

The best recommendation of Linda, in our opinion, is that we have so little to say about it. We spent a day or two with it; it speeds our computations by several times.

We regard our research as concerning things other than parallel algorithms and strategies; we would no rather spend time describing such things than we would describing the workstations we use, their operating systems, or the car we drive to work in. Those things are not germane to our research, yet we expect them to be in place and to operate reliably. The transparency of Linda leaves it nearly invisible in our applications, and this is exactly how we would have it. And that in itself constitutes a powerful endorsement of the Linda paradigm and the C-Linda programming language.

Linda should find many more uses in computer graphics. In fact it should, in the author's considered opinion, become a standard basis for graphics applications, much as the C language now is.

Appendix B. A Panoramic Virtual Screen for Ray Tracing

Preface

This appendix documents some of the results realized by the author during the summer he spent at NASA Ames in 1991. We present a specific mapping of the entire celestial sphere to a rectinliear virtual screen. It appeared as a section in the book Graphics Gems III. [97]

8.1. Introduction

With ray tracing's synthetic camera model, we can do something which would be difficult to impossible to do with a real camera: create a 360° by 180° field of view panoramic "photograph". The standard imaging model used in ray tracing is that of a pinhole camera with a flat *virtual screen*. We can supplement this model with a cylindrical virtual screen to obtain a 360° (or greater) lateral field of view. By using appropriate angular distribution of samples on the vertical axis, we can obtain a 180° vertical field of view as well.

The *standard virtual screen* model used in ray tracing is equivalent to placing a piece of graph paper in the "world", in front of the eye and perpendicular to the view direction, then firing rays through the little squares (the pixels) on the grid. (A notable improvement to this naive scheme was described by Mitchell. [89]) Changing the field of

view at a fixed image resolution corresponds to moving the piece of graph paper closer to, or farther away from, the eye. This scheme provides a good projection for relatively narrow fields of view, but it breaks down for wide angles: if we attempt to obtain a 180° field of view, the construction of the viewing projection becomes degenerate, as the eye lies in the plane of the screen. A field of view of greater than 180° in this scheme is, of course, nonsense.

Thus with a standard virtual screen we can only approach, never achieve, a field of view of 180°. We can also see that, as we approach the 180° field of view, the distortion introduced by the regular spatial sampling of the virtual screen grows: near the center of the screen the angular width of a pixel is much greater than near the edges (see Figure B.1).

Figure B.1. Angular width of a pixel as a function of position on the virtual screen: pixels near the edge of the screen subtend a smaller angle.

This distortion has the effect of, among other things, giving a sphere imaged near the edge of the screen, a pronouncedly elliptical projection on the image plane.* In a landscape image, features around the horizon get pinched down, degenerated into a line, as the vertical field of view approaches 180°.

---

* For an example of this, see the moon on the cover of the January, 1989 edition of IEEE Computer Graphics and Applications. [108]

Some distortion on the screen due to the viewing projection, is inevitable. This is due to the fact that the projection is an instance of a mapping of a sphere (the *celestial sphere* surrounding the eye) onto a plane (the image plane). The fact is, there exists no mapping from the sphere to the plane $f:S^2 \rightarrow R^2$, such that $f(x)-f(y) = x-y$.* Cartographers have long known this; hence the plethora of cartographic projections for maps of the globe. [113] Thus we may choose among various evils; among various distortions in our images of the world, but we cannot avoid them altogether. (It is interesting to note that designers of camera lenses face the same quandry.)

In this appendix we describe a scheme for sampling a virtual screen in such a way that we can map the entire celestial sphere onto a rectilinear image plane, with (what we deem to be) "acceptable" distortion. Our viewing projection, known to cartographers as a *cylindrical equirectangular* projection [113], generates a lateral-stretching distortion. The magnitude of this stretching grows with distance from the equator, i.e., with distance from the horizontal bisector of the image. Our projection does not become degenerate, however, at a 180° vertical field of view.

8.2. Cylindrical Virtual Screen

With the standard virtual screen, the horizontal field of view is determined by the width of the virtual screen in world space, and its distance from the eye. For a given (finite) world-space width of the virtual screen, as the horizontal field of view goes to 180°, its distance from the eye must go to zero. At 180°, the construction is degenerate, as the eye lies in the plane of the virtual screen.

In the case of the standard virtual screen, jittering aside, samples are generally taken at regular (e.g., equally-spaced) intervals on the screen. The vector defining a primary

---

* One can find a proof of this in a textbook on projective or differential geometry. [29]

ray (i.e., a ray from the eye which samples the virtual screen) is therefore generally determined by taking the vector difference of the sample point on the virtual screen and the eye point, and normalizing the resultant vector. As the sample cells on the virtual screen are equally-spaced in screen space, we can determine the *x* (i.e., horizontal) offset as a linear function of the screen column being sampled:

```
x_offset [i] = i * sample_spacing
```

for `i`$\in$`[-screen_width/2, screen_width/2]`.

For a cylindrical virtual screen, the construction of the primary ray is not quite so simple. We need a linear increment in *angle*, not screen space. This can be accomplished by applying a rotation to a ray directed at the center of the virtual screen. As this requires a matrix multiplication, and thus several floating point operations, we may want to precompute an array of horizontal (relative to the "up" vector for the screen) directions so that we only need perform this matrix multiplication once per pixel column. The size of this array of vectors is, of course, equal to the horizontal resolution of the virtual screen. The accompanying C code illustrates the construction of this array.

This array stores the cardinal horizontal directions for rays sampling the *n* columns of the virtual screen. One might then ask, "how do we handle jittering?" For reasonably large screen resolutions, a simple linear interpolation between adjacent cardinal directions is an adequate solution. (Linear interpolation across large angles would not be a good approximation to the proper cosine distribution, but pixels are generally of small angular size, so a linear approximation to the cosine is sufficient.). Again, this is illustrated in the accompanying C code.

8.3. Vertical Sampling

The cylindrical virtual screen described above allows unlimited horizontal field of view: one can as easily render a 1° field of view as a 720° field of view; fields of view greater than 360° yield periodic images, like wallpaper. There remain problems, however, with the vertical field of view. These include distortion at very wide fields of view, as the screen gets (relatively) too close to the eye, and a degenerate projection at 180° field of view.

The way we have chosen to obtain equal angular increments on the vertical axis, is to vary the vertical increments as the tangent of the *y* (i.e., vertical) index of the pixel on the virtual screen. (We assume that $y=0$ at the center of the screen.) In our scheme we construct two arrays of vectors, one for the horizontal directions and another for the vertical increments. To generalize for an arbitrary "up" vector, the latter is also an array of vectors, rather than of scalar increments. All vertical increment vectors are colinear (i.e., scalar multiples of the "up" vector), and orthogonal to all horizontal direction vectors. To get the direction vector for a ray to sample pixel (x,y) on the virtual screen then, we take the vector sum of entry *x* in the horizontal directions array and entry *y* in the vertical increments array, and normalize. (Again, see the code fragment.)

With this scheme, vertical fields of view greater than 180° yield periodic (but always right-side-up) images of the scene.

8.4. An Application: Martian Panorama

The panoramic virtual screen was developed for a specific application: creating realistic panoramic views of Martian terrains to be viewed in a virtual reality setting. The work was undertaken at the Visualization for Planetary Exploration (VPE) Lab at NASA Ames, in Mountain View, California, where the author worked in the summer of 1991. The goal of the development of the panoramic virtual screen model was to supplement

the existing, comparatively less-realistic, real-time rendering capability for terrain height fields, with the enhanced realism and aesthetic quality available in (far from real-time) ray-traced imagery.

The virtual reality implementation at VPE features interactive viewing of landscape panoramas, using a head-mounted display which tracks the user's movements. In this mode a large (e.g., 6000 by 3000 pixel) static image is loaded into video memory, and the display presents an appropriate viewport on the panoramic scene via real-time pan-and-scroll frame buffer animation. The image loaded into memory may be of arbitrary complexity, as update requires only presentation of a new viewport, as opposed to rendering an entirely new frame. Thus both the update rate and the visual quality are generally better than with real-time rendered animation.

At the outset of this work, VPE possessed the capability of Z-buffer rendering static panoramas using available hardware rendering capabilities. These panoramic views are constructed by abutting a series of flat-screen views edge-to-edge. The net result is a kind of faceted-cylinder virtual screen. (A cylindrical projection equivalent to that presented here can be obtained by reducing the facet widths to one pixel.) As these renderings use the hardware implementation of the viewing projection, they are prone to same kind of vertical distortion as seen when using a standard virtual screen. We sought to recreate this cylindrical projection, with improvements to the vertical sampling, in a ray tracer. A ray tracer gives access to certain realistic effects not readily available in a hardware Z-buffer renderer, e.g., shadows, atmospheric effects, and procedural textures. [91 ,105]

A result of this effort is seen in Plate 8.1, a panoramic view of the Valles Marineris on Mars. Note the extreme bow-shaped distortion of the nearly-linear, parallel valley features. This distortion is a natural and inevitable by-product of the viewing projection we have constructed. Note also that the image was not designed to be viewed in its

entirety, as it is reproduced here, but rather in a virtual reality system, wherein only a viewport on relatively small area of the image is visible at any given time. The idea was to construct an image such that anywhere the user looked, they would see an appropriate, if distorted, area of the Martian environs. The distortion near the bottom of the image serves to discourage the viewer from investigating that area; not altogether a bad thing, as the fixed-resolution terrain data being imaged shows little detail there, where it is closest to the eye point.

We have also implemented a method for creating stereo panoramas suggested by Lew Hitchner at VPE. In this scheme, the eye points describe a circle as the view direction scans about the virtual screen (see Figure B.2). A conventional stereo rendering using two fixed eye points will lack stereoscopy around the direction defined by the line through the two eye points, as the stereo separation goes to zero there. This rotating-eye model yields good stereoscopy over the entire 360° horizontal field. Its implementation appears in the code segment at the end of the appendix.

Figure B.2. Eye point rotation scheme for stereo panoramas.

8.5. Conclusion

We have constructed a panoramic virtual screen for ray tracing. This projection maps the entire view-dependent celestial sphere to a rectilinear screen. Introduction of distortion is unavoidable in this sphere-to-plane mapping; the distortion in this construction of the viewing projection is of a different character than that of a standard virtual screen, taking the form of horizontal stretching of the image as one approaches the poles of the sphere. The resulting panoramic images may be useful for interactive viewing of static imagery in a virtual reality system.

This panoramic viewing projection is interesting, as it is something that is relatively straightforward to implement in a synthetic camera, but difficult-to-impossible to accomplish with a real camera.

8.6. C Code Segment for Panoramic Virtual Screen

```
/*
 * Panoramic virtual screen implementation code fragment.
 *
 * Copyright (C) 1991, F. Kenton Musgrave
 * All rights reserved.
 *
 * This code is an extension of Rayshade 4.0
 * Copyright (C) 1989, 1991, Craig E. Kolb, Rod G. Bogart
 * All rights reserved.
 */

RSViewing()
{
      Float magnitude;
      RSMatrix trans;
      Vector eyeOffset;
      int x, y;

      VecSub(Camera.lookp, Camera.pos, &Camera.dir);
      Screen.firstray = Camera.dir;

      Camera.lookdist = VecNormalize(&Camera.dir);
      if (VecNormCross(&Camera.dir, &Camera.up, &Screen.scrni) == 0.)
            RLerror(RL_PANIC,
                  "The view and up directions are identical?\n");
      (void)VecNormCross(&Screen.scrni, &Camera.dir, &Screen.scrnj);

      /* construct screen "x" (horizontal) direction vector */
      if (!Options.panorama) {
            /* standard virtual screen setup */
            magnitude = 2.*Camera.lookdist *
                        tan(deg2rad(0.5*Camera.hfov)) / Screen.xres;
            VecScale(magnitude, Screen.scrni, &Screen.scrnx);
      } else {
            /* For panorama option, we need an array of screen "x"
             * vectors which we will build later in the code.  At this
             * point, we just construct the required rotation matrix
             * (rotations being about the "up" vector) and point the
             * "scrnx" vector to the edge of the screen.
             */
            Camera.lookdist = 1.;
            magnitude = -deg2rad(Camera.hfov);
            Screen.hincr = magnitude / Screen.xres;
            Screen.scrnx = Camera.dir;
              RotationMatrix( Camera.up.x, Camera.up.y, Camera.up.z,
                        -0.5*magnitude, &trans );
            VecTransform( &Screen.scrnx, &trans );
      }

      /* construct screen "y" (vertical) direction vector */
      magnitude = 2.*Camera.lookdist * tan(deg2rad(0.5*Camera.vfov)) /
                        Screen.yres;
      VecScale(magnitude, Screen.scrnj, &Screen.scrny);

      if (!Options.panorama) {
            /* Construct ray direction for standard virtual screen */
            Screen.firstray.x -= 0.5*(Screen.xres*Screen.scrnx.x +
                              Screen.yres*Screen.scrny.x);
            Screen.firstray.y -= 0.5*(Screen.xres*Screen.scrnx.y +
                              Screen.yres*Screen.scrny.y);
            Screen.firstray.z -= 0.5*(Screen.xres*Screen.scrnx.z +
                              Screen.yres*Screen.scrny.z);
```

```
	} else {
		/* Panorama option: requires that we allocate & fill
		 * horizontal and vertical direction arrays.
		 */
		Screen.horizdir = (Vector *)Malloc((Screen.xres+1) *
					sizeof(Vector));

		/* Set eye separation for stereo rendering */
		if (Options.stereo) {
			if (Options.eyesep == UNSET)
				RLerror(RL_PANIC,
					"No eye separation specified.\n");
			Screen.eyepts = (Vector *)Malloc((Screen.xres+1) *
					sizeof(Vector));
			if (Options.stereo == LEFT)
				magnitude = .5 * Options.eyesep;
			else
				magnitude =  -.5 * Options.eyesep;
			eyeOffset.x = magnitude * Screen.scrni.x;
			eyeOffset.y = magnitude * Screen.scrni.y;
			eyeOffset.z = magnitude * Screen.scrni.z;
		}

		/* Fill the array of horizontal directions and, if stereo
		 * rendering, eyepoints.
		 * The horizontal ("x") direction array contains rotations
		 * of "scrnx".  Each entry requires construction of an
		 * appropriate rotation matrix; rotation again being around
		 * the "up" vector.
		 */
		for ( x=0; x<=Screen.xres; x++ ) {
			Screen.horizdir[x] = Screen.scrnx;
			RotationMatrix( Camera.up.x, Camera.up.y, Camera.up.z,
					x*Screen.hincr, &trans );
			VecTransform( &Screen.horizdir[x], &trans );
			/* Offset the eyepoints for stereo panorama */
			if (Options.stereo) {
				Screen.eyepts[x] = eyeOffset;
				VecTransform( &Screen.eyepts[x], &trans );
				VecAdd( Screen.eyepts[x], Camera.pos,
					&Screen.eyepts[x] );
			}
		}

		/* The vertical ("y") array varies as the tangent of
		 * "scrny".
		 */
		Screen.vertdir = (Vector *)Malloc((Screen.yres+1) *
					sizeof(Vector));
		for ( y=0; y<=Screen.yres; y++ ) {
			Screen.vertdir[y] = Screen.scrny;
			magnitude = 0.5*Camera.vfov -
				    Camera.vfov * ((Float)y/Screen.yres);
			magnitude = tan(deg2rad(magnitude));
			VecScale(-magnitude, Screen.scrnj,
				 &Screen.vertdir[y]);
		}
	}

} /* RSViewing() */


SampleScreen(x, y, ray, color)
Float x, y;	/* Screen position to sample */
```

```
Ray *ray;         /* ray, with origin and medium properly set */
Pixel *color;           /* resulting color */
{
      Float dist;
      HitList hitlist;
      Color ctmp, fullintens;
      extern void ShadeRay();
      int ix, iy;

      /*
       * Calculate ray direction.
       */
      Stats.EyeRays++;
      if (Options.panorama) {
            /* Construct ray direction from vectors in tables,
             * using linear interpolation for jittering.
             */
            ix = (int)x;
            iy = (int)y;
            if (Options.stereo)
                  ray->origin = Screen.eyepts[ix];
            ray->dir.x = Screen.horizdir[ix].x +
                        (Screen.horizdir[ix+1].x - Screen.horizdir[ix].x)
                        * (x-ix) +
                        Screen.vertdir[iy].x +
                        (Screen.horizdir[iy+1].x - Screen.horizdir[iy].x)
                        * (y-iy);
            ray->dir.y = Screen.horizdir[ix].y +
                        (Screen.horizdir[ix+1].y - Screen.horizdir[ix].y)
                        * (x-ix) +
                        Screen.vertdir[iy].y +
                        (Screen.horizdir[iy+1].y - Screen.horizdir[iy].y)
                        * (y-iy);
            ray->dir.z = Screen.horizdir[ix].z +
                        (Screen.horizdir[ix+1].z - Screen.horizdir[ix].z)
                        * (x-ix) +
                        Screen.vertdir[iy].z +
                        (Screen.horizdir[iy+1].z - Screen.horizdir[iy].z)
                        * (y-iy);
      } else {
            ray->dir.x = Screen.firstray.x + x*Screen.scrnx.x +
                              y*Screen.scrny.x;
            ray->dir.y = Screen.firstray.y + x*Screen.scrnx.y +
                              y*Screen.scrny.y;
            ray->dir.z = Screen.firstray.z + x*Screen.scrnx.z +
                              y*Screen.scrny.z;
      }

      (void)VecNormalize(&ray->dir);

      /*
       * Do the actual ray trace.
       */
      fullintens.r = fullintens.g = fullintens.b = 1.;
      dist = FAR_AWAY;
      hitlist.nodes = 0;
      (void)TraceRay(ray, &hitlist, EPSILON, &dist);
      ShadeRay(&hitlist, ray, dist, &Screen.background, &ctmp,
               &fullintens);
      color->r = ctmp.r;
      color->g = ctmp.g;
      color->b = ctmp.b;

} /* SampleScreen() */
```

