A VISUALIZATION FRAMEWORK FOR LARGE-SCALE VIRTUAL ASTRONOMY

Chi-Wing Fu

Submitted to the faculty of the Graduate School
in partial fulfillment of the requirements
for the degree
Doctor of Philosophy
in the Department of Computer Science
Indiana University

December 2003
Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements of the degree of Doctor of Philosophy.

Andrew J. Hanson
(Principal Advisor)

Randall Bramley

Florin Cutu

Daniel P. Maki
"When I consider thy heavens,
the work of thy fingers,
the moon and the stars,
which thou hast ordained;
What is man,
that thou art mindful of him?
and the son of man,
that thou visitest him?"

Psalm 8:3-4
Acknowledgments

I would like to express my gratitude to my supervisor, Prof. Andrew J. Hanson, for his endless support in the last four and a half years. Without his supervision and guidance, I recognize that I could hardly have accomplished this research work and finished this dissertation. I would also like to thank the members of my doctoral committee, Prof. Randall Bramley, Prof. Florin Cutu, and Prof. Daniel P. Maki, for their assistance in this dissertation and for the valuable material I learned from their lectures. In addition, a special thanks goes to Stuart Levy at NCSA for his precious advice on star rendering when I just began my work on this area.

I would also like to acknowledge my friends in the Computer Science department, Mitja, Sriram, Eric Wernert, John Huffman, Ken Chiu, John Springer, for their support and help during my time at Indiana University. In addition, I would like to thank the administration staff, Lucy and Sherry, of our Computer Science department for their assistance in the paperwork, roster, and registration. And I would like to thank my previous supervisor Prof. Pheng-Ann Heng and my friend Prof. Tien-Tsin Wong for their support, especially...
when I was working as a research assistant in Hong Kong during the summers.

Besides, I must thank my parents, my girlfriend Yin-Ling Cheung, and my friends in Hong Kong for their care and patience, regardless the fact that I was always away from home, and also for their continuous prayers and support to strengthen me so as to finish this study. Although they are at a long distance from me, their care and support are real and true.

Finally, thanks to God, the creator of the real Universe, which this research work is trying to study and visualize.
Abstract

Motivated by advances in modern positional astronomy, this research attempts to digitally model the entire Universe through computer graphics technology. Our first challenge is space itself. The gigantic size of the Universe makes it impossible to put everything into a typical graphics system at its own scale. The graphics rendering process can easily fail because of limited computational precision. The second challenge is that the enormous amount of data could slow down the graphics; we need clever techniques to speed up the rendering. Third, since the Universe is dominated by empty space, objects are widely separated; this makes navigation difficult.

We attempt to tackle these problems through various techniques designed to extend and optimize the conventional graphics framework, including the following: power homogeneous coordinates for large-scale spatial representations, generalized large-scale spatial transformations, and rendering acceleration via environment caching and object disappearance criteria. Moreover, we implemented an assortment of techniques for modeling and rendering a variety of astronomical bodies, ranging from the Earth up to faraway galaxies.
and attempted to visualize cosmological time; a method we call the Lightcone representation was introduced to visualize the whole space-time of the Universe at a single glance. In addition, several navigation models were developed to handle the large-scale navigation problem. Our final results include a collection of visualization tools, two educational animations appropriate for planetarium audiences, and state-of-the-art-advancing rendering techniques that can be transferred to practice in digital planetarium systems.
Contents

Acknowledgments ........................................... v

Abstract ............................................. vii

1 Introduction .......................................... 1
   1.1 Background ......................................... 1
   1.2 The Development of Positional Astronomy ............... 5
   1.3 Positional Astronomy meets Computer Graphics ............. 8
   1.4 Related Work ........................................ 10
   1.5 Problems to be Addressed in this Research Work ............ 15
   1.6 Goals of this Research Work .......................... 16
   1.7 Organization of this Thesis ........................... 18

I A Framework for Large-Scale Visualization ............... 21

2 Existing Graphics Pipeline ............................ 23
   2.1 Problems in Existing Graphics Hardware .................... 23
2.2 Transformation Unit in the Fixed T & L Pipeline .................. 25
   Modelview Transformation ........................................... 26
   Projection Transformation ........................................... 28
   Perspective Division ................................................... 30
   Examples: Perspective and Parallel Projections .................. 31
2.3 Analysis: Transformation Unit ....................................... 35
   Computing $n$ and $f$ from the Projection Matrix .................. 35
   Distinguishing Perspective or Parallel Projection from values in the Projection Matrix ........................................... 36
   Where are the Eye point and $z = \pm \infty$? ....................... 37
   Conversion between $z_{\text{eye}}$ and $z_{\text{NDC}}$ ..................... 38
   Utilization of the Depth Buffer in Perspective Projection .......... 39
2.4 Programmable Graphics: Shaders and Cg ............................. 43
   Vertex Program (or Vertex Shader) .................................. 43
   Mimicking Vertex Programs on Older Graphics Hardware .......... 45
   Pixel/Fragment Shading and Cg: C for Graphics ........................ 45

3 Power Homogeneous Coordinate (PHC) ................................. 47
   3.1 Definition .......................................................... 47
   3.2 Analysis ............................................................ 49
       Conversions ......................................................... 49
       Linear VS Exponential .............................................. 51
3.3 Arithmetic for Power Homogeneous Coordinates ................................. 52

4 PHC Transformation: the Method .................................................. 58

4.1 Introduction .................................................................................... 58

4.2 Using PHC in Transformation Unit .................................................. 59

4.3 PHC Modelview Transformation .................................................... 60

The Model: Embedded Exponential Scale of the Camera ......................... 60

Mathematics for PHC Modelview transformation .................................... 62

Effects of $s_{mv}$ ................................................................................. 63

4.4 PHC Projection Transformation ..................................................... 65

The Idea: Depth Rescaling Method (DRM) .......................................... 65

The Model: Depth Rescaling Model ..................................................... 67

Mathematics for PHC Projection transformation .................................... 70

5 PHC Transformation: Implementation and Demonstration .................. 83

5.1 Introduction ..................................................................................... 83

5.2 Implementation: The PHC API ..................................................... 84

Description of the PHC API ................................................................ 84

Assumptions in this Implementation .................................................. 94

With/Without Vertex Programming .................................................. 95