Appendix C. Essay: Formal Logic and Self-Expression

9.1. Introduction

The digital computer is providing a rare opportunity for the fine arts: the advent of an entirely new process, one with deep conceptual roots in areas normally held as alien to the arts and with a mechanism of expression wholly new to the practice of artistic creation. As a potent engine of interpretation, the computer can be used to project the implications of formal scientific descriptions of a world (any world) into images; in turn these images may become artworks, artworks born of a bizarre new process, a process based upon and entraining the rigor, beauty, and intellectual depth of philosophy, mathematics and the physical sciences. This peculiar process starkly juxtaposes the cold, deterministic machinations of logic, with the ineffable, unquantifiable, spirituality of artistic self-expression. Those two distinct areas of human endeavor, so often viewed as mutually inimical and irreconcilable, come together in service of a common goal of visual aesthetic.

As a practitioner of this process, I will attempt to illuminate both the concerns of this particular artistic process and the significance of its links to the fields of mathematical logic and computer science. While the arguments presented are necessarily technical at turns, I will make every attempt to keep them comprehensible to the intelligent layperson -- as generally-comprehensible as such an esoteric digression may be made to be . . .

9.2. The Thesis

The thesis I propose is this: self-expression in representational imagery may be had strictly through formal logic; this practice marks a discontinuity of significant import in the history of the creative process. In addition, I claim that the resulting artworks resulting are conceptually enriched by the intellectual underpinnings this approach. When an artwork represents the unaltered result of an deterministic logical derivation, it entrains a conceptual depth not commonly achieved in the realm of visual arts.

Only time and our culture can determine the validity of the first two claims I make. The last I can illuminate; that is what this essay attempts to do.

9.3. Foundations

Science is the task of observing nature and deriving potent and internally-consistent descriptions (*models*) of the systems observed. Mathematics is the language of science; it provides both a terse notation and a logically consistent framework in which to couch such descriptions. Computer science is the study of the complex logical system that is the computer; it is largely based on the discipline of mathematical logic: the operation of the modern digital computer is described completely by, and at the lowest level is literally implemented in terms of, the predicate calculus of formal logic.

It is worth pointing out that it is a specific sub-branch of computer science, *numerical analysis*, which concerns itself with the problems of performing mathematical computations with a digital "computer". Note the sudden appearance of quotes around the word computer -- it turns out that this appellation is a misnomer: a (digital) computer is more rightly viewed as a symbol-manipulator, a string-rearranger,* than as a

---

* The term "string" has a very specific definition in computer science, but for our purposes it is sufficient to think of it as an arbitrary sequence of characters or digits.

mathematical calculating device. I point this out because this "string rearranger" model of the computer will be essential to my treatment of the computer as an artistic tool or medium.

9.4. Artwork as Theorem

A *formal system* is a sort of game; it is a fundamental concept of mathematical, or "formal", logic.** For our purposes we may think of a formal system as a given set of strings, called *axioms*, along with a set of rules for performing transformations on, or changes to, those strings. These are called *rules of production*. Successive applications of these rules of production to the axioms (which may be thought of as "input" to the system) constitutes the derivation of a *theorem* in the system. The specific sequence of application of rules of production in the derivation constitutes a *formal proof* of the theorem.

A simple, illustrative example of a formal system if Hofstadter's **MIU** system. [48] In this system, the only recognized symbols from which to compose strings are the characters **M**, **I**, and **U**. The only axiom, or starting (i.e., input) string is **MI**. There are four rules of production which may be applied to the axiom and its successors:

Rule I: If a string ends in **I**, you may add a **U** to the end.

Rule II: If you have **Mx**, where **x** is an **MIU** string, you may add **Mxx** to your collection.

Rule III: If **III** occurs in a string, it may be replaced with **U**.

Rule IV. If **UU** occurs inside a string, you may drop it.

** See Douglas Hofstadter's "Gödel, Escher, Bach" [48] for a thorough, layperson's treatment of formal systems. Bertrand Russell and Alfred North Whitehead's "Principia Mathematica" [132] provides the definitive mathematical treatment of formal systems.

Every string derived from the axiom by these rules of production may be added to your collection of valid strings. Hofstadter challenges the reader to derive the string **MU** from the given axiom **MI**, using the rules of production given for the formal system. Note that there is no ambiguity in these rules, no "maybes" or "kind of likes". The results of an application of a rule are deterministic; but there is choice in the order of application of the rules.

This is a very simple formal system, but it reflects exactly the behavior to which a computer is constrained: modifying strings by the application of well-defined deterministic rules of production.

Why do we bring up this rigmarole? Because this is exactly how a computer operates. We can describe the functioning of a computer completely through this formal treatment; all other "higher-level" functions of a computer are built on top of, and implement different instances of, such formal systems. Formal systems have, in turn, been studied intensively. Early in this century, Bertrand Russell and Alfred North Whitehead [132] set out to map all of mathematics into a single, unifying formal system; their difficulties were shown to be theoretically insurmountable by Kurt Gödel in his famous "incompleteness theorem". (This theorem demonstrates that, for any 'sufficiently powerful' formal system, there exist statements which are neither inconsistent with the system nor provable or disprovable within the system). In short, great minds of our century and before have worked on the ramifications of the machinations of formal systems; in fact, many smart mathematicians and logicians continue to do so today.

The consequence to us, of all of the above, is that when we are using formal logic (i.e., formal systems) we are "standing on the shoulders of giants", intellectually. There is a rich, preexisting mathematical and philosophical body of knowledge in this area, which we are implicitly drawing upon when we use the computer.

How does this concern us, artistically? As it turns out, all computer programs can be mapped into formal systems. Thus, when we use a computer, we are using a formal system; we are utilizing formal logic. Every time we execute a computer program, we are causing the derivation of a theorem in a formal system. "So what?" you might ask. This observation might seem to trivialize, to render vacuous, any claim that the derivation of a theorem in a formal system is in any way something special or intellectually weighty -- after all, computers do it all the time, day in and day out, all around us.

But how often do we call the result art? ("Too often", indeed.) How often is the "artist" cognizant of these arcane machinations? How often can the artist claim to have consciously engineered the entire procedure? When the formal system involved is a computer program written specifically by the artist for the purpose of producing the artwork; when the program itself embodies much or even most of the power to create the work; when the artwork represents something which could not have come into being in any other way; then these observations *vis a vis* formal logic become interesting. Indeed, they gain great import.

9.5. Theorem as Self-Expression

It is one thing to label a theorem derived in a formal system a work of art; it is another to claim that work of art represents self-expression on the part of the artist. Scientifically, the latter claim is weak, as it can be verified only by the artist; no independent formal verification is possible. I maintain that the claim can be valid and is readily verifiable in many cases, if only qualitatively (as opposed to quantitatively). There exist examples of such artworks -- the pure output of a computer program -- wherein it is readily evident that something of the artist's soul has been bared. As an example I offer "Blessed State", Plate 9.1.

Claiming self-expression through formal logic obviously involves massive constraints on what constitutes successful practice and an acceptable result in the artistic process. Yet another significant set of constraints is generated by the requirement that the works be *representational*. Abstract expressionism is an honored aesthetic in its own right, and formal systems and the computer can be -- and have been -- used in this context. But the requirement of literal realism in formal imagery spawns a host of problems and concerns which are only starting to be addressed, primarily in the research literature of computer graphics. Representationalism in synthetic imagery is, in general, an open problem.

Many of the problems of generating realistic-looking synthetic imagery have been solved, albeit often in *ad hoc* ways. Many such problems yet await satisfactory solution. For example, specular reflection from glossy surfaces has been handled to nearly everyone's satisfaction by *ray tracing*; however, general realistic lighting models -- including atmospheric scattering of light and interreflection between matte-finished surfaces -- are still under active development. In short, there are some things we can do very well with the current techniques of computer graphics; others which are imminently doable but not-yet-done; and some which are, and are expected to remain, "hard". As an active researcher in the field, I am involved in the effort to move more phenomena from the category of "doable" to that of "done". As a result, my own artworks more often than not serve not only as a form of aesthetic self-expression, but also as illustrations of techniques new to the field of computer graphics. This adds a dimension of technical significance to the works; however, I generally intend this to be transparent to the casual observer.

In fact, one of my key intentions as an artist is to keep this entire esoteric process which I am describing transparent, to make it invisible to the viewer. There are a variety of reasons for this: First, I do not wish to immediately and automatically invoke the instinctive fear of mathematics that the average person feels (myself included). Second,

it is a research goal to have the image look as natural, i.e., non-computer-generated, as possible; thus the formal process should be thoroughly sublimated in the result. Third, and most important, is that it would be no better than arrogant and obfuscatory to require the audience to confront and grapple with these issues -- the images should be able to stand on their own as aesthetic visual statements, outside of this technical context. I say "let them, or let them fail."

9.6. Deterministic Formalism and the Creative Process

An artist requires constraints, if for no other reason than to narrow down the "search space" wherein the desired result is sought. The formal logic approach certainly provides a rigorous set of constraints on the creative process. It also provides some interesting side-effects.

The determinism of the logic involved means that the result is reproducible: repeated runs of the same program with the same input provide, modulo the occasional hardware glitch, the same output. The artwork is reproduced *exactly*. (Or at least the numerical *metarepresentation* of it is; more on this later.) This is true despite the fact that randomness is an essential element in all my images -- the randomness employed is a deterministic randomness; it is not "truly" random, but what we computer scientists refer to as "pseudo-random".* Pseudo-random processes are simple yet sophisticated constructions from the discipline of *number theory* which are, for practical purposes, fully random (i.e., they lack predictable order or structure) yet are deterministic and therefore reproducible.

---

* Note that throughout the following text, I will freely use the term "random", generally meaning "pseudo-random" and thereby implying an ultimate determinism.

The fact that I constrain my artworks to be purely the output of a computer program insures that they feature this peculiar reproducibility. This could never be true of a painting, for instance, as a brush stroke is not an exactly reproducible act, on the microscopic scale at least. In the case of a computation the result is a string or, at the lowest level, a number or sequence of numbers or digits. This string or number can be checked character-by-character, digit-by-digit, for *exact* fidelity; there is no ambiguity or latitude for imprecision in the representation. Viewed in the light of computational result as artwork, and artwork as representational self-expression, this determinism and exact reproducibility are downright bizarre.

9.7. Distinguishing the Process

It is worthwhile to take a little time to point out what distinguishes this process from the more traditional practices of fine arts, such as painting, sculpture and photography.

9.7.1. Dimensionality

The product of this process is a two-dimensional image; this characteristic it shares with painting and photography. Like a painter or photographer, the artist is responsible for choosing an interesting point of view and framing for the image. As with a camera, a geometrically precise projection of the three-dimensional world onto the image plane is performed; painters have much greater latitude here. Like a photographer, one is free to roam the three-dimensional world, even to employ cinematography to add motion in a temporal exploration.

Unlike either painting or photography, the artist is responsible for the creation of the entire world being imaged: there are no preexisting objects "out there to be found" and creatively imaged; all objects and all interesting visual detail must be created explicitly. The elegant means we have for creating such visual complexity are at the heart of what makes this process successful and interesting.

9.7.2. Visual Complexity: Fractal Models

Fractal geometry [77] is the key to generating potentially unlimited visual complexity in my work, and in computer graphics in general. Fractal geometry is a language of shape, similar to the language of planes, circles, spheres, triangles, cones and cylinders of the more familiar Euclidean geometry. But as Benoit Mandelbrot has observed [77]:

> "Clouds are not spheres, mountains are not cones, coastlines are not circles, and bark is not smooth, nor does lightening travel in a straight line..."

The vocabulary of shape of fractal geometry provides, describes such complex natural shapes with great elegance.

There are two keys aspects to fractal descriptions of natural forms: *self-similarity*, or the repetition of similar shapes at different scales, and *randomness* in the model. The first means that we need only describe one fundamental shape plus the relationship of its manifestation to the scale at which it is manifest -- a very simple description indeed, for an object of potentially-unlimited complexity (the complexity is simply a function of the number of different scales at which we manifest the basic shape). The second aspect, randomness, is the key to having the resulting shapes look natural, rather than man-made or (worse still) computer-made. Control then takes the form of shaping statistical distributions in random processes, rather than explicit specification of exact form. Thus we exchange exact control over the shape for power in automatic generation of complex shapes: I have God-like powers in a Universe which has a will of its own, and the constant capacity to surprise even me, its creator.

9.7.3. Purity of Algorithmic Process

Of course, I could employ my God-like powers in this synthetic Universe to intervene and make specific, local changes wherever I saw fit. In adherence to a self-imposed constraint of process, however, I do not allow myself to do this. This often closes off

access to the shortest route to a desired result (as with a desired hue in a given highlight), by disallowing various local intrusions and modifications to the world or the image which would, in practice, be relatively easy to execute. What is gained, however, is purity of algorithmic process. Creation of an image becomes a dance with the opportunities and serendipity granted by the powerful, random fractal models which I create, embellish and (more or less) control. By disallowing post-process meddling with the results of various algorithmic processes I employ, I gain two compelling benefits: legitimacy in illustrating the descriptive power of these abstract fractal models, and claim to an elegance in the creative process -- the image is indeed a theorem proved, in one pass, in a formal system.*

Adherence to principles of algorithmic purity legitimizes one of the key claims I make about the significance of this process: that it entrains the intellectual depth of logic, mathematics and computer science as its foundations. In practice, it entails the pure use of formal logic to obtain the desired result.

Again, another (more or less arbitrary) constraint I impose upon this process is that the result represent self-expression. Expressionism is a practice the popularity of which has waxed and waned through the history of the fine arts; I do not claim that it makes my work in any way "better", I only note that it constitutes a significant constraint upon what I, as an artist, consider to be a successful result

---

* One could contest the claim that this is a one-pass process, on the grounds that the terrain models I image are usually generated outside of the rendering process, in a separate step. In my own defense I point out that A) I apply the same rules of algorithmic purity in the terrain-model generation process, B) that program could readily be incorporated into the renderer thus coupling them, and C) this is not always the case: I am moving towards a completely *procedural* process wherein even the terrain model is created on-the-fly, as the picture is being created (see Plates 4.1 and 4.3, which are entirely procedural).

9.7.4. Proceduralism

These concerns lead us to *proceduralism*. [126] Proceduralism is the practice of abstracting complex behaviors into relatively terse *functions* or *algorithms* which do not contain specific information about details of the phenomenon, but rather encode that behavior in a formal set of instructions which specify that behavior everywhere it might manifest itself and which may be evaluated only when and where such information is desired.

Thus in the procedural approach, a "virtual world" is abstracted into a compact procedure or set of procedures. These procedures are in turn controlled by a relatively few parameters which affect (only) *global* control. Alvy Ray Smith [140] called this *database amplification*; I refer to the process of creating landscape images within this paradigm "playing God in a found Universe" -- I may have God-like powers over these worlds, but in practice, because of the randomness they embody, they behave as if they have a will of their own. Furthermore, they have an ineffable sense of having existed *a priori*; of somehow being inherent in the timeless, universal formal procedures that specify the and of always having existed there as an aspect of Nature, or at least of Mathematics, just waiting to be discovered. As an artist, I simply interpret these forms visually. Thus they may represent, at least in part, "found art". But there nevertheless remains enormous latitude for the exercise of aesthetic judgment in the development of any given image. It is, after all, but one out of an unimaginably huge, if finite, multitude of images which might have been selected (more on this later).

9.7.4.1. Functions and Algorithms

Proceduralism in practice consists of devising *functions* which in turn are implemented as *algorithms*, or unambiguous sequences of instructions telling the computer exactly what to do, for any given input. Functions are a mathematical concept.

They may be viewed very simply as contraptions which assign values given as input, to other values -- the output. The input and output values might be very different: input may be numbers and output colors, or other stranger and more subtle mappings.

Mathematically, we refer to the action of a function f like this:

$$f: D \rightarrow R$$

which simply says that f sends input values from D (the *domain*) to R (the *range*) It is useful to distinguish the set of possible input values D from the set of possible output values R, as they may be quite different kinds of things.

The simplest kind of function is a *scalar-valued function of a single variable*, denoted $f(x) = y$. (We use lower case letters to refer to specific values, upper case to refer to the entire set from which the value may be chosen.) A *scalar value* is just a single number. A function of one variable has only one input value.

Most interesting functions are the more complex *vector-valued functions of several variables*, denoted $f(x_1, x_2, ..., x_n) = [y_1, y_2, ..., y_m]$. This particular function takes a number (n) of input values, and maps them to another set of m output values. Such functions are more common in my images. They typically take more than three values as input: the three spatial coordinates of the location where the function is being evaluated (as the function is defined over all of space) plus a set of variables controlling the behavior of the function. They output some small number of values, such as the red, green and blue components of a certain color and a spatial vector used to modify the apparent orientation of a surface (as with the water in Plate 9.1).

It is the concoction of functions like this with interesting visual behaviors, which constitutes the first step in this formal creative process. These functions are small parts of a computer program, which are embedded in the much larger program which orchestrates the creation of the picture. Examples of such functions in action can be seen in the

ripples in the water in Plate 9.1, as well as in the roughness of the moon and the coloring of the mountains. Each of these effects issues entirely from the functions evaluated on the surfaces there (believe it or not, the water is a perfectly flat plane, and the moon is a perfectly smooth sphere!) The fact that the functions are defined over all of space allows us to evaluate them anywhere we desire. Thus the moon is carved out of an infinite block of "moon-ness", the mountains out of a block of snow, rock and forest, and the water out of an infinite expanse of abstract "sea".

9.7.4.2. Global Parametric Control

The values x_i (i denoting the numbers 1 through n) which serve as input to our functions are known as *parameters*. The parameters beyond the three spatial coordinates at which the function is being evaluated, determine the behavior of the function. The way these functions are usually constructed, those values affect the function's output *everywhere* in space. This amounts to *global parametric control* of the function's behavior.

In practice this means that, for instance, I may exactly specify an color for a light source; if I dislike the resulting hue in a particular highlight (a *local* effect) I may change the color of the light source accordingly, but this changes tones everywhere that light falls in the scene. Similarly, if I dislike the shape or location of a given wave in the water or mountain peak in the terrain, I may change it, but this change will also affect all other waves or peaks and valleys. The randomness at the heart of the fractal models I use grants both enormous flexibility and expressive power, but it also entails complete abdication of control over specific details in relation to their global context. While this global parametric control represents a profound creative constraint, it also entails an enormous (and often elegant) simplification of the final stage in the creation of the image: after the program is written, all that is to be done is to select values for these parameters.

9.7.5. Self-Expression vs. Conceptualism

Conceptualism supplanted self-expression in the recent history of visual art, after representationalism was deemed a solved problem and abstract expressionism lost its novelty. But this new process reopens the problem of representationalism. It may thus push us back a few steps in the cycle of aesthetic evolution (or is just forward, one step?) And again, I wish to emphasize throughout this essay the depth of the conceptualism inherent in this process, and to cast a faint glimmer of light into those depths.

9.7.6. Lighting

The artist's responsibility for lighting in synthetic scenes brings this process into relation with lighting as used for photography and stage performances. This responsibility is something new for landscape rendering, where artists have traditionally relied on serendipity in Nature to provide striking effects. As the author of a synthetic world, we will find nothing there that we do not explicitly create.

The process of providing lighting is exactly analogous to stage lighting. We have light sources with color, brightness, direction, and area of influence. We can position those lights wherever we want. We can have as many of them as we like (though in practice I rarely use more than two -- a warm sunlight and a cool skylight). In addition, we are responsible for specifying, mathematically, the interaction of light with surfaces in the scene: are they mirror-like, glossy, or matte? or something different, perhaps completely unnatural? There are no set limits here. This mathematical treatment of light and color also marks a new practice in the visual arts; we will expand upon it later.

9.7.7. A Model of the Creative Process

A particularly fascinating view of the parametrically controlled creative process is that of *searching n-space for local maxima of an aesthetic gradient.* Let me explain what

I mean by this: We have created a procedural, parametrically-controlled model of a synthetic universe. Say there are *n* independent parameters in that model and the specification of its projection onto the image plane. As these parameters are independent, we can think of each as representing a *degree of freedom* or an additional dimension or direction in which we may move. Taken together, the *n* parameters define an *n-dimensional space* or *n-space* for short. In this space we are free to move not just up and down, right and left, or forward and back, but in a whole lot of other (abstract) directions as well. This may seem obtuse to the layperson, but mathematicians never hesitate to work in spaces with many more dimensions than the familiar three of our everyday world.

The task of the artist is first to create these *n* parameters (*n* being usually around 200 to 300) and their (deterministic) meaning, through creating the procedures or functions which they drive, then to "tweak" the values of these parameters to obtain a satisfactory result or image. The creation of the parameters in formulating the formal system corresponds to defining the *n*-space; the process of refining their values, or choosing axioms to start with, corresponds to searching that space for *local maxima* of an *aesthetic gradient*. A *local maximum* is location in the space from which all directions lead "downhill", that is, it is a kind of hilltop in *n*-space. "Downhill" is defined by the *aesthetic gradient function* -- the completely subjective (non-deterministic) assessment on the part of the artist of what constitutes a "better" image, in terms of the parameter values. Obviously, this so-called "function" is not unambiguous: its value will depend on the criterion by which the image is being assessed, and even on the mood of the artist at the moment of evaluation.

Ambiguity notwithstanding, this *n*-space gradient-ascent model is more than just entertaining: it points out that a given image represents merely a *local* maximum of the aesthetic gradient field. Other, more global maxima ("higher hilltops", corresponding to

"better" pictures or possibly "better" self-expression) undoubted exist in the abstract *n*-space of potential images defined by the formal system. This is very much akin to noting that a photographer might have gotten a better shot by choosing a different vantage point or time, except that we have much, much more control here. Creating and searching this *n*-space is a strange way of obtaining self-expression.

9.7.7.1. Searching N-Space for Aesthetic Maxima

What does this look like, in practice? I have a bunch of numbers, usually about two to three hundred, which define the entire scene I'm creating (other than the landscape itself, which consists of thousands of numbers which, again, I don't allow myself to change or fiddle with). This is a lot of number to deal with. And it turns out that if you change more than one or two at a time, the effects are usually conflated, and you can't be sure which change accomplished what effect. Thus I spend long hours, massaging the values one or two at a time, until I am sufficiently satisfied or exhausted to "call it a picture".

This is a very tedious process. It is also very obscure: no one else can hope to use my programs -- the meanings of the parameters are simply too obscure for another artist to practically deal with. In fact, I am only really cognizant of their intended effects at the time that I create the functions, this intent is quickly forgotten in the complexities of my work and daily life. If later I need to reconstruct that meaning, I generally have to go back and look at the computer code that I've written, and figure it out by inspection and the memory that that inspection triggers.

This is not a highly desirable interface or way of working. When people ask me "can other people use your programs, too?" I have to answer "no". (I certainly lack the time and patience to explain or document all of these things.) This deplorable state of affairs I would attribute to the youth of the method -- it is certain to be improved over time.

Powerful mathematical methods can be brought to bear in such endeavors. *Principle components analysis* may be used, for instance, to reduce the dimensionality of the parameter space, and to maximize the effects of changes in parameter values (though such reorientation of parameter vectors may destroy an original intuition as to parameter meaning).

9.7.7.2. Genetic Algorithms

One very promising method for managing the creation and search of the high-dimensional parameter space is *genetic algorithms*. In the genetic approach, we borrow some concepts from biology, namely *genotype, phenotype, mutation,* and *sexual reproduction*. *Genotype* is the encoding of an organism's form in its DNA, while *phenotype* is the physical manifestation of that coded form in an actual organism. *Mutation* is the spontaneous change in the encoding itself, and *sexual reproduction* is the recombination of genotype information from two individuals, by "mixing and matching" parts of their genetic code. This is a powerful approach to creation -- after all, it appears to have gotten us to where we are today, as intelligent sentient beings.

Richard Dawkins popularized the genetic approach in his book "The Blind Watchmaker". [28] Several artists are using genetic algorithms to create striking works (though they are not representational, in the sense that I am using here). Karl Sims creates wonderful abstract images very rapidly with his genetic software, running on a massively parallel supercomputer. [137,138] My own experience with his program showed it be an astonishingly fecund process. And it as simple as can be: the computer puts up a sequence of images, you pick one you like which the computer proceeds to mutate for you, or you pick two which the computer then "breeds" for your pleasure. Mutation is random, and "natural" and sexual selection are performed by you, the user. William Latham also uses genetic methods in creating his fantastic sculptural forms of Cambrian beasts-that-never were. [151]

While this genetic approach to the management of procedural models is incredibly promising, it is currently limited to the creation of such free-form objects and images as Latham's and Sims'. It is, unfortunately, not immediately apparent how to apply these methods of selection and random mutation to non-biological natural phenomena.

9.8. What the Process Is Not

To further distinguish the process, it is worthwhile to point out certain aspects of what the process is *not*, to clarify by defining the negative space around it.

9.8.1. A 2-D Canvas

One thing this process is not, is a flat canvas. While the final image is indeed two dimensional, its creation takes place in 3-D. We are responsible for the creation of an entire three dimensional world, which we proceed to image by projecting it onto a film plane like a photographer, only doing so with mathematics.* The potential of the process will be expanded when we gain the capability of rendering scenes at video frame rates -- then the viewer will no longer need be passive, but will be able to enter the synthetic world and explore it, as one moves about to inspect a piece of sculpture. In an immersive VR environment, this is foreseen to be quite an exciting development, though more akin to entertainment than art perhaps.

It is important to me, as an artist, to emphasize a certain point: The really interesting uses of the computer in the creation of artworks will not be in the traditional role of a canvas and paintbrush. Certainly, the computer can function as such and offers some unique capabilities, such as infinite erasure and reworking capabilities, not possible with

---

* Because the projection is described in the abstract, with mathematics, we may employ projections which would be difficult to impossible to obtain with a camera (see Appendix B and Plate 8.1 for an example). Indeed the projections may be completely non-intuitive, as for example with the projection of a four-dimensional quaternion Julia set down into three dimensions, and subsequently down to the two dimensions of the image plane. [92]

paints. But that does not mark a significant conceptual breakthrough, merely incremental progress for an established process. Not that there is anything wrong with using the computer in this way -- most of the best computer art has been, and will continue to be, produced in this way. I simply wish to emphasize that the process I am describing has very little in common with that, aside from their common aesthetic disciplines of composition, color usage, and so forth. The means of creation are utterly different, and it is only the new one which is truly significant as an intellectual event in the history of art.

9.8.2. Local Control

Almost every established process in the visual arts involves local control. Details will be manipulated in isolation from the whole. Any given brush stroke, for instance, while it certainly may indirectly affect and be indirectly affected by its global context, represents an absolutely local act. It does not directly affect anything beyond the area where the paint is applied.

Changing a global parameter, in contrast, immediately and directly affects everything, everywhere its function has influence. Thus I again wish to emphasize the contrast with, for instance, painting and sculpture, where the work is usually realized incrementally by a series of fundamentally local actions. When working with global control only, we have a much less precise control over details, but gain in return something akin to the power of "painting with a broad brush" -- we can cover a lot of territory with a single action.

9.8.3. "Of the Hand"

As the only access to expression is through the formal logic of the computer program, there is no "evidence of the hand" in the final work (or if there appears to be, it is illusory). Some may find this anathema, but it is important to point it out as a distinction of the process. The mechanism of creation that I use, is extraordinarily abstract and removed from the product. This is part of what is interesting and bizarre about the

process: that such prosaic imagery comes about through such indirection and abstraction. I claim that this is significant.

9.8.4. Pure Mathematics

I am often mistaken for a mathematician. That I am not. While all of the models employed are based on logic, and many are mathematical models of natural phenomena, the mathematics I employ is generally quite simple compared to what a "real" research mathematician would be involved with.

Pure mathematics, after all, usually shuns applications or associations with "reality". And what I am up to, is recreating reality as we know it (more or less).

9.8.5. Computer as Creator

Finally, and most importantly, this process does ***not*** represent creative action by the computer. A computer, given no instructions, will just sit there dumb as a rock, if a little warmer. A computer (on a good day) will cheerfully do exactly what you tell it to do, with blinding speed and precision. It will never do anything useful that you, the operator, did not describe explicitly and in excruciating detail. Remember: the computer operates as a formal system, and that admits no ambiguity and no choice, only deterministic cut-and-dried yes-or-no instructions and conditionals. Certainly, the complexity of the instructions we hand the computer rapidly surpasses our human ability to track every detail thereof, while the computer never loses track of one iota. But the computer remains a simpleton; a very fast and capable simpleton, but a simpleton nevertheless. If we puny humans were given eons and inhuman patience, we could track, produce, and reproduce every tiny detail of what the computer does -- only we'd make a lot more mistakes along the way.

The point is, the computer acts as a powerful tool, maybe even like a semi-intelligent slave/apprentice in practice, but is in no way the creator, the author of the product. It simply did as it was manipulated to do, as with a paintbrush in the hand of a painter. The main difference is that the form of the manipulation is highly abstract and rigorous, and very different than the physical manipulation of tangible media that we are more familiar with in the visual arts.

9.9. The Process in Action

How does one proceed to create an image through this process? First, we have to posit our model of the world; next we must map it into a formal system, a computer program. Then we devise axioms, or input to the program. Lastly, we run the program to create the output which we will interpret as an image. This output is, like the input, in the form of a string of symbols or values (i.e., numbers; ones and zeros). A string is hardly an image; therefore we call this the *metarepresentation* of the image. This metarepresentation still requires considerable, sophisticated machinery and methodology of interpretation, translate it into the intended image.