5.3 Demonstration Programs .............................................................. 100
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coordinate systems</td>
<td>178</td>
</tr>
<tr>
<td>9.2</td>
<td>Basic Physics and Mathematics about stars</td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>Parallax, the Parsec, and Distance</td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>Brightness of stars: luminosity, apparent magnitude, and absolute magnitude</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>Proper motion, Radial velocity, and Space velocity</td>
<td>183</td>
</tr>
<tr>
<td></td>
<td>Local standard of Rest (LSR)</td>
<td>185</td>
</tr>
<tr>
<td></td>
<td>Temperature and Color of stars</td>
<td>186</td>
</tr>
<tr>
<td>9.3</td>
<td>Stellar Database Structures</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>Introducing the Tree structures</td>
<td>190</td>
</tr>
<tr>
<td></td>
<td>Customizing the Trees for Star Rendering</td>
<td>195</td>
</tr>
<tr>
<td></td>
<td>Pruning Rules for Rendering</td>
<td>197</td>
</tr>
<tr>
<td>9.4</td>
<td>Photorealistic Rendering of stars</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Measuring Star Size against Apparent Magnitude ( (V_{mag} \rightarrow \text{pixel} - \text{unit}) )</td>
<td>202</td>
</tr>
<tr>
<td></td>
<td>Star Resizing Method</td>
<td>206</td>
</tr>
<tr>
<td></td>
<td>Star Modeling Method</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>Star Coloring Method</td>
<td>208</td>
</tr>
<tr>
<td></td>
<td>Star Rescaling Method</td>
<td>209</td>
</tr>
<tr>
<td></td>
<td>Brightness Adjustment</td>
<td>210</td>
</tr>
<tr>
<td></td>
<td>Performance Issues</td>
<td>211</td>
</tr>
<tr>
<td>9.5</td>
<td>Rendering Results</td>
<td>212</td>
</tr>
<tr>
<td>9.6</td>
<td>Conclusion and Future Work</td>
<td>224</td>
</tr>
</tbody>
</table>
# 10 Modeling and Rendering the Milky Way Galaxy

10.1 Background .............................................. 225

Datasets ....................................................... 226

Coordinate System ........................................... 227

Physics and Mathematics .................................... 228

10.2 The Milky Way at Solar system scale: Panoramic images .............................. 233

10.3 The Milky Way at the stellar scale: No Hipparcos stars ................................. 236

Labeling star pixels ......................................... 238

Filling star pixels using Texture Synthesis ..................... 240

The Controversy .................................................. 242

Color Analysis ................................................... 242

10.4 Representing the Milky Way in Three-dimensions ........................................ 244

The Goddard Model .............................................. 244

The Particle Model ............................................ 245

10.5 Conclusion and Future work ............................................ 249

# 11 Modeling and Rendering Galaxies

11.1 About Galaxies .............................................. 250

Datasets .......................................................... 251

Coordinate Systems ............................................. 252

Basic Knowledge about Galaxies ................................. 253

11.2 Modeling and Rendering at the Scale of Galaxies ........................................... 261
Modeling and Rendering Galaxies ........................................ 261
Cosmic Background Radiation (CMBR and WMAP) ............. 262
11.3 Rendering Results ...................................................... 264
11.4 Conclusion and Future work ....................................... 269

III Visualizing Cosmological Time ...................................... 270

12 The Cosmic Clock – Lightcone ...................................... 272
12.1 Time in Cosmology .................................................... 272
Background ............................................................... 272
The Universe is Expanding .............................................. 273
12.2 The Making of the Lightcone ...................................... 276
The Basic form ............................................................ 276
The Warp form ............................................................ 278
The $a(t)$-rescaled form ............................................... 279
12.3 Visualization Results with Lightcone ......................... 281
12.4 Conclusion ............................................................ 285

13 Time Management in the Universe ............................... 286
13.1 Introduction .......................................................... 286
Challenges of Large-scale Dynamic Astrophysics Visualization . 287
13.2 Managing Time in the Virtual Environment .................. 287
13.3 Motion Visualization Methods ...................................................... 290
Screen-Time Objects ........................................................................... 290
Slow Objects ....................................................................................... 291
Fast Objects ....................................................................................... 292
13.4 The Interaction Graph ................................................................. 295
13.5 Conclusion .................................................................................. 298

IV Large-scale Navigation 299

14 Astronomical Navigation ................................................................. 301
14.1 Introduction .................................................................................. 301
14.2 Characteristics of Astronomical Navigation ................................... 302
14.3 Navigation Models ....................................................................... 307
Overview: Navigation Models .............................................................. 307
The Unaided Navigation Model ............................................................ 307
The Spatial-scaling Navigation Model .................................................... 310
The Path-Based Navigation Model ....................................................... 312
The Constrained Navigation Model (Manifold-based) ......................... 314
14.4 Rendering Issues ......................................................................... 319
Cycling Mechanism: Environment Caching and Object Disappearance ... 319

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Powercubes Visualization  .................................................. 321
Minimum Spanning Tree Visualization  ............................. 323

14.5 Conclusion and Future work  ........................................ 325

V Results, Conclusion, and Future work  327

15 Results  329

15.1 Introduction  ............................................................. 329
15.2 Visualization Tools  .................................................. 329
Overview  ................................................................. 329
Tool: Earthday  ......................................................... 330
Tool: starview  ......................................................... 332
Tool: galview  .......................................................... 333
Tool: specview  ........................................................ 336
Tool: skyalign  ......................................................... 336
Tool: distEdTool ....................................................... 338
Tool: power10time ..................................................... 340
15.3 Animations  ........................................................... 343
Overview  ................................................................. 343
Animation: Cosmic Clock ............................................... 343
Animation: Solar Journey ............................................... 346
15.4 Conclusion  ........................................................... 346
16 Conclusion and Future work

16.1 Conclusion .......................................................... 350
16.2 Development Highlights ........................................ 352
16.3 Future work ......................................................... 355

A Astronomy-Related .................................................. 356
A.1 Units ....................................................................... 356
A.2 Basic Statistics ....................................................... 357
A.3 Datasets for Virtual Astronomy ................................. 369
A.4 Coordinate Systems ................................................ 387
A.5 Time and Calendar ................................................ 396
A.6 Powers of Ten Tables .............................................. 401

B Vertex Programs for PHC Transformation ..................... 404

B.1 Vertex Program Overview ........................................ 404
B.2 Vertex Program Example: (vp.nicest.cutoff.vp)
   (Nicest + Cutoff + Perspective Projection) .................. 406
B.3 Vertex Program Example: (vp.nicest.nocutoff.parallel.vp)
   (Fastest + NoCutoff + Parallel Projection) .................. 416
## List of Tables

1.1 Astronomy Animations ........................................... 11

3.1 Powers of 10 – objects at different scale .................. 48

3.2 Differences between PHC and HC .............................. 52

5.1 Average Frame Rates for the PHC/Depth Rescaling Library rendered with/without Vertex Programs (in frames per second) ........................................... 109

7.1 Related terms in the Two Criteria .............................. 136

9.1 The Hipparcos Catalogue (apparent magnitude - in Johnson V units) ........ 176

9.2 Algorithm for Finding a Vantage Point ...................... 191

A.1 Units for Distances (www.unc.edu/~rowlett/units) ........ 356

A.2 Units for Wavelength, Frequency, Energy, and Temperature .......................... 356