We can then further subdivide the process of creation into two separate undertakings: creating the metarepresentation and interpreting it. This essay concerns itself primarily with the first; it is here that the bulk of the intellectual content resides. The second represents primarily an engineering problem, though there is a considerable dose of color science involved and that is none too simple in itself. [163] In artistic terms, these two parts correspond to *process* and *medium*: the first concerns itself with the machinations of artistic creation while the second is about producing a physical representation. After the first part is done, all we have generated is some still highly-abstract and intangible form. It is the second step which maps this abstraction into something which can be perceived in a sensible way, and maybe even felt, held, or hung on a wall. It is interesting that the two, process and medium, are so neatly partitioned in this way of working.

9.9.1. Creating a Metarepresentation

Again, the first step is to create the metarepresentation: the theorem, the string, the sequence of digits, the one huge number, the signature on a magnetic or optical storage medium, or the image file; however you care to view it.

9.9.1.1. Creating the Formal System

We begin the process unconsciously as a young child: observing and cataloging sights, phenomena, and behaviors in Nature. Over time we build some potent and internally-consistent models of Nature and the behavior and visual manifestation of phenomena there: clouds, mountains, water, light and color, to name but a few. Some training in the sciences teaches us the practice of mapping this intuition into formal, mathematical models of the behavior of natural systems. We become familiar with many such formal models that scientists before us have devised and refined, and we learn where to find descriptions of such models -- in the scientific literature. Becoming a practitioner of computer graphics, we learn the practice of mapping such models into formal systems which the computer can efficiently use to generate pictures. (Note I say "efficiently", as the scientific literature consists mainly of picayune and non-general models, along with some very elegant and general ones which are simply not suited to the practice of image synthesis: witness the wave model of light. This is a potent, elegant model of Nature which the computer just can't deal with, as it involves too much complex calculation. What we require are models with potent descriptive capabilities, and which admit to reasonable computational implementations.)

It is this formulation of a model of Nature and mapping it into a computer program which constitutes the first phase of the process. It is in the creation of the functions, the writing of the program, that we create the parameters and give them their functional "meaning". The program, again, represents the rules of production in the formal system,

which will be repeatedly applied to the axioms, or the input, in the process of deriving the theorem that is the result or metarepresentation.

Our tools at this stage are such abstractions as shaping functions such as polynomials with continuity in a desired number of derivatives, logic in the form of conditional "if/then" statements, and algebra as applied to color (more on that later). Largely by combination and recombination of a series of standard building blocks, such as fractal functions, bump maps, color maps, etc., we construct a relatively small set of functions which we intend to use for creation of the image.

The process of generating the formal system is so involving that, in practice, almost all of my own images have come about as verifications of some abstract idea which I was attempting to map into such a system. In this sense they represent illustration of the model being developed; I use the word "illustration" deliberately, despite the stigma which may be attached to it in the visual arts. Keep in mind that in our new paradigm, representationalism is no longer a "pedestrian concern" -- it is once again an unsolved problem, and we are working towards solving it. Thus the work cannot be dismissed as "mere illustration" or "simply representational"; these are honorable labels in our context.

There is one inevitable and undesirable side effect of this stage of the process: *parameter proliferation*. In the process of developing a potent model of complex phenomena, we almost always end up introducing a large number of parameters which control the behavior of our models or functions. This means that the artist will be faced with a bewildering array of values which must be assigned, to create an image, and refined, to create an artwork. Again, we currently know of no way around this time, but that may just be a symptom of the youth of the endeavor.

9.9.1.2. Generating Axioms

The next step is to formulate values for those multifarious parameters. This is not quite as bleak a prospect as it may sound, as the same intuition which drove the formulation of the model and the functions, also informs the choice of values for the parameters. Thus we are not groping in complete darkness; we generally have a good idea of where to start and how to change the values, to obtain the desired effects.

Nevertheless, as described before, this is a long and tedious process in practice. The goal is the creation of an *input file* which will be fed to the program when we execute it to create an image. The process consists of sitting at a terminal, working in a text editor to change the strings in the input file, running the program with the modified file, inspecting the results, going back into the editor to make changes, running again, and so forth. I generally spend the equivalent of about two to six weeks of full time work in this loop, for one of my finished images.

But it is important to note that the procedure isn't quite as neat and sequential as I've presented it so far. These first two stages are not really so distinct -- while I am refining the parameters to the functions, I am generally simultaneously developing and extending the functions themselves. Since the theorem proved is determined by both the axioms and the rules of production, we naturally massage both of them more or less simultaneously as we develop that theorem into the image we desire. Furthermore, as even the author of the formal system would not generally care to be confronted with the need to specify *every* single parametric value in the model, many of the axiomatic values are hard-coded as constants in the program and thus are not part of the input file. Thus the separation of axioms into input and rules of production into program, is not very precise. It would actually be easy to be very thorough about so partitioning the system, but in practice it is neither necessary nor desirable.

9.9.1.3. Deriving the Theorem

Once we have a set of production rules and axioms we may proceed to derive a theorem, to create a image. Again, this means firing up the program and running it with the given input file. Execution time for the program varies widely for my own images, from a minimum of about a minute to a maximum of several days. This at a rate of tens of millions of operations per second* -- there are obviously many, many steps in the derivation of the theorem, far more than any human being could ever hope to perform.

Again, each of these operations (other than memory accesses) represents a transformation to a string: one sequence of ones and zeros is translated, deterministically, into another. The sum total transformation is that of translating the input file into an image, an image which may represent self-expression in an artwork to the person executing the program.

9.9.1.4. The Loop of Scientific Discovery

Gregory Nielson points out [110] that this process embodies the basic loop of scientific discovery: one posits a formal model, observes the behavior of the model in comparison to Nature, then refines the model and makes further observations, proceeding in an iterative loop. Perhaps the main difference between mainstream science and this practice in computer graphics, is the time required for a single iteration of the loop. For a scientist, it may be decades, even a lifetime or longer, whereas in computer graphics it is typically measured in minutes.

---

* I almost always perform my computations in *parallel* on several computers, each of which is capable of performing several million operations per second.

9.9.1.5. The Role of Intuition

Both the scientist and the artist are driven by intuition. No scientist derives potent models of Nature through exhaustive search of all the possibilities provided by first principles. Neither does any mathematician originally get to the proof a hard theorem by simple extrapolation of logical principles. No, they both will retrofit their conclusions with a deterministic logical derivation -- those deterministic formalisms are what both mathematics and the physical sciences are founded upon, after all. But if not for the role of intuition, computers would immediately leave us all in the dust, so to speak. It turns out that, ultimate expositions in deterministic proof notwithstanding, no mathematician can explain exactly *how* he or she originally conjectured the result, or even how they arrived at the formal derivation finally presented. If they could, we could program a computer to do it faster. No, in the creative process scientists, mathematicians, and artists all rely on intuition to the same degree and in exactly the same way. It is in aspects of their respective products that they so differ. None of us knows precisely how to get where we want to go *a priori*, but we all conjecture worthwhile goals and eventually intuit some path that indeed gets us to our desired ends. Such is the magic of human intelligence, and this is what continues to distinguish us from any "artificial intelligence" yet devised.

9.9.1.6. The Role of Serendipity

Finally, we must note the role of serendipity in this formal process. The fact is, we don't always know exactly what the results of our derivations will be, and we can't realistically expect to always be able to accurately foresee the behavior of our models (the emerging science of chaos is making that abundantly clear).

Serendipity comes from the unforeseeable, as with random models; from the unforeseen, as with a model which has not yet been subjected to thorough intellectual

scrutiny; and from errors and mistakes, as with typos and program bugs. Each of these factors has played an important role in the genesis of my own images. Plate 9.1, for example, did not come from a preconceived idea for a visual composition. Rather it came from the unforeseen, or a sort of bug: I had moved the program which I expected to generate the mountains seen in Plate 1.2, to another computer. This computer had a different random number generator, which I had not foreseen in writing and porting my program. Thus when I ran the program I was confronted with a wholly unexpected landscape, which serendipitously harmonized with the large moon I had put in the sky, but not yet scaled own to a reasonable size. Perhaps every artist can tell similar tales, but here it is important to see that, though we work through a formal, deterministic process, we still dance with chance and the unknown.

9.9.2. Interpreting the Metarepresentation

As I said before, the theorem we derive is nothing more than a string of symbols in the computer's memory. Nothing tangible or image-like about that, yet. But we do intend an image, and we have (thankfully) a preexisting machinery of interpretation for that metarepresentation. I will outline that machinery, and sketch how that machinery is currently woefully inadequate to the creation of works of art. This, too, is a symptom of immaturity of the process and medium, and will change for the better with time.

The problem at hand is to map the formal metarepresentation, i.e., the string or numbers, to a certain appearance in a physical manifestation. Obviously, we have enormous latitude in this transformation, as the metarepresentation has no intrinsic meaning: it is merely the deterministic result of applying a series of abstract transformations to some input symbols; there is no meaning in that other than what we (more or less arbitrarily) ascribe to it. Also obviously, we always had a certain interpretation in mind for the result, throughout the process.

Unfortunately, when we leave the idealized, uncertainty-free world of formal logic and its embodiment in the computer an enter the "meat" world of physical manifestations, we lose the grace and precision of Boolean digital representation and enter the fickle, imprecise, and heinously ill-defined world of things analog and continuous. The real, "analog" world is far less well-behaved than the formal and deterministic world in which have been dwelling. We face a whole new, different, and largely unrelated set of problems, problems usually without the clean, irrefutable solutions we've been using. This is the world of color monitors, color printers, and photographic reproduction. This is where we do well to hand our theorem over to the artisans skilled in working with such things, and beg, cajole, plead with and threaten them to do our bidding.

Such is the Real World, with which our abstract idealizations must eventually interface.

9.9.2.1. Numbers as Colors

We have a huge string, usually of hundreds of millions of symbols, or *megabytes* of data, which we wish to interpret as a picture. "How?" one might ask. Well, again fortunately, there are conventions for this interpretation which we can follow to make our lives easier.

The main convention is to regard the string as a sequence of numbers, usually comprised of eight 0/1 symbols or digits each. Such an eight-bit string can, again by convention, encode a single number between 0 and 255, inclusive (those 256 values correspond to the 2^8 possible distinct combinations of eight ones and zeros). According to the *tristimulus* model of human vision, we can encode all perceptible colors into combinations of exactly three primary colors. [70] By convention, we interpret our string of eight-bit numbers to represent consecutive triples of values for those primary colors. Thus we know what the derived string "means": it is a sequence of color values for *pixels*

(pixels being the atomic colored dots of which our final image is composed). These color values proceeding a canonical order, as do the pixels they are meant to represent. (There is a wide variety of standard digital image file formats, such as GIF, TIFF, JPEG, etc., but they all simply represent different encodings of the same information.)

This is an arbitrary interpretation, but then so is any interpretation of an intrinsically meaningless formalism. By being as specific as we can be about the intended meaning of the metarepresentation, we take on another arbitrary set of constraints that greatly simplify our task.

9.9.2.2. The Finite Number of Possible Outcomes

As each pixel is represented by three eight-bit numbers, it can have exactly one of $2^8 \text{x} 2^8 \text{x} 2^8 \approx 16$ million values. If we have, say, $2^{10} \approx 1$ million pixels in the image, then the entire image can take on exactly one of $2^{8^{10}}$ values. While $2^{8^{10}}$ is one very large number, it is finite. Thus, at a given number of pixels and a given number of possible colors, there is a large but finite number of pictures which can be represented. The actual number will be far less than $2^{8^{10}}$, of course, as no human observer would be able to discriminate between the different visual representations of many slightly different metarepresentations.

9.9.2.3. Additive vs. Subtractive Color

Another factor which distinguishes working with the computer from most of visual media, is that we work in an *additive color space*, versus the more familiar *subtractive* colors. The difference is that when using pigments, on is *subtracting* color energy out of the impinging light which illuminates the work. If there is no illumination there is no visible work, and presumably the optimum illumination is with white light, as it contains all the colors, in equal proportions, to start with. Thus a red pigment absorbs the green and blue energy in a white illuminant, and reflects the red.

In computer graphics, we start with a dark (optimally, black) surface, and *add* in the color energy we desire. Thus a red area is simply made to emit red light, and the work is visible in complete darkness (and conversely, may be hard to see well in a brightly lit environment). This convention came about because the standard output device for computer graphics is a television monitor, as opposed to a sheet of paper.

The main difference between additive and subtractive color, is that the primary colors are complementary. In subtractive color (contrary to what you were taught in grade school), the primaries are magenta, yellow and cyan. In additive color, they are red, green, and blue. Thus, for instance, we must learn to think of yellow as a sum of red and green (not immediately obvious), and brown as a dim version of an orange yellow.

We also find that images developed on the luminous monitor may not be nearly so striking, when mapped to a subtractive medium. Plate 9.1 is one of the few examples in my own experience, that looks fairly good in both media -- though there is a magical luminous quality on the monitor, which is missing in a reflective print.

There is a hard-copy solution to this: back-lit transparencies. Unfortunately, these are quite expensive to produce; the light box alone can cost several hundred dollars (and be ugly to boot) for good-sized print. Back-lit transparencies to have one significant advantage over reflective prints, however: the lack of surface detail, such as the impasto of a painting, is obscured as one's attention is simply not naturally drawn to the physical surface in such a display.

9.9.2.4. Archival Reproduction

Color reproduction from digital data is a difficult problem. It seems unlikely that a television monitor would be accepted as an artwork, by the art consuming public. Monitors are large, heavy, low-resolution, and, well, they look a big TV, not something to hang on your wall. The market is, and will remain for some time to come, for (thin)

two-dimensional images on a surface, like a painting or a print; not for a four-inch-deep, ungainly light box with a dangling power cord to be plugged in somewhere, and certainly not for an expensive, high-quality video monitor.

Thus we face the problem of making high-quality reflective prints of the artworks, which both the artist and the collector can be happy with. Achieving the artist's satisfaction requires a large investment of time and money on the his or her part, to find a photographic or offset printer to produce such prints (there are currently no other viable high-quality, large format color printing media). The artist can expect to spend several thousand dollars on this, and what is produced is not generally a one-of-kind object, but a series of prints. This affects the market for the work; it is not like a painting, but more like a lithograph or photographic print series.

The second criterion, making the collector happy, complicates the reproduction problem further. Serious collectors require *archival* artworks -- pieces which can be expected to last 100 years, without serious fading or other such degradation. This rules out color photographic prints, as none are considered archival (gloss Cibachrome prints are considered to be semi-archival, i.e., they may last about 50 years; no backlit transparency even comes close, due to the high, UV-rich light levels in a light box). What this leaves, then, is four-color offset printing. Such prints can be made on acid-free paper, or at very high (400 dpi) line screen resolution using carbon pigments on a polyester substrate. The former is the equivalent of a quality lithographic print; the latter is superarchival, with a life expectancy of about 500 years, but is very expensive and constrained to modest sizes.

These problems mean, in practice, that color reproduction is largely an unsolved problem. It is not realistic to expect the artist to be able to sink several thousand dollars into each finished work, as artists are notoriously indigent. Thus I, for one, consider myself to be an artist without a product.

9.9.2.5. What is the Product?

Given that there were a product, this raises the well-known question in computer art: what exactly *is* the "product"? Is it an object, such as a color print? Is it the metarepresentation, the image data? Is it the formal system? Or is it the formal system plus its machinery of interpretation, i.e., the program and the computer that runs it?