A.3 Units for Angles .............................................. 357

A.4 Statistics of the Sun from [1, 102] ............................ 358

A.5 Statistics of the Moon from [1] ............................... 358


xx
A.7 Statistics of selected asteroids from [1, 102] ........................................... 360
A.8 Statistics of some periodic comets from [2] ........................................... 360
A.10 Table for Greek Alphabet (both lowercase and uppercase) ............ 361
A.11 Constellations in the sky: Note that constellations refer to a region in
the sky, not the lines connecting stars ............................................ 362
A.12 The 26 brightest stars in the sky (data from the Hipparcos Catalogue). 364
A.13 Statistics of the Milky Way Galaxy from [102] .............................. 364
A.14 List of all objects in the Messier catalog (asterisks(*) mark galaxies). 365
A.15 Members of the Local Group of Galaxies from [102] ..................... 367
A.16 Brightest galaxies derived from Brent Tully's galaxy catalog (sorted by the
apparent magnitude) .................................................................... 368
A.17 Powers of $10$ – objects at different scale ....................................... 402
A.18 Powers of $10$ – phenomena at different scale ................................. 403
List of Figures

1.1 Simulation: Night sky around the region of Scorpius. ......................... 2
1.2 Ancient Universe model with stars and zodiac fixed on the outermost shell. 4
1.3 Simulation of three-dimensional planets, stars, and galaxies resulting from Positional Astronomy. .................................. 7
1.4 Scientific Process of Astronomy. ....................................................... 9
1.5 Organization of this Thesis. ............................................................... 19

2.1 Transformation unit: coordinate transformation sequence. .................... 25
2.2 Modelview Transformation. ............................................................... 29
2.3 Eye → Clip → NDC (without shearing). Positive z is to the left, Positive y points upward, and white wireframes indicates the viewing frusta. ........... 32
2.4 Eye → Clip → NDC (with shearing). Positive z is to the left, positive y points upward, and white wireframes indicates the viewing frusta. .......... 34
2.5 The Utilization of the Depth Buffer. ................................................... 41
2.6 OpenGL Pipeline: (1) Transform & Lighting (T&L) Unit in Conventional OpenGL (1.2 or before) and (2) Shader Enabled OpenGL (1.3 or after). . 44
3.1 Expressing the three-dimensional point (1, 0, 0) as homogeneous coordinates (left) and power homogeneous coordinates (right): plot $w$ against $x$ and plot $s$ against $x$ .................................................. 51

4.1 PHC Modelview and Projection Transformations. .................. 59
4.2 Effect of varying $s_{mv}$ results in Powers-of-ten transition. .......... 64
4.3 Depth Rescaling: By distorting vertices along their lines of sight uniformly, we can bring far-away objects into the viewing frustum. .......... 66
4.4 The Depth Rescaling Model. ........................................... 67
4.5 The PHC Projection Transformation. ................................. 71
4.6 Depth Rescaling (Perspective Projection). .......................... 76
4.7 Depth Rescaling (Parallel Projection). ............................... 78
4.8 Mapping $s'_{eye}$ onto $z_{NDC}$. ................................. 79

5.1 Interface of the Demonstration Program (Pages 1 and 2). ........... 101
5.2 Examples showing Depth Buffer errors for bare OpenGL at large scales. 103
5.3 Correcting the Depth Buffer errors using the depth rescaling technique. 104
5.4 Going from $10^{78}$ down to $10^{-80}$. .............................. 105
5.5 The use of the Linear Cutoff Region. .................................. 106
5.6 Rendering with different projections. .................................. 107

6.1 Looking at 61 Cygnus at different times of the year. ............... 113
6.2 Parallax of a star as observed from two different camera locations. .... 114
6.3 Maximum Parallax is attained at a given $s$. ....................... 115
8.6 The Longitude/Latitude grid (markers and grid: yellow, Equator: red, Tropics of Cancer (23.5° N) and Capricorn (23.5° S): blue), the Country/TimeZone boundaries, and the International Date line on Earth model.

8.7 Different types of satellite have different colors.

8.8 Display mode for satellites: points, boxes, frames, and 3D-models.

8.9 APOD 14th July 2003: Satellites over the Sky (GMT June 24th 2003 2:21pm) by Andrew J. Hanson, Philip Chi-Wing Fu, and Priscilla Frisch.

8.10 Rendering the Sun using three Gaussian textures.


8.14 Retrograde motions exhibited by Mercury, Jupiter, and Pluto as seen from Earth.

8.15 Rendering the Heliosphere: the Bow Shock (blue), the Heliopause (green), and the Termination Shock (yellow).

8.16 The Sun's Heliosphere and Heliopause. Model by Timur Linde, Credit & Copyright: P. C. Frisch (University of Chicago) et al., Indiana University.

9.1 Geometry for measuring Parallax.

9.2 Transverse velocity (Proper motion) + Radial velocity = Space velocity.

9.3 Peculiar motion of stars.

9.4 Partitioning Mechanism of the Vantage-Point Tree.

9.5 Computing $\alpha_{\text{cluster}}$: visual angle of a star cluster.
9.6 Note that $\alpha_{\text{view}}$ is the angle subtended by the diagonal of the image plane at the viewing position. .......................... 199

9.7 We can prune the star cluster when the sum of the two visual angles, $\alpha_{\text{cluster}}$ and $\alpha_{\text{view}}$, is smaller than $2\theta$. .............................. 199

9.8 Perform pruning from outer to inner branches of a k-ary VP-tree: Both pruning rules are applicable. .............................................. 201

9.9 Star registration interface. ...................................................... 203

9.10 Locating stars on the Orion photograph. ................................. 204

9.11 Plotting extracted star sub-images: apparent magnitude against B-V. ............... 205

9.12 Colormap: from $\log_{10}(\text{meter})$ distance to star color (the further away from the Earth, the larger the red shift and thus, stars (or galaxies) become redder in color). ............................................. 209

9.13 Rendering stars as textured polygons (left) or anti-aliased points (right). .... 213

9.14 Adjustment of star brightness: the constellation Leo. .............................. 214

9.15 Increasing apparent the magnitude limit: progressively adding more stars to the simulated night sky, looking at Taurus (left) and Pleiades (right). .... 215

9.16 Various Constellations: name (yellow), boundary (green), and connecting lines (yellow). ......................................................... 216

9.17 Navigating at the stellar scale – moving towards Pleiades: notice the changes in star size as we move closer to the stars. ................................. 217

9.18 Powers-of-10 transition from the Solar system scale to the stellar scale. .... 218

9.19 Visualizing positional errors, $\Delta d$ (see Subsection 9.2): Red bars denote the error, which increases for stars further away from Earth. The cubes mark the spatial scale. ......................................................... 219

xxvi

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
9.20 Visualizing space velocity of stars (trail length is set to 20,000 years).

9.21 Displaying orbits of extra-solar planets as rings. Note that we have to scale the orbit length by 50,000 to make them visible. We zoom in to two stars with extra-solar planets in the middle row. Also, the sphere shown in the last row has size about 50 light years, which is about the time since we first sent out radio waves from the Earth.

9.22 Rendering stars in the presence of other data sets at various scales. Note that the Heliopause is the shock wave resulting from the Solar wind and Loop I is a gas cloud produced by a supernova explosion in the Scorpius association.

9.23 Photorealistic rendering – comparing photographs and our simulations: Orion (top), Big Dipper (middle), and Canis Major (bottom). Note that the Hipparcos Catalogue does not have all the stars we can see in the photographs ((c) AAO/ROE, photographs courtesy David Malin).

10.1 Image Coordinates \((u,v)\) and Galactic Coordinates \((l,b)\).

10.2 The Aitoff transformation: From all-sky to a uniform-area elliptical shape.

10.3 The differential rotation of galaxies: rotation curve.

10.4 The rotation curve of our Milky Way Galaxy.

10.5 All-sky multi-wavelength panoramic images.

10.6 A spherical environment map encloses the whole sky around the viewpoint.

10.7 The spectral slider.

10.8 Doubled stars occur when we move to the stellar scale if we do not remove 2D Hipparcos stars in the Mellinger photograph.
10.9 Removing Hipparcos stars (apparent magnitude < 7) from the Mellinger Image [119].
10.10 Textures having statistical patterns: knitwear, marble, and wood.
10.11 Doubled stars are eliminated when we move to the stellar scale after removing Hipparcos stars in the Mellinger photograph.
10.12 The Milky Way VRML Model from the Goddard Space Flight Center at NASA.
10.13 Rendering examples of Particle systems from www.cs.unc.edu/~davemc/Particle.
10.14 Photographs of some galaxies ((c) AAO/ROE, the last two photographs courtesy of David Malin).
10.15 Renderings of the 3D Milky Way Galaxy using our particle system.
11.1 The Hubble relation: \( V = Hd \) (by the Hubble Space Telescope Key Project).
11.2 Positioning the galaxy plane: major and minor axes.
11.3 Tilting the galaxy plane.
11.4 The Hubble sequence: Elliptical \( \rightarrow \) Lenticular \( \rightarrow \) Spiral \( \rightarrow \) Irregulars.
11.5 De Vaucouleurs classification scheme (the central \( SABb(\tau s) \) type is thought to be very close to the shape of our Milky Way) (from [173]).
11.6 The zone of avoidance.
11.7 The original COBE Cosmic Microwave Background Radiation Anisotropy (CMBR) [15, 13] (top) and the more recent Wilkinson Microwave Anisotropy Probe (WMAP) [14] (bottom).
11.8 The interface of “galview.”
11.9 Representing galaxies as points (left column), as axial frames (central column), and as textured polygons (right column), with 10X exaggeration for visibility. ........................................ 266

11.10 Powers-of-ten transitions starting from the Milky Way Galaxy, left-right, top-bottom order with increasing scale. ........................................ 267

11.11 Visualizing the zone of avoidance. Galaxy size exaggerated by 10X (top) and 100X (bottom) for visibility. ........................................ 268

12.1 The expansion of the Universe given in terms of its effective radius $a(t)$ as a function of time $t$, in units of the age of the Universe ................ 275

12.2 Constructing the Lightcone (Earth at the tip of the cone). ................ 277

12.3 Warping data points into the Lightcone. ........................................ 278

12.4 Animation sequence: warping the Universe’s entire data volume into the Lightcone. ........................................ 280

12.5 Animation sequence (from left to right): rescaling the Lightcone according to Physical Coordinates using the effective radius, $a(t)$ from Figure 12.1 281

12.6 The Lightcone as a spatial slider – we can slice through the cone and look at light in the night sky originating in different time ranges. ................. 282

12.7 Navigation with comoving coordinates. ........................................... 283

12.8 Navigation with physical coordinates, using the comoving coordinate scales in Figure 12.7 for absolute reference. ........................................ 284

13.1 The Earth with time-streaked satellite animations and static stars and galaxies. 291
13.2 An object that is too slow to show motion in the current scale can be annotated to show how many powers separate its motion from the current screen time scale.

13.3 Motion-blurred trails of objects moving, orbiting, or rotating too fast for animation in current screen time.

13.4 "Too fast" representations at (a) roughly 300 years per screen second and (b) roughly 30 years per screen second. (c) "Too slow" representations for the motion of planets in our Solar system at a scale of approximately 10 days per screen second.

13.5 Motion Blur Representation: (a) Normal texture, (b) Texture blur for one rotation in two screen seconds, (c) Texture blur for one rotation in less than one screen second.

13.6 The Interaction Graph for the Universe dataset collection.

14.1 Most datasets are Earth-centered in nature.

14.2 Controlling Field-of-view linearly and logarithmically in different ranges.

14.3 The Cortona VRML browser showing the Heliopause in the stellar space.

14.4 Global centers along the Interaction Graph.

14.5 Constructing a camera path.

14.6 Constructing the Navigation/Animation Path using the "moviePET."

14.7 The Constrained Navigation Model.

14.8 Images corresponding to three viewpoints on the logarithmically scaled "wedge" path from the Earth to the Moon.

xxx
14.9 Images corresponding to three viewpoints on the adaptive twisted-orientation constraint manifold encompassing the Earth, the Solar system, and the Milky Way galaxy. ............................ 318

14.10 Visualizing spatial context using Powercubes. ................................. 322

14.11 Visualizing spatial context among stars using a Minimum Spanning Tree. ................................. 324

15.1 Screenshots of earthday. .............................................................. 331

15.2 Screenshots of starview. .............................................................. 334

15.3 Screenshots of galview. .............................................................. 335

15.4 Screenshots of specview. .............................................................. 337

15.5 Screenshots of skyalign. .............................................................. 339

15.6 Screenshots of DistEdTool (Distance Editing Tool). ............................. 341

15.7 Screenshots of power10time. ....................................................... 342

15.8 Annotation of the Cosmic Clock animation in the Electronic Theater Proceedings (top) and cover of the Proceedings (bottom) (Siggraph 2000). ............................. 344

15.9 Snapshots taken from Cosmic Clock. .............................................. 345

15.10 Snapshots taken from Solar Journey (Note: since this computer animation is to be shown in Beijing, we have used Chinese characters for the annotations). .............................................................. 347

16.1 Large-scale visualization framework with “power homogeneous coordinates” as the basis. .............................................................. 351

A.1 The Terrestrial Coordinate system: Longitude and Latitude. ...................... 387

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
A.2 The Equatorial Coordinate system: Right Ascension (RA) and Declination (DEC) .................................................. 388
A.3 The Ecliptic Coordinate system ............................................ 390
A.4 The Galactic Coordinate system: Galactic Longitude (l) and Galactic Latitude (b) ............................................................ 391
A.5 The Super-Galactic Coordinate system (non-stippled axes) with respect to the Galactic Coordinate system (stippled axes) ............... 392
A.6 The Sidereal day is slightly shorter than the Solar day ............... 396
Introduction

The night sky is a piece of art, hanging high above the Earth for billions of years long before our own recorded history. Through its beauty and mystery, it attracted our ancestors to understand and explore it through the ages. Figure 1.1 shows our simulation of the night sky in the region of the constellation Scorpius, lying between us and the Center of the Milky Way.