Of all of these possibilities, the only reasonable one is the first: it is some tangible hard-copy object or print. The metarepresentation is not particularly valuable, as it is *exactly* reproducible, due to the determinism of the process which creates it. It cannot be the formal system -- I have years invested in the program that creates all of my images; I would not sell it for any price. And even if I did, I am capable, in principle at least, of recreating an exactly equivalent formal system, and indeed upon a sale of this sort I would immediately have to do my best to do something very much like that, just to be able to get back to work. This would lead to disgruntlement among the collectors of my work, no doubt. Finally, it is absurd to propose the last option, that the work consists of both the program and the computers that run it: even if I were to give the software away for free, the hardware would cost well over $100,000 and would serve no purpose whatsoever to the collector, as it is exactly replaceable by me. That is, it has absolutely no uniqueness associated with the image -- it would be like selling the paintbrushes and easel with a painting; they are generally quite replaceable and of no relevance to the finished piece.

9.10. Discussion

Let me move now to a brief discussion of the implications of this new process.

9.10.1. What Role Intent and Understanding?

As I have pointed out, the computer can be quite readily be used as a novel canvas, paint, and paintbrush, for use as with prior two-dimensional media for the visual arts. Used that way, the resulting works will be essentially "of the hand", and thus part of the existing continuum of two dimensional media.

When the artwork is *algorithmic*, issuing directly and unmodified from a formal description, it becomes more interesting. When the algorithm is deterministic, it becomes more interesting still (after all, artists such as Sol Lewitt have produced non-deterministic algorithmic artworks, and issued the algorithms therefore, for some time now).

But I maintain that deterministic algorithmic artwork is only truly significant when the artist is also the author of the formal system, and can claim to understand it thoroughly and to have intended (modulo serendipity) to create the result produced. Thus artworks created by someone else using my software would lack the conceptual significance, even if they were more aesthetically sophisticated. If Picasso had invented a "Picasso engine", and others used it to create Picasso-like works, these works would simply would not be quite the same as an original Picasso, after all -- even if others were able to "improve upon" Picasso.

The artist can only really claim to have accomplished self-expression through formal logic, when he or she authored, for that specific purpose, the formal system through which the expression is obtained.

9.10.2. What of Turnkey Systems?

What then, of turnkey software for creating computer art? There are many powerful programs becoming available which unlock the substantial potential of the digital medium, and there will continue to be ever more, of greater sophistication and novelty.

Programs such as Adobe Illustrator and Photoshop are revolutionizing the way many artists, and perhaps most designers, work.

There is, and will always be, a role for such systems. Indeed, the vast majority of practicing "computer artists" will always use such "canned", preexisting software. It would be absurd to propose that all, or even many, artists pay the substantial dues required to get up to speed in this peculiar process I am describing. No, this process will always exist and be practiced on the fringes -- there will never be more than a handful of people who are qualified to use this process, requiring as it does an extensive background in art, science, mathematics, logic, and computers.

Let me use an analogy: there have been great drivers, for almost as long as there have been cars. But these drivers are rarely the builders of the cars they drive. Indeed, no single person can expect to build an automobile of any sort, much less a race car, without the help of many others (no more can I expect to build the computers I use, or to have invented every technique I apply). But a good driver, whose vehicle is largely the result of his own creative vision, would always be a special competitor, though he might never turn in the fastest time.

There will always be room for the virtuoso users of tools provided by others, and they will always predominate the field of performers. Likewise, there will always arise, here and there, now and again, visionaries with "the madness of the poet" who will create their own tools and do with them what might never have occurred to others. And there is, at least, always some significance to being the first do have done something of interest and significant difficulty. This process I describe is probably such an undertaking.

9.10.3. The Role of Traditional Media

New as it may be, it does not stand outside of precedence. As the result is a two-dimensional work, all the rules and discipline of two dimensional art apply, most saliently

those of visual composition and color usage. As the modelling is done in three dimensions, rules of form and lighting also apply. When animation is undertaken, the rules of cinematography will come into play. When we produce a tangible product, any sort of physical manifestation, all the rules and practice of the medium in which that product is executed will apply. We cannot presume to create a new art form from scratch; we will need to borrow and appropriate everything we can use, from what has come before.

We may, however, need to invent a viable new medium in which to represent the product. It may be that computer art as a whole will not truly come into its own, until some essentially new display technology, such as large, bright, flat-panel color displays or laser projectors, comes into common usage.

9.10.4. Mastering the Process and Medium

As painting has been mastered, so must the computer media and this new process. Painting, photography and sculpture did not reach maturity overnight; neither can we expect computer art to do so. The fact is, the computer artwork has not yet been produced which could stand a side-by-side comparison with, say, a van Gogh painting. My own best image would pale, stood beside a Bierstadt. The austere beauty of the underlying formalism strips computer generated imagery of the fascinating, continuous behavior of such a medium as oil paint -- there is simply nowhere near the amount of information in a standard digital image file, as there is in a well-executed painting. The range of scales over which a good painting is interesting, is generally much larger than that for a computer-generated image, fractals notwithstanding. We will need to include such complexity, or simply find another grounds for legitimacy with as much weight, before we can call our works truly fine art.

One interesting and useful distinction was drawn by Ansel Adams, who used the analogy that the negative is the score, and the print the performance. In this analogy, we currently have the capability in this new process to produce the negative, almost literally. But we currently lack the means of translating this score into an impressive performance. That is the challenge I have outlined above.

One wonderful distinction of the process I've presented, is its simultaneous use of both analytic and intuitive thinking. Sitting at the computer creating an image, one must rapidly switch back and forth between the "right brain" mental faculties required to assess aesthetic issues, and the "left brain" analytic processes required to deal with the logic-based machinery of production. This is certainly an unusual way to go about producing a visual artwork; its closest analogue may be in musical composition.

9.11. Conclusions

This new process may mark a truly novel event in the history of creative process in the fine arts. Provided, of course, that the artist intends, understands, and can in some valid sense take responsibility for, the formalisms behind the product. I am claiming that a number, along with the correct (and well-defined) interpretation can represent artistic self-expression, that this number can be derived deterministically, and that the machinery of this derivation adds significance to the result.

Be careful to note that I am not claiming that the machine is self-expressing. A computer has no more aesthetic ability than any inanimate object, and indeed, it can be more refractory than most. The expression is the human artist's; the computer is the tool through which the artist makes his or her statement.

Editorially, I wish to add that it is fortunate that landscapes are my prediction for self-expression. As an painter and photographer, I have always preferred to render landscapes. When I entered the field of computer graphics research, landscape modelling

and rendering were in their infancy; it has been my pleasure to substantially improve the state of the art in such, through the course of my doctoral research at Yale under Benoit Mandelbrot, the father of fractal geometry. In a remarkable bit of serendipity, I appear to have been the right person in the right place at the right time. There was a narrow temporal opportunity, which I happened to precisely meet-- had I shown up a few years earlier or later, the opportunity would not have existed.

9.11.1. Constraints and Opportunities

Let me quickly recap the significant constraints and opportunities of this new process, as I see them.

9.11.1.1. Working in Three Dimensions

While sculptors and stage designers have worked in three dimensions for millennia, the peculiar way in which we do is significantly different. We differ at least in scale: we are creating landscapes, entire planets, and even, potentially, a small synthetic universe. The challenges are different, and appropriate practice will therefore undoubtedly be different. Thus we will need to invent and refine some new methodologies. While landscape rendering has as rich a precedence as any other area of visual art, prior landscape artists were not generally responsible for creating their entire scene, in full three dimensions. Soon, when interactive exploration of our scenes becomes possible, we may find ourselves confronted with responsibility for guiding, through whatever means we find artful, the explorations of visitors to our worlds.

9.11.1.2. Algebraic Color

While we cannot and should not expect to redefine the rules of color usage, we cannot manipulate color in the ways which visual artists are accustomed. First, we work in the unfamiliar additive color space, where heuristics for mixing colored pigments are either

inverted or simply invalidated. Second, there is no physical system in which color interacts -- it is all simply a model. Nothing happens at all, except for what we specify. We may seek to have those specifications mimic as closely as possible, the behavior of the real world (a very hard thing to do, in general), or we may bend or break such laws in our system. In any case, the specification and interactions of colors on surfaces is couched in the mathematical language of algebra -- certainly an unfamiliar way of dealing with color for the average studio artist. Colors are all numbers; they mix by the arithmetic operations of addition, subtraction, and multiplication, and they are often modulated by exponential operators.

Color theory for computer graphics is often elegant, and is quite internally-consistent. But it is not something familiar to the average artist.

9.11.1.3. Proceduralism

Proceduralism, the practice of encoding behaviors in formally defined, deterministic functions, is at the very heart of this process. Strict adherence to this practice is whence the intellectual significance we claim for the process emanates. We can gain a wonderful elegance in this approach, as with the fractal models which can so succinctly describe behavior of potentially unlimited visual complexity. It marks a significant challenge, to maintain a discipline of using only such relatively simple logical constructs for visual expression, and it is a significant constraint to work only with global parametric control.

There can come great benefits from such discipline, though. Imagine a procedurally defined planet, or array of planets, which possesses detail everywhere, detail which the artist did not explicitly and laborious specify, but which issues directly and automatically from the functions from which the model is composed. Plate 4.3 is an example of such a model; the animation "Spirit of Gaea" currently in production, will serve as a proof of this concept. In this animation, the point of view will move in from deep space, up to the

planet, down through its atmosphere to its landscape, up very close to the terrain (the equivalent of about ten feet away), then back out into space. This will all be accomplished with a single procedural model, and while rendering will be far from real-time, it is only a matter of engineering to get to where we can move around like that interactively, at will. That will be an unprecedented development.

9.11.1.4. Formal Logic

Our use of formal logic for self-expression entrains with it the precision and lack of ambiguity of mathematics and science. Lack of ambiguity is not familiar, or even desirable, in the arts. But such precision in the creative process does not in any way preclude the kind of deliberate ambiguity which lends depth and interest to art. Rather, it stands beneath, as an unusually solid foundation for artistic creation. Its use allows scientific models to mapped into creative opportunities -- something which I personally find an exciting undertaking, having always been fascinated with the beauty of such models in their own right. Finally, our basis in formal logic entrains with it, the intellectual depth of the philosophical discipline of logic and the mathematical models of the sciences. These are deep conceptual roots, which we have only begun to tap.

9.11.2. Some parting Questions

I will conclude with some questions, questions which do admit to immediate answer.

How do we obtain self-expression through formal logic?

I claim to have done so, but I can no more tell you how, than the average painter can tell you precisely how they painted a particular painting.

How do we know when we have?

If my claim is valid, it should be verifiable. There are only two ways to do this: ask the artist, and ask yourselves, the audience. Success or failure will be found to be a fickle thing.

So what's new here?

I have attempted to illuminate that in this essay, but I feel very incomplete about it. It seems that my own analysis of this event preliminary; I may spend the rest of my days fleshing it out. I have this feeling that, as is typical in new areas of intellectual inquiry, the ideas formulated and presented so far may be vague and somewhat half-baked. Certainly my own arguments could stand a better foundation in the history of art. But I hope it that the time is ripe to begin to expound them, that they might be honed or discredited through the dialectic.

Is it important?

Time, of course, will tell, at least in the eyes of our culture. Obviously, *I* think so. But then, I am primarily trained as a scientist rather than as an artist, and I am certainly no art historian. Nevertheless, I do know enough to recognize and put my professional reputation at stake, that something big is going on here. Unfortunately, the requirements for a full appreciation are backgrounds in mathematical logic, natural sciences, and computer science, as well as aesthetic training and sensitivity. Thus the audience who can apprehend, and perhaps be impressed by, these arguments is necessarily small.

Will it fly?

Again, time will tell. If I continue to suffer occasional visual inspiration, I may help bring it to maturity as an art form. I am certainly counting on others to help, and hopefully to create works which will make my own appear crude and preliminary. But I am not alone. I quote my colleague Judson Rosebush [126]:

> In practice, proceduralist computer art is among the most contemporary products of our culture, and will increasingly be appreciated as a major art movement by this and future generations.

If Mr. Rosebush and I are correct, we may be witnessing one of the truly definitive events in the history of Art.

Appendix D. Essay: Mathematics -- the Language of Nature

10.1. Introduction

This appendix consists of an essay written for, and appearing on the inside cover of, the 1993 Fractals Calendar [92] which I produced during the spring and summer of 1992. The target audience is the interested, but uninformed, layperson. The content is therefore expository rather than technical; its scope goes as far as original philosophical speculation.

This essay derives its basic structure, and even some of its content (specifically, the quote of Galileo and the numbers for fractal dimensions) directly from the similar essay by Richard Voss in the previous year's Fractal Calendar. [153] Much thanks is due to Dr. Voss for paving the way so smoothly for this essay and, indeed, the entire 1993 calendar project.

10.2. Mathematics: the Language of Nature

A basic tenet of Western science is that the behavior of systems in nature can be accurately described with mathematics. In 1623 Galileo wrote:

> Philosophy is written in this grand book -- I mean universe -- which stands continuously open to our gaze, but it cannot be understood unless one first learns to comprehend the language in which it is written. It is written in the language of mathematics, and its characters are triangles, circles, and other geometrical figures, without which it is humanly impossible to understand a single word of it; without these, one is wandering about in a dark labyrinth.

Scientists today still regard this statement as essentially correct. However, Galileo's language of nature -- that attributed to Euclid of ancient Greece, and known today as Euclidean geometry -- was a bit short in vocabulary. Geometry is a language of shapes, but many of the shapes in nature lie in Galileo's vague category of "other geometrical figures", not within the scope of Euclid's shapes. They require another kind of geometry, one unimagined in Euclid's or Galileo's time.

Recently the mathematician and natural philosopher Benoit Mandelbrot observed:

> "clouds are not spheres, mountains are not cones, coastlines are not circles, and bark is not smooth, nor does lightning travel in a straight line..."

This observation heralded his introduction in 1975 of the new branch of mathematics known as **fractal geometry**.

Fractal geometry can be thought of as a new dialect of the mathematical language of shapes (geometry). Unlike Euclidean geometry, fractal geometry deals with the very complex shapes found in nature: things like trees, river networks, and billows of smoke. The difference between Euclidean shapes, such as planes, spheres and cones, and fractal shapes is that Euclidean shapes are all *locally flat*: if you look at them closely enough, they become flat, planar, and boring, while fractal shapes can be complex *at every scale*, that is, they may possess an infinite wealth of detail.

Where does this complexity come from? It could come from two places: there could be a vast quantity of essentially independent information -- such as the nicks, scratches and stains on an old pair of tennis shoes -- or there could simply be a *repetition* of an underlying form, at a variety of *scales*. The latter type of complexity is the secret of

fractal geometry: **self-similarity**, or the repetition of form over a variety of scales. Note that the former kind of complexity is hard to conceive of except as the result of an accumulated history of independent events over a substantial period of time, while the latter -- fractal complexity -- can often be succinctly described by a simple specification of the underlying shape, plus the relation of its manifestation to the scale at which it is manifest.

Many forms in nature are a combination of both types of complexity; the addition of the fractal vocabulary of shape to our scientific language of nature has revolutionized our ability to describe, in the internally-consistent, formal and deterministic language of mathematics, the forms and order we find in nature. The images in this calendar -- particularly the fractal "forgeries" of natural scenes -- attest to the success of fractal geometry as a language of nature. Even the wholly abstract fractal images have a "natural" feel to them; they echo the complexity of form so common in nature.

It is interesting to contrast the places where we find Euclidean versus fractal shapes: Except for the near-perfect spheres and ellipses of soap bubbles, astronomical bodies (planets and suns) and their orbits, it is a bit unusual to find Euclidean shapes in nature -- they generally appear in the domain of man-made objects, such as buildings and machines. Fractal objects are generally too complex to be explicitly constructed by man; nature, on the other hand is full of them: mountains, trees, clouds, and fractures (indeed, the word "fractal" intentionally shares the Latin root of "fracture"), to name but a few. Hence it is appropriate to think of fractal geometry as a "geometry of nature".