1.1 Background

Though the earliest interpretations of phenomena in the sky took the form of myths, our exploration of the Universe began at that moment when our ancestors looked at the sky and made up ideas about it. Legends have been passed down to us from both ancient western and ancient Chinese worlds. For instance, the ancient Chinese thought that Solar eclipses happened when the "sky dog" ate the Sun [39]; therefore, they had to make great noise to scare the sky dog away in order to free the Sun. On the other side of the world [24], ancient Indians believed that the Sun was eaten by an invisible dragon and they had to immerse themselves in water so as to help the Sun and Moon defending against the dragon. In ancient Japan, it was a custom to cover wells during the eclipse in order to prevent poison
I. Introduction

Figure 1.1: Simulation: Night sky around the region of Scorpius.

from dropping into the water. Although these stories sound ridiculous to modern people like us, these myths mark our very first step of exploring the Universe. This very first step is observation.

Sensing that interesting astronomical phenomena happened on a regular basis, our ancestors began to make records of them using text and pictures. Gradually, certain repetitive patterns were observed within the long period of records. For example, the Egyptians were able to observe and record the heliacal rising of Sirius, called the Dog Star in ancient Egypt, which is the brightest star in the sky and belongs to the constellation Canis Major; they then made calendar based on its position and predicted flooding in the River Nile accordingly [135]. On the other hand, based on the long records from 1,600 B.C. to 800 B.C., the Babylonians were able to roughly predict planetary motions and eclipses [182]. Historical records belonging to the Babylonians were inscribed on clay tablets, discovered
by modern archaeologists. Observations were important to ancient Chinese astronomers as well. Motivated by agricultural needs, as far back as 194 B.C., during the time of emperor "han wu di" (漢武帝), ancient Chinese created the calendar, namely "tai chu li" (太初暦), based on the motions of both the Sun and the Moon. This accounted for leap years and months in a systematic way [39]. In addition, ancient Chinese were good at making accurate records of astronomical phenomena. For example, 900 Solar eclipses and 600 lunar eclipses have been documented over the past 2,600 years [39, 65] and most of them have been proved to be fairly accurate based on current technology. Similar to the Babylonians, eclipse predictions were made based on the previous records.

The process of doing Astronomy does not end with observations nor the making of records; the next step is to create a model to qualify the observations and the records. This step has already taken more than two thousand years in our history and we were sometimes trapped by previous mistakes or false interpretations of the observations. As mentioned in the previous paragraph, the Babylonians were very successful in making rough predictions of planetary motions and even eclipses by using their long observational records. However, perhaps they were satisfied with the predictions they made for their agricultural or astrological needs, since they did not have or seek a geometrical nor a physical model behind their predictions.

Inheriting from the Egyptian and the Middle-Eastern civilizations, the Greeks were able to make this break-through in astronomy and create some simple models of the Universe based on their experience in mathematics and geometry. One of the first models was devised by Aristotle around 350 B.C.; it is a model refined from Anaximenes (585-526 B.C.), Pythagoras (582-500 B.C.), and Eudoxus (408-355 B.C.). Basically, Aristotle's model is a geocentric model with the Earth being the center of the Universe. Planets, including the Sun and the moon, rotate around the Earth over a layer of spherical shells [135, 25]. And at the end, stars and zodiacs were fixed on the two outermost shells believed to be just outside
the Saturn shell. It is worth noting that since we are limited by the resolution power of our naked eyes, only five planets (Mercury, Venus, Mars, Jupiter, and Saturn) were known in ancient times. Later on, in 140 A.D., Claudius Ptolemy in Alexandria carefully quantified this geocentric model mathematically by characterizing planetary motions precisely with epicycles and later published the famous book “Almagest of Ptolemy” [141]. After that, for almost 1,400 years, Ptolemy’s model was widely used for predicting planetary motions until Nicholas Copernicus of Poland (1473–1543) proposed his heliocentric model in his book “On the Revolutions of the Celestial Spheres” during 1543 [33]. Using Tycho’s (1546-1601) observational data, Kepler (1571-1630) further refined Copernicus’ heliocentric model and summarized Kepler’s three famous laws experimentally. He then published his work in the book, “Rudolphine Tables” [104], in 1627.

Figure 1.2: Ancient Universe model with stars and zodiac fixed on the outermost shell.
Independently in the Far East, models called "gai tian shuo" (蓋天說) \[6\] (~100 B.C.) and "hun tian shuo" (混天說) \[152\] (~117 A.D.) were developed in ancient China. The former theory suggested that the sky was a big hemisphere covering the whole Earth, which was thought to be a square plane. The latter theory was similar to Aristotle’s model; it proposed a celestial sphere and studied the motions of the planets, the Sun, and the moon across the celestial sphere. Interestingly, there was one more famous theory developed in China, namely "xuan ye shuo" (旋衍說) \[7\] (~295 A.D.), which asserted that the Universe is infinite and all planets and stars are floating in a big empty space. However, since ancient astronomers failed to detect star motion (or stellar parallax), most ancient astronomers believed that objects beyond Saturn were heavenly entities mounted on a big star ball (or the celestial sphere), unlike the planets, the Sun, the moon, and the Earth. Figure 1.2 depicts such an ancient Universe model with stars and regions of the zodiac fixed on the outermost shell.

1.2 The Development of Positional Astronomy

Shortly after Copernicus introduced the heliocentric model in 1543, observational astronomy advanced to another level when Galileo Galilei (1564-1642) pointed the newly invented telescope at the night sky. Telescopes enabled a new level of observation so that observational astronomy was no longer limited by the resolving power of the naked eye. New objects like moons of Jupiter, craters on the moon, and many previously-unobserved stars were soon discovered by Galileo. Later on, in 1687, Newton developed his theory of gravitation, and he was able to devise a physical model, published in his book "The Mathematical Principles of Natural Philosophy," \[126\] which accounted for the laws of planetary motion proposed by Kepler. This step took us from a pure mathematical model to a physical model associated with the physical process behind the observation. Astronomers...
then worked hard with Newton's three-dimensional planetary model, and this gradually led to the discovery of more planets in our Solar system. With the exception of Uranus in 1781, which was discovered accidentally by William Herschel, the discovery of Neptune in 1846 (Urbain Le Verrier and John Couch Adams) and of Pluto in 1930 (Clyde Tombaugh) were based on Newtonian mechanics. Newton's laws continue to form the physical basis of astronomy in modern times: the Jet Propulsion Laboratory (JPL) of NASA has precisely computed the three-dimensional planetary motions in our Solar system, namely the Solar System Ephemeris, over hundreds of years with an accuracy of less than 25 meters for planets and less than 1 meter for the moon [162, 163]. Similarly, detailed Earth satellite orbit information is available in the NORAD Element Sets [92].

In addition to three-dimensional planetary motions within our Solar system, three-dimensional star information began to be available as the resolving power of telescopes increased. In fact, during the 1700's, knowing the fact that the Earth is orbiting about the Sun, James Bradley had already attempted to measure stellar parallax by exploiting the Earth's orbital motion. However, since no telescope at his time could resolve angles smaller than one arc-second, his attempt failed [65]. Finally, in 1838, Friedrich Bessel (and simultaneously Wilhelm Struve and Thomas Henderson) observed the star "61 Cygnus" in the constellation Cygnus at different times of the year and found that "61 Cygnus" had a parallax of 0.3" (arc-seconds). The star ball concept could then finally be abolished and stars were known to be individual objects located in an enormous three-dimensional space, just like our Sun. Recently, in 1997, the European Space Agency [53, 137] finished measuring the parallax of 118,218 stars and published the result in the Hipparcos Catalogue; positions of stars ranging from a few light years up to hundreds of light years became available. Other popular star catalogs include the Bright Star Catalog developed by Hoffleit et al. [90] at the Yale University Observatory, which includes fewer 3D star positions than the Hipparcos Catalogue.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
With the development of spectroscopy in 1814 by Joseph Fraunhofer, scientists were then able to measure the electro-magnetic (EM) spectrum of light. In 1842, Christian Doppler of Austria observed that the wavelengths of light emitted from stars could be shifted. This discovery established the famous Doppler effect, which became an important astronomical tool to study and measure the Universe. Astronomers use indicators such as size, brightness, etc., to study nearby galaxies by comparing them with our own Milky Way galaxy. Certain very bright stars, the Cepheid variables discovered by Henrietta Leavitt, were found to be so consistent in their behavior that their brightness alone permitted direct measurements of intergalactic distances. In addition, other approaches like measuring brightest stars, HII regions, globular clusters, etc., served as additional indicators to enable the measurement of galaxies from 100 kparsec to 1000 kparsec in distance. However, beyond 1000 kparsec, Hubble’s law has to be used to find the distances to the galaxies [65]—proposed in 1923, this law relates galactic distances with the red shift values measured from the Doppler shift. One of the highest quality galaxy catalogs was compiled recently by Brent Tully at the University of Hawaii, and records red shift values of about 35,000 galaxies. Figure 1.3 shows rendering examples of three-dimensional planets, stars, and galaxies resulting from this research work.

![Fig.1.3: Simulation of three-dimensional planets, stars, and galaxies resulting from Positional Astronomy.](image)

With modern high power telescopes, both earth-based and satellite-based, astronomers
began to gather three-dimensional information about extra-solar planets starting about two
decades ago. More and more extra-solar planets have been discovered during the last
decade [112, 113]. Through June 23th 2003, 102 extra-solar planets have been discov­
ered [148] and the extra-solar planet dataset includes masses, periods, radii of revolution,
and eccentricities. For example, on April 15th 1999, three large extra-solar planets were
identified orbiting around the star Upsilon Andromedae, marking the first time that a multi­
planet solar system similar to our Solar system was confirmed to exist.

1.3 Positional Astronomy meets Computer Graphics

In the early 1970’s, not long before the time when astronomers began to search for extra­
solar planets, computer scientists began to study computer graphics and to apply computer
graphics technology to visualize data from different areas of science [127]. Making use
of computer graphics technology, scientists from different disciplines can view their data
from a new perspective (viewing mode, pseudo-coloring, image-processing, etc), as well as
processing and manipulating their data interactively. In recent decades, numerous visual­
ization systems have been developed for scientific disciplines such as: molecular chemistry,
geometry and mathematics, biomedical science, aerial or flight simulation, etc. Of course,
astronomical visualization is one of them.

When positional astronomy information is put into a computer graphics system, a new
level of understanding and exploring the night sky and the Universe becomes possible, just
as it did when Galileo first looked at the night sky through his newly invented telescope.

As summarized in Figure 1.4, the learning process of astronomical knowledge began
with observation; ancient astronomers made records and predictions based on observed
repetitive patterns. Next, mathematical models like Ptolemy’s, Copernicus’, or Kepler’s
were created to describe the geometry, while Newton’s physical model advanced our understanding by giving physical meaning to the mathematics. Standing on the shoulders of these giants, we have the luxury of accessing many varieties of three-dimensional astronomical information. With the help of computer graphics, virtual reality, and computer simulation technology, we can interactively visualize accurate physical models as well as manipulating real data. In this information age, astronomers are generating data at such a high rate that it is no longer feasible to view the data using printed text or graphs. Visualization technology raises the level of understanding so that astronomers can acquire more knowledge and insights from their data. One example is the Great Wall structure at the intergalactic scale. Scientists built a three-dimensional model of galactic density distributions and found that a large number of galaxies were concentrated on surfaces in thin sheets, while the remaining space is relatively empty [71].
1.4 Related Work

Astronomy Animations

With virtual reality technology, travel into outer space is no longer limited to voyages made by astronauts or space probes built by engineers. Graphics technology enables us to enjoy virtual journeys through space and time, looking at stars and planets from any desired location unrestricted by the bonds of Earth's gravity, and navigating across the huge galactic environment unlimited by the speed of light.

One early example of using computer graphics technology for virtual journeys is Blinn's set of "Voyager Fly-by Animations" [19, 55, 56, 57] (1977-1987) produced in the Jet Propulsion Laboratory of NASA. This series of animations illustrated various space missions including the Voyager 2 fly-by of Jupiter and the Pioneer 11 fly-by of Saturn. Subsequently, these fly-by animations were used both for mission planning and for presentation to the public. Later on, Blinn also developed animations used in Carl Sagan's COSMOS PBS series (1979-1980) [20] and in the California Institute of Technology's Mechanical Universe educational video series (1983-1986) [21, 58, 59, 60].