Some artificially-generated fractal shapes appear to imitate phenomena in nature such as mountains, clouds and trees. Others appear regular and man-made, such as the Sierpinski Gasket and the von Koch Snowflake. Yet others, such as the Mandelbrot and Julia sets, appear to be entirely abstract in form. Yet all fractals share the characteristic of self-similarity: they appear more or less the same at a variety of scales. This self-

similarity may be **exact**, as in the case of the Sierpinski Gasket and the Menger Sponge; it may be **statistical**, as in the case of fractal mountains; or it may be more difficult to characterize, as with the Mandelbrot set. Although the character of the self-similarity differs among fractals, they all have in common the fact that when you look at them ever more closely, you see more of the same, and they all have in common a potentially-unlimited complexity which is a result of this repetition of form over a potentially-unlimited range of scales.

Fractals also share a common mathematical characteristic: the **fractal dimension**. The fractal dimension is an analytic measure of the "wigglyness" of a fractal line or the "roughness" of a fractal surface. It is a number which agrees with our everyday sense of dimension (three dimensions define space, two dimensions a plane, one dimension a line, and zero dimensions a point) but which has non-integer values: for the Sierpinski Gasket it is $\log(3)/\log(2) = 1.58...$; for the von Koch Snowflake it is $\log(4)/\log(3) = 1.26...$; and for fractal mountains it is usually between 2.1 and 2.3. The larger the fractal dimension, the more wiggly the shape, and as the fractal dimension of, for instance, a line approaches the integer value of the next higher dimension (in this case, going from 1.0 to 2.0), the fractal curve becomes *space-filling*, that is, it fills the entirety of some part of the next higher dimension (in this case, a plane). Not an immediately intuitive notion, this idea of fractal dimension, but it becomes so surprisingly quickly -- one comes to think of the fractal dimension as simply a numerical measure of just how convoluted the curve or surface is, and with some practice one can estimate its value fairly accurately just by eye. Cumulus clouds, for instance, have a fractal dimension that's usually about 2.2 to 2.3.

10.3. A Bit of History

Around the turn of the century, there occurred something of an upheaval in the world of mathematics. Mathematicians such as Weierstrass, Cantor and Peano conceived of bizarre constructions: strange "dusts" of unending complexity; functions with wildly

unpredictable behavior; curves that could fill space or have no tangents. They boasted that these entities had no counterpart in nature and were somehow inherently intractable; having discerned what they could about them, the mathematicians labeled them "monsters", ill-begotten and unwanted children of mathematical speculations. Interestingly, while they had studied the beasts formally, they were never concerned by what they actually *look* like and pictured only the simplest among them. In general, the tedious and copious calculation required to make such a pictures would have to await the invention of the modern digital computer.

As a young man in the 1940's, Benoit Mandelbrot had an uncle (also of the name Mandelbrot) who was a well-known mathematician in his own right. One day he showed the young Benoit a mathematical paper by Fatou, telling him "there is a career to be made in this, for the person who can figure it out and pursue this work". Young Benoit had a look at the paper, and concluded "this is definitely not for me". Over the years, however, the ideas germinated in the back of his mind, and he came to be working on problems less far-removed than he himself realized, at the time. As is usually the case with scientific discovery, fractal geometry took form in his mind as if by a slow process of awakening, rather than an instantaneous "aha!"

Mandelbrot was employed by IBM as a research scientist in 1958. This gave him ready access, in the 1960's, to the then-exotic digital computer. Armed with the requisite mathematical insight, the computer provided the tool he needed to explore the latter-day "monsters" of mathematics and discern something more of their true nature. What he found is illustrated in the images in this calendar.

Fractals and the computer are inextricably intertwined. While ordinary mathematics often takes the form of *equations* such as $y = f(x)$ which may be solved explicitly, fractals are generally specified in terms of *recursive procedures* such as $f(x_{i+1}) = f(f(x_i))$, that is, the mathematical operation is applied repeatedly to its

previous result. This is a recipe for tedium, for a human calculator, but it is exactly the kind of thing a computer does best: vast quantities of relatively simple operations. The fractal equation is then:

[*simple operations*] times [*a mind-numbing number of repetitions*]

= [*mind-boggling complexity*].

The "simple operations" term means that fractals are relatively easy to program into the computer. The "mind-numbing number of repetitions" term means that the computer will do it much faster, more accurately, and with less pain and complaint than a human. The result -- "mind-boggling complexity" -- was a complete surprise, and means that we need to invoke special measures to deal with the problem of making sense of the results of the computation.

It turns out that the way we human beings are wired up, of all our senses, vision provides by far the greatest bandwidth (amount of data per unit time) for getting information to the cerebral cortex, to the higher brain centers. Then, rather than looking at the results as a bunch of numbers (which is how the computer deals with them), we can make *pictures* out of them. The original fractal computer graphics mark one of the earliest uses of the computer for the purposes of **scientific visualization**. In fact, fractal computer graphics are being used even now in the service of making scientific discoveries: the images make possible the recognition of patterns and relationships which would otherwise be lost in floods of computational data. Serendipitously, the complexity inherent in these synthetic fractal images often comes hand-in-hand with an astonishing beauty, making accessible to every person some of the much-touted abstract beauty of pure mathematics. Indeed, some fractal geometers would claim that the images provide a sort of intuitive, visual proof of the existence of such beauty.

10.4. Fractals in Science and Philosophy

What, one might ask, are fractals good for? Well, obviously, they generate some convincing models of natural phenomena such as mountains and clouds for use in computer graphics imagery, and they provide some very compelling abstract pictures. But recently, something like one third of all physics papers submitted to journals for publication at least mentioned fractals somewhere. The fact is, fractal geometry is *so* new on the scientific scene that its uses are still being puzzled out. The modern philosopher Martin Heidegger argued that language itself allows for, even *generates* a world. If this is true, we can expect a fundamentally new language with a hitherto-untouched domain of expression to generate a new world view. Fractal geometry is in the process of doing just that.

Perhaps the most profound impact of fractal geometry to date is in the new science of **chaos**. Scientists have recently discovered order in natural systems, where previously there had seemed to be none; the language of fractals provides the vocabulary with which they can speak of this order; without it, that order probably would have remained unrecognized. The science of chaos deals with the behavior of **nonlinear dynamical systems**, that is to say, "equations that model natural systems well, and how they evolve through time". Scientists have long used **linear approximations** to **nonlinear systems** as a matter of mathematical expedience. The nonlinear mathematics models nature more accurately, but is intractable in comparison to the linear approximations. When computers made it possible for scientists to begin to cope with these previously-intractable nonlinear systems, they discovered something very surprising: they call it **deterministic chaos** or **sensitivity to initial conditions**; it means that *any* perturbation to the initial state of the system, no matter how small or seemingly insignificant, will cause the system to **diverge**; i.e., to evolve into an arbitrarily different future state, within a finite period of time.

This is a counterintuitive notion: we would expect systems that started off in very nearly the same state to continue, forever, to evolve upon reasonably parallel paths. Not so, we find, and this has profound philosophical consequences. In the 1600's Descartes and Newton, as natural philosophers, fleshed out a world view so compelling that, if the average educated person in our society today stops and thinks about it, it seems to be "the obvious way that things are". In the Cartesian universe with Newtonian dynamics, if we knew 1) the position and velocity of every particle in a closed system and 2) the rules for their interactions, and we had sufficient power to compute all those interactions, we would have the power to predict the future, forever, for that system. Obviously correct, right? If our "closed system" were the entire universe, this would have profound philosophical implications: there could be no **free will**, it implies that we are all witless automatons, mere puppets in some sort of deterministic, already-written cosmic script. It would affirm the nihilistic philosophy of fatalism, and undermine the basis of human morality: that we have a *choice* in matters, and that what we choose to do -- and not to do -- makes some kind of a difference.

Perhaps fortunately, our century has seen a series of repudiations to this deterministic model of the universe. First, Einstein dealt a blow to Cartesian geometry with his theories of relativity: space/time is curved, and a line is straight or curved relative to the observer's frame of reference. Then came quantum mechanics, wherein the Heisenberg Uncertainty Principle states that, for particles on the subatomic scale, we can know their position *or* the velocity, but not both -- this torpedoes the first premise for computing the future in a deterministic universe. Very recently -- within the last 15 years or so -- the science of chaos has driven a second nail into the coffin of the deterministic universe: Suppose we *did* know the position, velocity and rules of interaction for all particles. Then *any* error, no matter how small, in the initial data, in its representation (or that of intermediate results), or in the computation, would lead our computation-of-the-future to be *wrong* by an arbitrary amount within a finite period of time.

We are left with a bizarre world view: The universe may indeed be deterministic (determinism means that the past absolutely determines the future, that there is no "choice" at any time, and therefore no true randomness and no free will) *yet that observation is useless*, it is meaningless, it does us no good whatsoever -- we may as well be living in a nondeterministic universe brimming over with free will! This is philosophically profound, and evidence that our fractal geometry has indeed "generated a new world", in the sense that it has fundamentally changed the way we see our universe, as well as the way we expect it to behave. Such is the character of scientific revolutions.

10.5. Fractals in Art and Music

Back on the more prosaic plane of everyday life, we find that fractal geometry is beginning to influence the visual arts. As a language of shape and form of unprecedented richness, it is fairly easy to see that it can provide a new language for art. Fractals images are most readily generated with computer graphics, but computer graphics as a medium for the fine arts is nascent: the medium and process need to be developed and refined, and artists with madness (in Plato's sense of the word) and understanding must come to work with it. Fractals provide a visual dialect of natural forms, couched in the formalisms of mathematics. The latter makes it challenging -- even alienating -- to artists, while the scientists and mathematicians who are prepared to deal with that are rarely trained and practiced in the discipline of visual aesthetics. There is beauty to be found in deterministic fractals such as the Mandelbrot set and in random fractals such as fractal mountains; indeed, that beauty often has the character of seeming to exist *a priori*, somehow inherent in the (in fact always-deterministic) procedures used to calculate them. The role of the artist is in exploring the fractal forms and interpreting them, visually, in a way that brings aesthetics to the forefront.

Another fascinating association of fractals with the arts is in music. It turns out that music from all cultures is fractal in an essential way: there is a repetition of form over a

variety of scales, "scale" in this instance being over time. This fractal character is somehow so essential that a sequence of random notes sounds quite "musical", if it is only constrained to have fractal changes. That is, if the random "score" is constrained, say, to look like the profile of a mountain range, we hear something that is tantalizingly close to a non-random, human-crafted musical score. Much as the **fractional Brownian motion** we use to create fractal mountains lacks some of the features of real mountains, yet nevertheless captures the essence of "mountainous", a random fractal musical score somehow has the essence of music, without the structure that a human composer would impart.

Musical composers such as Wuorinen and Legeti are (consciously) using fractals in their compositions, yet this area of exploration of the applicability of fractals has also barely begun. An early synthesis of fractally-informed music and fractal imagery was the experimental performance by Mandelbrot and Wuorinen of "New York Notes" at the Guggenheim museum in the Spring of 1990, with a reprise performance at the Alice Tulley Hall of Lincoln Center in the Spring of 1991. This and other seminal fractal artworks make it clear that the association of fractals and the arts is a potentially rich field, with much exciting work yet to be done.

10.6. The Fractal Calendar

This calendar of fractal images was designed with a combination of emphasis on aesthetics and on variety of imagery. Some attempt has been made to represent the major types of fractals: **deterministic fractals** such as the Mandelbrot set and its variations, as well as the whimsical "non-Platonic solids"; and **random fractals** as manifest in landscapes with their fractal mountains, clouds, waves, trees, and surface textures, as well as the **diffusion-limited aggregation** or **DLA**. I have also striven to represent a wide variety of fractal generation methods: iterations on the complex plane and in the quaternion numbers; iterated function systems; graph grammars or L-systems; midpoint-

displacement and spectral synthesis methods for terrains; procedural textures for surface detail and artistic image processing; simulation of physical processes; and genetic algorithms for image synthesis. Some of the images are intrinsically two-dimensional, others are of three-dimensional objects; some are abstract and some represent "fractal forgeries of nature" and are thus quite realistic. On the whole, they represent some the most interesting of recent fractal images, and I hope that they will offer both pleasure and stimulation, and perhaps even inspire some future fractalists.

Appendix E. Color Plates

Plate 1.1 "The Road to Point Reyes" was executed in 1983 by a team six of the top researchers in computer graphics at the time. [24] It features many of the kinds of models which we have sought formalize and document in the course of our own work, as well as some of the foibles we wish to remedy (such as the creases in the distant terrain). This image appears by the courtesy of Rob Cook. [21]

Plate 1.2 "Misty Mountains" illustrates both a sedimentary rock strata texture and the exponential-by-altitude "mist" atmospheric function.

Plate 1.3 "Zabriskie Point" is an example of an image which is simultaneously successful as art and scientific illustration. The image was conceived and executed as an illustration of the mirage model seen in the foreground. [95] It also embodies some of the greatest aesthetic subtlety and sophistication in any of our images. Note that the clouds are the same model used in Plate 4.3.

Plate 2.1 "Bahama" demonstrates the fractal edge generated by the hexagon-subdivision terrain generation scheme and our terrain generation-time model of fractal drainage networks. [78] This is a grid-traced height field of dimension 1207^2, making it a record-breaking scene, in terms of the number of unique geometric primitives present, at the time it was rendered in 1987.

Plate 2.2 This patch demonstrates the flexibility of noise synthesis in modulating frequency content of a stochastic function. The patch goes from planar to space-filling, as the its spectral content goes from $1/f^{\infty}$ to $1/f^{0}$.

Plate 2.3 This noise-synthesized patch demonstrates varying fractal dimension from 2.0 to 3.0.

Plate 2.4 In this noise-synthesized patch, we vary spectral content with altitude, to get a first approximation of eroded terrain.

Plate 2.5 "Spirit Lake" shows the above model, used in a realistic rendering. Note that the relatively smooth character of the terrain allows us to tessellate it with large triangles (see the vicinity of the peak). This implies a small height field, which can be rendered rapidly. Also present is the physical model of the rainbow described in section 4.2.

Plate 2.6 Here we have a noise-synthesized patch where the spectral content is modulated by horizontal position, to generate a ridge line.

Plate 2.7a-d Noise functions and fBm generated from them: a) `Noise` as 2-D intensity plot. b) `VLNoise` as 2-D intensity plot. c) `Noise`-based fBm. d) `VLNoise`-based fBm.

Plate 2.8 Here we see a heterogeneous terrain function used to emulate terrain morphology on a very large scale. Note the variety present: plains, rolling foothills, and alpine areas; all in one model.

Plate 2.9 "Hell" is a fanciful rendering which features a fixed-resolution, heterogeneous ridged-fBm terrain model with a procedural flame texture applied to simulate lava. The terrain model is 1600^2 height field patch, rendered in Rayshade with hierarchical (16^2-tree) grid tracing.

Plate 2.10 A noise-synthesized terrain patch before erosion simulation.

Plate 2.11 The above terrain patch after 10^6 time steps of the fluvial erosion simulation.

Plate 2.12 A noise-synthesized height field before thermal weathering.

Plate 2.13 The above height field after 100 time steps of thermal weathering.

Plate 4.1 "Slickrock" is a procedural terrain with adaptive level of detail. This rendering method makes possible use of a basis function with a discontinuous derivative, for construction of the terrain. Thus the terrain model features coherent ridgelines at all scales. The exponential mist model with the Rayleigh scattering approximation provides atmospheric perspective: note how the distant mountains are bluer and of lower contrast, giving a sense of truly grand scale. The background is black; Rayleigh scattering makes the sky blue. Note that the color of the atmosphere is off-white, as seen at the horizon.

Plate 4.2 "Carolina" demonstrates the utility of the exponential mist model. The effect obtained here is exactly what Lynch describes. [75] The Rayleigh scattering approximation adds realism.

Plate 4.3 "Gaea & Selene" illustrates the planetary atmosphere model and some procedural models of planets. The primitive geometric objects in the scene consist of three spheres (the Earth, clouds, and Moon), plus the atmosphere. All visual detail is provided by (rather elaborate) procedural textures.