At about the same time, in 1977, Charles and Ray Eames produced a ten-minute film, called "Powers of Ten" [29], depicting a virtual journey across the scales of the Universe, from nuclear scale up to extra-galactic scale. In the animation, the journey starts with a picture set at one meter, the scale of our daily life, which shows a man sleeping on the field. Zooming into this picture, the next picture at a scale of $10^{-1}$ meters shows his hand gradually filling the field of view. By repeating this mechanism, the animation makes an exponential zoom smoothly into the hand of the man, into a blood vessel, and finally finishes at the nucleus of a carbon atom. Conversely, by reversing this cycling mechanism, the film zooms smoothly out from the nucleus back to the man, to the whole city of Chicago, to the Earth, to the Milky Way, and finally to the entire observable Universe. Annotated pictures
<table>
<thead>
<tr>
<th>Year</th>
<th>Title</th>
<th>Authors/Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1977</td>
<td>Powers of Ten</td>
<td>Charles and Ray Eames [29]</td>
</tr>
<tr>
<td>1979</td>
<td>Voyager 2</td>
<td>Blinn et al. [55]</td>
</tr>
<tr>
<td>1980</td>
<td>JPL/Saturn</td>
<td>Blinn et al. [56]</td>
</tr>
<tr>
<td>1982</td>
<td>Galileo</td>
<td>Blinn et al. [57]</td>
</tr>
<tr>
<td>1983</td>
<td>Space Simulator</td>
<td>Galicki [69]</td>
</tr>
<tr>
<td>1984-86</td>
<td>The Mechanical Universe</td>
<td>Blinn et al. [58, 59, 60]</td>
</tr>
<tr>
<td>1988</td>
<td>Interaction of Cosmic Strings</td>
<td>Fangmeier [62]</td>
</tr>
<tr>
<td>1989</td>
<td>Mars – The Movie</td>
<td>KJ Hussey, RA Mortensen, CT Kelly [95]</td>
</tr>
<tr>
<td></td>
<td>Voyager: Journey to the Outer Planets</td>
<td>Rueff [146]</td>
</tr>
<tr>
<td>1991</td>
<td>Cosmological N-Body Simulations</td>
<td>Richards [144]</td>
</tr>
<tr>
<td></td>
<td>Voyager</td>
<td>Van Ogtrop [132]</td>
</tr>
<tr>
<td></td>
<td>Magellan at Venus</td>
<td>Eric M. DeJong and Jeffery R. Hall [42]</td>
</tr>
<tr>
<td>1996</td>
<td>Cosmic Voyage</td>
<td>Bayley Silleck (IMAX Corporation) [153]</td>
</tr>
<tr>
<td></td>
<td>Cosmic Voyage Galaxy Formation &amp; Interaction</td>
<td>NCSA/UIUC [124]</td>
</tr>
<tr>
<td></td>
<td>Cosmic Voyage: Scientific Visualization for IMAX film</td>
<td>Cox, D.J. [138]</td>
</tr>
<tr>
<td>1997</td>
<td>Scientific Visualization of Space Based Debris</td>
<td>Tim Cotter [37]</td>
</tr>
<tr>
<td>1999</td>
<td>Galaxy Cluster Dynamics</td>
<td>Anjana Kai [101]</td>
</tr>
<tr>
<td>2000</td>
<td>Cosmic Clock</td>
<td>Andrew J. Hanson and Philip C.W. Fu [80]</td>
</tr>
<tr>
<td></td>
<td>Volume Visualization of the Orion Nebula</td>
<td>Carter Emmart, David R. Nadeau, Jon Genetti, and Erik Wesselak [51]</td>
</tr>
<tr>
<td></td>
<td>Passport to the Universe</td>
<td>Hayden Planetarium Show [5]</td>
</tr>
</tbody>
</table>

Table 1.1: Astronomy Animations.
shown in this animation were published in the companion book, “Powers of Ten” [121], by Philip Morrison and Phylis Morrison in 1982. Following these pioneer works in the 1970s, more astronomical animations were produced and presented to the general public; Table 1.1 shows a chronological list of some of the most prominent astronomical animations of which we are aware.

**Computer Graphics Literature related to Astronomy**

As computer graphics became popular, researchers began to apply it to astronomy [129, 161, 70]. One example of an important astronomical event in which computer graphics played a role was the head-on collision of the Comet Shoemaker-Levy 9 with Jupiter in 1994. A physical simulation was made and compared with the actual explosion [99]. Subsequently, Gooch [74] employed volume rendering techniques to study radio spectral data from the night sky — two spatial dimensions and one frequency dimension made up the three-dimensional data volume. And in 1996, Mathews [115] of the National Space Science Data Center (NSSDC) proposed to use VRML [12] (Virtual Reality Modeling Language) for three-dimensional astronomy data so that astronomers could share and visualize results over the web more efficiently.

Simulation and visualization have been used together to explore astronomical bodies. For example, Wijk et al. [122, 170] studied the scattering properties of the atmosphere of Venus by comparing the simulated result with actual pictures taken by a Venus mission; this permitted the estimation of the parameters of the atmosphere of Venus. Ostriker and Norman [133] at NCSA later proposed a framework for simulating cosmology, and reviewed the requirements for high performance computing, file management, and visualization systems needed for the task. Various animations such as [62, 124, 38] were produced by this NCSA group. Kahler et al. [100] of NCSA also simulated the life-span of a star using a supercomputing simulation. Volume rendering methods were directly applied to the AMR
In this research work, Genetti and Nadeau et al. [72, 123] simulated a fly-through of a three-dimensional model of the Orion Nebula using volume rendering methods [171]. The animation was shown in both a Hayden Planetarium show [5] and the Siggraph evening show [51] in 2000. Our own research work led to another animation, "Cosmic Clock," that was shown in the same Siggraph 2000 evening show. Our animation explored the scale of the Universe and explained the concept of cosmological time through the device of a cone-shaped model ("the Lightcone representation") of the spatiotemporal sources of observed light.

In addition to supercomputer-based simulations, photo-realistic simulations began to thrive in the late 1990's. Stutz et al. [164] attempted to build a comprehensive system for modeling the Solar system and the planetary motions accurately [131, 118] using computer graphics. Oberschelp [130] described the rendering of eclipses and Baranoski [9] proposed a rendering method for simulating the Aurora Borealis, known as the northern lights. Jensen et al. [97, 98] worked out a physically-based model to render the night sky as seen from Earth; it included the modeling and rendering of stars, the moon, the Milky Way [119], and the zodiacal light. However, the viewpoint was limited to the surface of the Earth. In other work [94, 93, 176, 175], relativistic effects on rendering were studied, showing physical phenomena observed when moving at speeds close to the speed of light, as well as general relativistic effects of strong gravitational fields.

Computer graphics techniques have also proven useful for navigation planning, for example, in the Mars Pathfinder project [43]. After the Pathfinder acquires pictures on Mars using its stereo cameras, the three-dimensional environment around the Pathfinder can be reconstructed. Based on this three-dimensional model, the management team in the NASA center on Earth can plan the next job for the Pathfinder on Mars. Note that the distance from Earth to Mars is so large that real-time navigation control has proven to be impossible; the management team has to plan the job for the Pathfinder beforehand. By examining...
the computer-generated three-dimensional environment, scientists will be able to generate plans efficiently.

**Astronomy-related Software**

Since this research work is intended to construct a large-scale visualization framework for virtual astronomy, we will review some astronomy software and visualization systems in this section. In the late 1980s and early 1990s, due to the limited computational power of available hardware, astronomy software was generally limited to 2D star chart programs or Solar system modeling [19, 164, 131, 118].