Plate 4.4 This detail of "Gaea & Selene" highlights our Rayleigh scattering model, as applied to the planetary atmosphere. [103] Note the continuous segue in the homogeneous atmosphere model from blue against the black background of

space (from scattering) to a smoggy orange-yellow (from extinction), against the pale background of the moon.

Plate 4.5 A quarter-lit rendering of a planet with the cylinder-umbra model. Note the sharp shadow in the atmosphere at the poles. Close inspection of the terminator near the center of the planet reveals the white line caused by the adding-epsilon "solution" to the problem of floating point imprecision in ray tracing. This epsilon, added to the ray after intersecting the cylinder where is nearly coincident with the planet surface, causes the secondary ray to miss the planet sphere and proceed inside the planet, thereupon integrating to the base color of the atmosphere (white).

Plate 4.6 In this rendering, the atmosphere color and extinction coefficient vector vary with angle to the sun. Thus the sky is blue in the areas of full daylight, and fades to red towards the terminator. The umbra model is not needed, as the atmosphere is darker and of lower density on the night side.

Plate 4.7 Here the radial fog or planetary atmosphere model is used for visualization of a DLA (diffusion limited aggregate) cluster. Foreground and background are distinguished for this visually-complex object by the atmospheric effect. Note that there is no light source in this scene; the non-physicality of the atmosphere model makes it "semi-luminous": it appears light, but cannot illuminate other objects. It can, however, be reflected by shiny balls.

Plate 4.8 "Fractal Mandala" illustrates an artistic use of the models described in this paper. The extinction coefficient vector has been used not to simulate Rayleigh scattering, but to simulate a range of blackbody radiator temperatures. The goal was to depict a glowing ball of gas resembling a collapsing (dustless) protostellar nebula.

Plate 4.9 Here we have a dispersing prism on checkerboard, with the empirical rainbow model above (rendered with non-empirical angular width and position parameter values). Close inspection reveals green fringes where the dispersed image of the checkerboard transitions from red to white. This artifact results from the use of red energy to represent violet with magenta.

Plate 4.10 Spectral aliasing in the physical rainbow model, due to the use of a small number of samples and a point light source.

Plate 4.11 Spectral antialiasing, as provided by Jittering and convolution of the rainbow simulation data with a one-dimensional, normalized kernel representing the angular illumination distribution across the Sun's disk.

Plate 4.12 Here we see an ideal rainbow rendered with a black background and no terrain to truncate the bottom; thus the rainbow appears as a circle. Intensity modulation by cloud shadows and non-uniform rain distribution is simulated here with 2-D fBm intensity map, generated with a procedural texture.

Plate 4.13 "Medicine Lake" illustrates Alexander's band clearly: see how the sky appears lighter inside the primary arc, and outside of the secondary. The tree in the foreground is a preliminary L-system model of tropism by Prusinkiewicz. [123]

Plate 5.1 "Bay Fog" employs an fBm texture on a plane to simulate a cloud stratum. The texture actually assigns a fog value to the ray which is refracted (at no change in index of refraction) through the plane. Thus the fog is more dense where the terrain behind is farther away, and it disappears near the plane's intersection with the terrain. The fog also casts shadows on the terrain below. Note also the subtle use of Rayleigh scattering as a scale cue.

Plate 5.2 A planetary-scale model of clouds, with a vector-valued noise used for distortion. Note that the distortion varies smoothly, as the cubic polynomial interpolant of the noise function.

Plate 5.3 The above cloud model, generated with an fBm-valued distortion vector. Here we use a fractal distortion in an attempt to get an approximation of turbulence. Unfortunately, the result looks more like raw cotton than turbulent flow.

Plate 5.4 A procedural texture as a model of a cyclonic storm system. Note that different fractal models are used on the large and small scales, in accordance with observations of Nature. [62] This may be viewed as a first approximation to viscous damping in a model of turbulent flow.

Plate 5.5 A procedural model of the cloud tops of Venus, with the Coriolis effect. This relatively faithful model of Nature was generated by fairly small code segment.

Plate 5.6 A procedural model of the cloud tops of Jupiter, with Io in the foreground and the shadow of Ganymede behind.

Plate 5.7 The Voyager image of Jupiter and Io, after which Plate 5.6 is modelled. Note the vortices in the clouds of Jupiter, a salient aspect of turbulent flow not yet modelled well with fBm-based procedural textures.

Plate 5.8 "Other State" features a procedural model of Saturn and its rings.

Plate 5.9a-f Stages in the construction of a terran procedural texture.

Plate 8.1 A panoramic rendering of the Valles Marineris on Mars, synthesized using elevation data derived by the USGS from Viking imagery. The field of view is 370° by 180°. A procedural texture with adaptive level of detail has been

used to enhance surface detail and the visibility of topographic morphology. The height field was grid-traced using Rayshade, with Phong-shaded triangles used to disguise surface tessellation, particularly in the foreground where the triangles would appear quite large.

Plate 9.1 "Blessed State" is an example of artistic self-expression gained strictly through the access provided by the formal logic of a computer program. This image, as well as all others presented here, was constrained to represent the exact output of a computer program. That program creates a two dimensional image from a three dimensional model. The random fractal elements which grant the image its visual complexity feature parametric control of statistical behavior only; no direct control of details is possible in this approach.

Color Slide Pages

Appendix F. Curriculum Vitae

Acknowledgments

This dissertation is dedicated to my father The Judge, my sister Ruthie, and my fiancée Beth, and to the memory of my mother Ruth and eldest sister Marie.

My greatest burden of gratitude is to Benoit Mandelbrot, for supporting for years, my work at the fringe of his dynamic Physics Group in the Yale math department. Next to David Gelernter, who's idea it originally was for me to pursue my PhD here in the Yale CS department, and who has generously sponsored me throughout my tenure here. Then to Martin Schultz, for his generosity in supporting me in my final year of graduate school -- in of the most unexpected acts of beneficence I've ever witnessed! And my undying gratitude goes to Michelle Boyd, Nancy Marx, and Silicon Graphics Inc. for their generous support, in the form of a six-month loan of an Iris Indigo graphics workstation for use at home. This made possible the artistic refinement of many of these images; if they are beautiful, we have Michelle and her colleagues to thank.

Special thanks are due to Craig Kolb, for all his help in implementing many of the results described here, and with many more which are not mentioned.

I sincerely thank Craig, Ken Perlin, and Bill Gropp for their help in formulating and implementing the radial fog model, and Heath Erskine, Kirby Hawkes, Miguel Garcia, Tom Adams, and Mike Loteazka of the Greater New Haven State Technical College for their help in implementing the physical model of the rainbow. Thanks are also due to Benoit for tolerating the seemingly non-fractal atmospheric research -- of which he cryptically quipped "Well, many things which do not appear to be fractal, are indeed fractal." Thanks also to Mike Gigante of the Royal Melbourne Institute of Technology, for providing a (mostly) peaceful and quiet home away from home at which some of this text was written and revised.

All images were rendered in parallel using C-Linda [16] with modified versions of the Optik [1] and Rayshade [67] ray tracers, on a network of UNIX workstations. This work was funded in part through the Office of Naval Research contract N00014-88-K-0217 and grant N00014-89-J-1671. The parallel systems used are supported in part by NSF grants DCR-8601920 and DCR-86576167, and by the generous support of IBM, DEC, and HP/Apollo.

And finally, the strictures of decorum compel me to offer thanks to the yammering legions of wit-free proofreaders, who cheerfully hammered all but the last vestiges of humor out of these pages (typos, and spare commas, too). I'll see you all in Hades.

References

[1] Amanatides, J., F.K. Musgrave, R. Skinner, L. Slipp, and A. Woo, *Optik Users' Manual, Version 0.9.1*, June, 1987, University of Toronto.

[2] Anderson, D.P., *personal communication*, 1993, Southern Methodist University.

[3] Anderson, M.G., *Modelling Geomorphological Systems*, ed. M.G. Anderson, 1988, New York, John Wiley & Sons.

[4] Arvo, J., *personal communications*, 1992, Cornell University.

[5] Arvo, J. and D. Kirk, *Fast Ray Tracing by Ray Classification*, Computer Graphics, July, 1987, **21**:4, pp. 55-64.

[6] Arvo, J. and D. Kirk, *Particle Transport in Image Synthesis*, Computer Graphics, August 1990, **24**:4, pp. 63-66.

[7] Barr, A.H., *Teleological modeling*, in *Making them move: mechanics, control, and animation of articulated figures*, N.I. Badler, B.A. Barsky, and D. Zeltzer, Editors, 1991, Morgan Kaufmann, pp. 315-321.

[8] Berger, M., T. Trout, and N. Levit, *Ray Tracing Mirages*, IEEE Computer Graphics and Applications, May, 1990, **10**:3, pp. 36-41.

[9] Besancon, R.M., *The Encyclopedia of Physics*, 1974, New York, Van Nostrand Reinhold Company.

[10] Bjornson, R., *Linda on Shared-Memory Multiprocessors*, Doctoral Dissertation, Yale University, 1993.

[11] Blinn, J., *Simulation of Wrinkled Surfaces*, Computer Graphics, August, 1978, **12**:3, pp. 286-292.

[12] Blinn, J.F., *Light Reflection Functions for Simulation of Cloud and Dusty Surfaces*, Computer Graphics, July, 1982, **16**:3, pp. 21-29.

[13] Bolton, E., *personal communications*, 1991, Yale University.

[14] Born, M. and E. Wolf, *Principles of Optics*, 1980, Oxford, Pergamon Press.

[15] Bouville, C., *Bounding Ellipsoids for Ray-Fractal Intersection*, Computer Graphics, July 1985, **19**:3, pp. 45-52.

[16] Carriero, N. and D. Gelerenter, *Linda in Context*, Communications of the ACM, April, 1989, **32**:4, pp. 444-458.

[17] Carriero, N. and D. Gelernter, *How to Write Parallel Programs: A First Course*, 1990, Cambridge, MIT Press.

[18] Chandrasekhar, S., *Radiative Transfer*, 1960, New York, Dover.

[19] Considine, D.M., ed., *Van Nostrand's Scientific Encyclopedia*, ed., ed., 1976, Van Nostrand Reinhold Company, New York.

[20] Constantin, P., I. Procacchia, and K.R. Sreenivasan, *Isoscalar Contours in Turbulence*, Phy. Rev. Lett., 1991, **67,** pp. 1739.

[21] Cook, R.L., *personal communication*, 1992.

[22] Cook, R.L., *Shade Trees*, Computer Graphics, July, 1984, **18**:3, pp. 223-230.

[23] Cook, R.L., *Stochastic Sampling in Computer Graphics*, ACM Transactions on Graphics, January, 1986, **5**:1, pp. 51-72.

[24] Cook, R.L., *et al.*, *"Road to Point Reyes"*, Computer Graphics, July, 1983, **17**:3, title page.

[25] Cook, R.L., T. Porter, and L. Carpenter, *Distributed Ray Tracing*, Computer Graphics, July, 1984, **18**:3, pp. 137-145.

[26] Cook, R.L. and K.E. Torrance, *A Reflectance Model for Computer Graphics*, Computer Graphics, August, 1981, **15**:3, pp. 307-316.

[27] Cornsweet, T.N., *Visual Perception*, 1970, New York, Academic Press.

[28] Dawkins, R., *The Blind Watchmaker*, 1987, New York, W. W. Norton & Co.

[29] Dobkin, D., *personal communication*, 1991, Princeton University.

[30] Fabri, E., G. Fiorio, F. Lazzeri, and P. Violino, *Mirage in the Laboratory*, American Journal of Physics, January 1982, **50**:6, pp. 517-521.

[31] Foley, J.D., A.v. Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice*, 2nd Edition ed., 1990, Reading, Massachusetts, Addison-Wesley.

[32] Fournier, A., *The Modelling of Natural Phenomena*, *Proceedings of the Graphics Interface '89 - Vision Interface '89*, June, 1989, Toronto, Ontario, Canadian Information Processing Society.

[33] Fournier, A., *personal communications*, 1987, UC Santa Cruz.

[34] Fournier, A., D. Fussell, and L. Carpenter, *Computer Rendering of Stochastic Models*, Communications of the ACM, 1982, **25,** pp. 371-384.

[35] Fournier, A. and T. Milligan, *Frame Buffer Algorithms for Stochastic Models*, IEEE Computer Graphics and Applications, 1985, **5**:10.

[36] Fujimoto, A., T. Tanaka, and K. Iwata, *ARTS: Accelerated Ray Tracing System*, IEEE Computer Graphics and Applications, April, 1986, **6**:4, pp. 16-26.

[37] Gardner, G.Y., *SIGGRAPH course notes: Functional Modelling, SIGGRAPH '88*, 1988, Atlanta.

[38] Gardner, G.Y., *Simulation of Natural Scenes Using Textured Quadric Surfaces*, Computer Graphics, July, 1984, **18**:3, pp. 11-20.

[39] Gardner, G.Y., *Visual Simulation of Clouds*, Computer Graphics, July, 1985, **19**:3, pp. 297-303.

[40] Gedzelman, S.D., *Atmospheric Optics in Art*, Applied Optics, August 20, 1991, **30**:24, pp. 3514-3522.

[41] Glassner, A.S., *Space Subdivision for Fast Ray Tracing*, IEEE Computer Graphics and Applications, October, 1984, **4**:10, pp. 15-22.

[42] Greengard, L., *The rapid evaluation of potential fields in particle systems*, 1988, Cambridge, MA, MIT Press.

[43] Greenler, R., *Rainbows, Halos, and Glories*, 1980, Cambridge, Cambridge University Press.

[44] Hall, R.A., *Illumination and Color in Computer Generated Imagery*, 1989, New York, Springer-Verlag.

[45] Hanrahan, P., *personal communications*, 1991, Princeton University.

[46] Hanrahan, P., D. Saltzman, and L. Aupperle, *A Rapid Hierarchical Radiosity Algorithm*, Computer Graphics, July 1991, **25**:4, pp. 197-206.

[47] Hentschel, H.G.E. and I. Procacia, Phys. Rev., 1984, **A29,** pp. 1461.

[48] Hofstadter, D.R., *Gödel, Escher, Bach: an Eternal Golden Braid*, 1979, New York, Vintage Books.

[49] Howat, J.K., *American Paradise - The World of the Hudson River School*, 1987, New York, Metropolitan Museum of Art.

[50] Imhof, E., *Cartographic Relief Presentation*, 1982, New York, Walter de Gruyter.

[51] Kadanoff, L.P., Physics Today, February, 1986, **39,** pp. 6.

[52] Kahaner, D., C. Moler, and S. Nash, *Numerical Methods and Software*, 1989, Engelwood Cliffs, New Jersey, Prentice Hall.

[53] Kajiya, J.T., *Anisotropic Illumination Models*, Computer Graphics, 1985, **19**:3, pp. 15-21.

[54] Kajiya, J.T., *New Techniques for Ray Tracing Procedurally Defined Objects*, ACM Transactions on Graphics, July, 1983, **2**:3, pp. 161-181.

[55] Kajiya, J.T., *New Techniques for Ray Tracing Procedurally Defined Objects*, Computer Graphics, July, 1983, **17**:3.

[56] Kajiya, J.T., *The Rendering Equation*, Computer Graphics, August, 1986, **20**:4, pp. 143-150.

[57] Kajiya, J.T. and B.P.V. Herzen, *Ray Tracing Volume Densities*, Computer Graphics, July, 1984, **18**:3, pp. 165-174.

[58] Kaplan, M., *personal communications*, 1987.

[59] Kay, T.L. and J.T. Kajiya, *Ray Tracing Complex Scenes*, Computer Graphics, August, 1986, **20:4**pp. 269-278.

[60] Kaye, G.W.C. and T.H. Laby, *Tables of Physical and Chemical Constants, 14th Edition*, 1973, London, Longman Group Ltd.

[61] Kelley, A.D., M.C. Malin, and G.M. Nielson, *Terrain Simulation Using a Model of Stream Erosion*, Computer Graphics, August, 1988, **22**:4, pp. 263-268.

[62] Kelley, K.W., *The Home Planet*, 1988, New York, Addison Wesley.

[63] Kerlow, I.V., *Art & Design & Computer Graphics Technology*, Communications of the ACM, July, 1991, **34**:7, pp. 30-31.

[64] Klassen, R.V., *Modelling the Effect of Atmosphere on Light*, ACM Transactions on Graphics, July, 1987, **6**:3, pp. 215-238.

[65] Klassen, R.V., *personal communications*, November, 1992.

[66] Kolb, C. and A. Sherman, *Ray Tracing with Network Linda*, SIAM News, January, 1991.

[67] Kolb, C.E., *Rayshade Users' Guide*, January 1991, Yale University Department of Mathematics.

[68] Kolb, C.E. and F.K. Musgrave, *Procedural Ray Tracing of Height Fields*, submitted to SIGGRAPH 1991.

[69] Kuhlar, E., K. Thyagarajan, and A.K. Ghatak, *A Note on Mirage Formation*, American Journal of Physics, January 1977, **45**:1, pp. 90-93.

[70] Land, E.H., *Six Eyes of Man*, Three-Dimensional Imaging, SPIE, 1977, **120**pp. 43-50.

[71] Lee, M.E., R.A. Redner, and S.P. Uselton, *Statistically Optimized Sampling for Distributed Ray Tracing*, Computer Graphics, July 1985, **19**:3, pp. 61-67.

[72] Lewis, J.P., *Algorithms for Solid Noise Synthesis*, Computer Graphics, July, 1989, **23**:3, pp. 263-270.

[73] Lewis, J.P., *Generalized Stochastic Subdivision*, ACM Transactions on Graphics, July, 1987, **6**:3, pp. 167-190.

[74] Lovejoy, S. and B.B. Mandelbrot, *Fractal Properties of Rain, and a Fractal Model*, Tellus, 1985, **37A**pp. 209-232.

[75] Lynch, D.K., *Step Brightness Changes of Distant Mountain Ridges and Their Perception*, Applied Optics, August 20, 1991, **30**:24, pp. 3508-3513.

[76] Mandelbrot, B.B., *Comment on Computer Rendering of Stochastic Models*, Communications of the ACM, 1982, **25**:8, pp. 581-583.

[77] Mandelbrot, B.B., *The Fractal Geometry of Nature*, 1982, New York, W. H. Freeman and Co.

[78] Mandelbrot, B.B., *Fractal landscapes without creases and with rivers*, in *The Science of Fractal Images*, H.O. Peitgen and D. Saupe, Editor, 1988, Springer-Verlag, New York, pp. 243-260.

[79] Mandelbrot, B.B., *Fractals: Form, Chance, and Dimension*, 1977, San Fransico, W. H. Freeman and Co.

[80] Mandelbrot, B.B., *personal communications*, 1988.

[81] Mandelbrot, B.B. and J.R. Wallis, *Some Long-Run Properties of the Geophysical Records, Water Resources Research 5*, 1969.

[82] Mastin, G.A., P.A. Watterberg, and J.F. Mareda, *Fourier Synthesis of Ocean Waves*, IEEE Computer Graphics and Applications, March, 1987, **7**:3, pp. 16-23.

[83] Max, N., *Atmospheric Illumination and Shadows*, Computer Graphics, August, 1986, **20**:4, pp. 117-124.

[84] Max, N., *Vectorized Procedural Models for Natural Terrain*, Computer Graphics, 1981, **15**:3, pp. 317-324.

[85] Mie, G., *Bietage zur Optik truber Medien Speziell Kolloidaler Metallosungen*, Annalen der Physik, 1908, **25**:3, pp. 377.

[86] Miller, G.S.P., *The Definition and Rendering of Terrain Maps*, Computer Graphics, 1986, **20**:4, pp. 39-48.

[87] Miller, G.S.P., *personal communications*, 1988.

[88] Minnaert, M., *The Nature of Light and Colour in the Open Air*, 1954, New York, Dover.

[89] Mitchell, D.P., *Generating Antialiased Images at Low Sampling Densities*, Computer Graphics, July, 1987, **21**:4, pp. 65-72.

[90] Moravec, H.P., *3D Graphics and Wave Theory*, Computer Graphics, August 1981, **15**:3, pp. 289-296.

[91] Musgrave, F.K., *About the Cover: Natura ex Machina II*, IEEE Computer Graphics and Applications, November, 1990, **10**:6, pp. 5-7.

[92] Musgrave, F.K., *The Art of Fractal Images: a 1993 Calendar*, 1992, New York, Universe Publishing.

[93] Musgrave, F.K., *Formal Logic and Self-Expression*, manuscript, 1993.

[94] Musgrave, F.K., *Grid Tracing: Fast Ray Tracing for Height Fields*, Research Report, July, 1988, Yale University.

[95] Musgrave, F.K., *A Note on Ray Tracing Mirages*, IEEE Computer Graphics and Applications, November, 1990, **10**:6, pp. 10-12.

[96] Musgrave, F.K., *Painting By Numbers*, SunWorld, October, 1991, pp. 56-69.

[97] Musgrave, F.K., *A Panoramic Virtual Screen for Ray Tracing*, in *Graphics Gems III*, D.B. Kirk, Editor, 1992, Academic Press, Boston.

[98] Musgrave, F.K., *Prisms and Rainbows: a Dispersion Model for Computer Graphics, Proceedings of the Graphics Interface '89 - Vision Interface '89*, June, 1989, Toronto, Ontario, Canadian Information Processing Society.

[99] Musgrave, F.K., *Procedural Landscapes and Planets, SIGGRAPH '93 course Procedural Modeling and Rendering Techniques*, 1993, Anaheim, California, ACM SIGGRAPH.

[100] Musgrave, F.K., *Procedural Models of Natural Phenomena, Notes for SIGGRAPH '92 Course 23: Procedural Modelling and Rendering Techniques*, July, 1992.

[101] Musgrave, F.K., *A Realistic Model of Refraction for Computer Graphics*, Masters Thesis, University of California at Santa Cruz, September, 1987.

[102] Musgrave, F.K., *A Realistic Model of Refraction for Computer Graphics, Modelling and Simulation on Microcomputers 1988, conference proceedings*, February,1988, San Diego, Society for Computer Simulation.

[103] Musgrave, F.K., *Some Methods for Aerial Perspective*, submitted to SIGGRAPH '93, 1993.

[104] Musgrave, F.K., *Some Tips for Making Color Hardcopy*, in *Graphics Gems II*, J. Arvo, Editor, 1991, Academic Press, Boston.

[105] Musgrave, F.K., *Uses of Fractional Brownian Motion in Modelling Nature, Fractal Modelling in 3D Computer Graphics and Imaging - SIGGRAPH '91 Course 14*, July, 1991.

[106] Musgrave, F.K., C.E. Kolb, and R.S. Mace, *The Synthesis and Rendering of Eroded Fractal Terrains*, Computer Graphics, July, 1989, **23**:3, pp. 41-50.

[107] Musgrave, F.K. and B.B. Mandelbrot, *The Art of Fractal Landscapes*, IBM Journal of Research and Development, September, 1991, **35**:4, pp. 535-539.

[108] Musgrave, F.K. and B.B. Mandelbrot, *Natura ex Machina (About the Cover)*, IEEE Computer Graphics and Applications, January, 1989, **9**:1, pp. 4-7.

[109] Newcott, W., *Venus Revealed*, Natoinal Geographic, February, 1993, **183**:2, pp. 36.

[110] Nielson, G.M., *Visualization in Scientific and Engineering Computation*, IEEE Computer, September 1991, **24**:9, pp. 58-66.

[111] Nishita, T., Y. Miyawaki, and E. Nakamae, *A Shading Model for Atmospheric Scattering Considering Luminous Intensity Distribution of Light Sources*, Computer Graphics, July, 1987, **21**:4, pp. 303-310.

[112] Nussenzveig, H.M., *The Theory of the Rainbow*, *Scientific American*, April, 1977.

[113] Paeth, A.W., *Digital Cartography for Computer Graphics*, in *Graphics Gems*, A. Glassner, Editor, 1990, Academic Press, Boston, pp. 307-320.

[114] Paglieroni, D.W., *A Unified Distance Transform Algorithm and Architecture,* Machine Vision and Aplications, 1992, **5,** pp. 47-55.

[115] Paglieroni, D.W. and S.M. Peterson, *Parametric Ray Tracing of Height Fields, Proceedings of the Graphics Interface '92,* May, 1992, Toronto, Ontario, Canadian Information Processing Society.

[116] Papoulis, A., *Systems and Transforms with Applications to Optics*, 1969, New York, McGraw-Hill.

[117] Peachey, D., *Procedural Textures, Notes for SIGGRAPH '92 Course 23: Procedural Modelling and Rendering Techniques*, 1992.

[118] Peachey, D.R., *Solid Texturing of Complex Surfaces,* Computer Graphics, 1985, **19**:3, pp. 279-286.

[119] Perlin, K., *An Image Synthesizer,* Computer Graphics, July, 1985, **19**:3, pp. 287-296.

[120] Perlin, K. and E.M. Hoffert, *Hypertexture,* Computer Graphics, July, 1989, **23**:3, pp. 253-262.

[121] Pharr, M., *Compact and Machine-Independent Representations for Height Fields*, manuscript for research report, March 3, 1993, Yale University Dept. of CS.

[122] Pharr, M., *CS490: Special Projects, Mid-Point Report,* CS 490 research writeup, December 12, 1992, Yale University Department of CS.

[123] Prusinkiewicz, P., *The Algorithmic Beauty of Plants*, 1990, New York, Springer-Verlag.

[124] Prusinkiewicz, P., *personal cummincation*, 1992, University of Calgary.

[125] Rogers, D.F., *Procedural Elements for Computer Graphics*, 1985, New York, Mc Graw Hill.

[126] Rosebush, J., *The Proceduralist Manifesto,* Leonardo, 1989, supplemental issue, pp. 55-56.

[127] Rubin, S. and T. Whitted, *A Three Dimensional Representation for Fast Rendering of Complex Scenes,* Computer Graphics, July, 1980, **14**:3, pp. 110-116.

[128] Ruelle, D., Proc. R. Soc. London Ser., 1990, **A 427,** pp. 241.

[129] Ruelle, D., ed., *Chance and Chaos*, 1991, Princeton University Press, Princeton, NJ, 178.

[130] Rushmeier, H.E. and K.E. Torrance, *The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium*, Computer Graphics, July, 1987, **21**:4, pp. 293-302.

[131] Ruskai, M.B., ed., *Wavelets and Their Applications,* ed., ed., 1992, Jones and Bartlett Publishers, Boston.

[132] Russell, B. and A.N. Whitehead, *Principia Mathematica*, 1962, Cambridge University Press.

[133] Saupe, D., *Algorithms for random fractals,* in *The Science of Fractal Images,* H.O. Peitgen and D. Saupe, Editor, 1988, Springer-Verlag, New York, pp. 71-136.

[134] Saupe, D., *Point Evaluation of Multi-Variable Random Fractals,* in *Visualisierung in Mathematik und Naturissenschaft - Bremer Computergraphik Tage 1988,* H. Juergen and D. Saupe, Editor, 1989, Springer-Verlag, Heidelberg.

[135] Schumm, S.A., *Experimental Fluvial Geomorphology*, 1987, New York, John Wiley & Sons.

[136] Shaw, T.M., Phys. Rev. Lett., 1987, **59,** pp. 1671.

[137] Sims, K., *Artificial Evolution for Computer Graphics,* Computer Graphics, July 1991, **25**:4, pp. 319-328.

[138] Sims, K., *Interactive Evolution of Dynamical Systems, Proceedings of the European Conference on Artificial Life,* December 1991, Paris, MIT Press.

[139] Slyusarev, G.G., *Aberration and Optical Design Theory*, 1984, Bristol, Adam Hilger Ltd.

[140] Smith, A.R., *Plants, Fractals, and Formal Languages,* Computer Graphics, July, 1984, **18**:3, pp. 1-10.

[141] Snyder, J.M. and A.H. Barr, *Ray Tracing Complex Models Containing Surface Tessellations,* Computer Graphics, July, 1987, **21**:4, pp. 119-128.

[142] Sommerer, J.C. and E. Ott, *Particles Floating on a Moving Fluid: A Dynamically Comprehensible Physical Fractal,* Science, 1993, **259**:15 January, pp. 335-339.

[143] Southall, J.P.C., *Mirrors, Prisms, and Lenses*, 1933, New York, MacMillan Company.

[144] Sparrow, E.M. and R.D. Cess, *Thermal Radiation and Heat Transfer*, 1978, New York, McGraw-Hill.

[145] Sreenivasan, K.R., R. Ramshankar, and C. Menveau, Proc. R. Soc. London Ser., 1989, **A 421,** pp. 79.

[146] Stam, J. and E. Fiume, *A Multiple-Scale Stochastic Modelling Primitive, Proceedings of Graphics Interface '91,* June, 1991, Morgan-Kaufmann.

[147] Stone, M.C., W.B. Cowan, and J.C. Beatty, *Color Gamut Mapping and the Printing of Digital Color Images,* ACM Transactions on Graphics, October, 1988, **7**:4, pp. 249-292.

[148] Strutt, J.W. (Lord Rayleigh), *On the light from the sky, its polarization and colour, Philos. Mag. 41 (April 1871),* 1964, New York, Reprinted in Lord Rayleigh, Scientific Papers I, Dover.

[149] Sweeny, M.A.J., *The Waterloo CGL Ray Tracing Package,* Masters Thesis, University of Waterloo, 1984.

[150] Thomas, S.W., *Dispersive Refraction in Ray Tracing,* Visual Computer, January, 1986, **2**:1, pp. 3-8.

[151] Todd, S. and W. Latham, *Evolutionary Art and Computers*, 1992, London, Academic Press.

[152] Voss, R., *Fractals in nature: from characerization to simulation,* in *The Science of Fractal Images,* H.O. Peitgen and D. Saupe, Editor, 1988, Springer-Verlag, New York, pp. 21-70.

[153] Voss, R.F., *The Art of Fractal Images: a 1992 Calendar*, 1991, New York, Universe Publishing.

[154] Voss, R.F., *Fourier Synthesis of Gaussian Fractals: 1/f Noises, Landscapes, and Flakes, SIGGRAPH '83 course notes: State of the Art in Image Synthesis,* 1983.

[155] Voss, R.F., *Random Fractal Forgeries,* in *Fundamental Algorithms for Computer Graphics,* R.A. Earnshaw, Editor, 1985, Springer-Verlag, Berlin.

[156] Weghorst, H., G. Hooper, and D. Greenberg, *Improved Computational Methods for Ray Tracing,* ACM Transactions on Graphics, January, 1984, **3:1,** pp. 52-69.

[157] Whitted, T., *caption for image on back cover,* Computer Graphics, August, 1981, **15**:3.

[158] Whitted, T., *An Improved Illumination Model for Shaded Display*, CACM, June, 1980, **23**:6, pp. 343-349.

[159] Whitted, T., *personal communications,* 1987, UNC Chapel Hill.

[160] van Wijk, J.J., *Spot Noise: Texture Synthesis for Data Visualization*, Computer Graphics, 1991, **25**:4, pp. 309-318.

[161] Wood, R.W., *Physical Optics*, 1911, New York, MacMillan Company.

[162] Worley, S.P., *personal communication*, 1993.

[163] Wyszecki, G. and W.S. Stiles, *Color Science: Concepts and Methods, Quantitative Methods and Formulas*, 1967, New York, Wiley-Interscience.

[164] Yaeger, L., C. Upson, and R. Meyers, *Combining Physical and Visual Simulation - Creation of the Planet Jupiter for the Film "2010",* Computer Graphics, August, 1986, **20**:4, pp. 85-93.