The webpage “www.seds.org/billa/astrossoftware.html” lists most of the astronomy software available for Mac, PC, or even Palm. These packages include popular stargazing software such as the commercial packages: Starry Night [159], Digital Universe [117], Voyager III [158], and TheSky [18], as well as the shareware packages: Stellar 2000 [160] and Celestia [54]. Though some of the software listed in the webpage does offer some level of three-dimensional navigation around the Solar system or nearby stars, most software focuses on two-dimensional star charts for amateur stargazing. In industry, Loren Carpenter developed the Star Renderer, which is then integrated into Pixar’s commercial particle renderer called Starman. This was used in the IMAX film “Cosmic Voyage” [153]. On the other hand, there are several commercial software systems driving real planetariums, including Evans & Sutherland’s “Digistar 3” [116], SkyScan’s “SkyVision” [156], and SGI’s on-going project “Digital Universe” [151], etc.

Around the same time this research work was being developed, some popular open-source astronomy projects such as Celestia [54], Stellarium [30], and Open Universe [4] were developed in the community. Celestia [54] (version 1.2, 2001–2003) and Open Universe [4] (the project ended in Nov. 2001) both offer navigation around the Solar system as well as nearby star clusters. But interestingly, with Celestia, we can fly to a nearby
galaxy like M10 (it includes only a few nearby galaxies around our Milky Way) using its exponential zoom feature. This implies that some kinds of large-scale visualization technology are embedded in the Celestia system. Though this feature sounds similar to the power homogeneous coordinate concept developed in our research work [83] during 2000, no literature or documents are published about this software feature, so details about this technique are unknown. The Stellarium package [30] is an earth-based astronomy system simulating the night sky as seen from Earth.

In the academic world, Joel Welling [177] implemented his star rendering method at the Pittsburgh Supercomputing Center, which was then used to create animations for astrophysical particle simulation data. In addition, Stuart Levy at NCSA developed the partiview visualization system [107], which was used for various astrophysics projects in NCSA. This software provides real-time navigation of astrophysical environments across several orders of magnitude on a variety of platforms such as the CAVE, SGI, PC, etc. — partiview was used by the Hayden Planetarium as well. Furthermore, Thiebaux et al. developed the Virtual Director [167] as a tool for creating animation paths in the CAVE system. It can be used together with partiview for creating animations.

1.5 Problems to be Addressed in this Research Work

1) Conventional Graphics Hardware. Except for a few software systems such as [54, 4, 107, 167], most astronomy software implementations are earth-based visualization systems. That means this software restricts the viewpoint to the Earth’s surface and looks at the night sky only as seen from the Earth. Nevertheless, with the exception of Celestia [54], which appears to offer some large-scale navigation, all three-dimensional-based software limits the navigation ranges to maintain compatibility with the graphics hardware. The reason for this limitation relates to three issues in conventional graphics hardware: 1) the
floating point representation, 2) the coordinate transformation mechanism, and 3) precision limits in depth buffering. We will discuss these issues in Chapter 2 of this thesis in detail, and present our own comprehensive solution.

2) Too much Data. Another difficulty in implementing a large-scale virtual astronomy system arises from the fact that there is an extremely large amount of data. The numerous objects that must be accounted for include stars, galaxies, planets, etc. Considering just stars, we have access to over 100,000 stars in the Hipparcos dataset [53, 137], so it is already very hard to render all of them on the screen in real-time. Hence, methods to cache rendering results and to cull unnecessary rendering are essential.

3) Large-scale Navigation. The last difficulty is related to navigation. Since the Universe is huge and mostly filled with empty space, astronomical bodies are usually separated by enormous distances. If we use just an unguided free navigation model in a variable-viewpoint astronomy system, users will easily get lost in the vast emptiness and fail to locate objects of interest.

1.6 Goals of this Research Work

Goal of this research work

Because of the difficulties listed above, in order to set up a large-scale virtual astronomy simulation system, we must address at least the following problems:

- Develop techniques to model space so that conventional graphics hardware (as well as computing hardware) will not hit the machine limits (e.g., precision) when representing, computing, or manipulating positions or vectors in the space.
1. Introduction

- Develop techniques for modeling and rendering astronomical objects (e.g., stars, galaxies, etc) with the requirement that these objects can be rendered across huge viewing scales.

- Develop visualization techniques for time so that we can visualize the effect of time in the Universe and observe objects in motion.

- Develop scalable navigation techniques that support travel in space with the guarantee that we will not easily get lost.

- Provide rendering and navigation in real-time so as to ensure effective visualization and user friendliness.

Research Outcome

The final result of this research work is a theoretical and practical framework enabling the continuous interactive exploration of virtual representations of the entire Universe. These results have been realized as an assortment of visualization tools (starview, galaxyview, earth-based view, sky-alignment tool, multi-spectral visualization of the sky, gas cloud editing tool, etc), two animations ("Solar Journey" and "Cosmic Clock") explaining astronomical concepts, and techniques suitable for incorporation into a real planetarium system (the Digital Universe system being developed with SGI). Furthermore, since the large-scale framework is independent of the data context, it is possible in principle to employ the framework for other visualization applications such as physical data from the Planck-length scale up to molecular and human scales.
1.7 Organization of this Thesis

This thesis is divided into five parts: 1) large-scale visualization framework, 2) modeling and rendering for virtual astronomy, 3) visualizing cosmological time, 4) large-scale navigation, and 5) results and conclusion. The organization outline is depicted in Figure 1.5.

In part I, we will first review the existing graphics rendering pipeline and discuss in detail the reasons why it is not capable of doing large-scale visualization across huge spatial scale if used naively (Chapter 2). Then, we will introduce the power homogeneous coordinate (PHC) concept in Chapter 3. This is the basic building block used to represent spatial quantities such as distances, vectors, and positions in large-scale environments. Based on this representation, we can generalize the transformation (modelview and projection transformations) mechanism in the existing graphics pipeline: the details of the method (incorporating PHC into the transformation unit and the depth rescaling method) and the implementation will be presented in Chapter 4 and Chapter 5, respectively. Based on the PHC representation, we derive the Environment Caching Criterion (Chapter 6) and the Object Disappearance Criterion (Chapter 7) to accelerate large-scale rendering. Basically, the large-scale framework presented in part I is independent of the context of the data in the visualization system; in addition to large-scale environments like astronomy, the framework is suitable for bio-molecular environments, physical environments (molecular, atomic, or Planck-length scale), or any environment spanning huge spatial scales.

Part II puts our large-scale framework into practice in the context of astronomical data. Here, an assortment of astronomical data is studied, and we present our modeling and rendering methods for each data type in detail: the Solar system model (earth, satellites, planets, etc.) (Chapter 8), stars (Chapter 9), the Milky Way galaxy (Chapter 10), as well as the galaxy catalog (Chapter 11).
Figure 1.5: Organization of this Thesis.
Part III further extends our large-scale framework from the spatial domain to the temporal domain in the context of astronomy. Chapter 12 introduces an iconic representation of the Universe, called the Lightcone, which helps us to visualize whole spatiotemporal framework of the Universe at a single glance. Chapter 13 presents several issues: time simulation, animation problems in dynamic environments, etc.

Part IV focuses on navigation in large-scale environments; Chapter 14 presents several navigation models developed for large-scale environments as well as some related rendering methods.

Part V presents the two last chapters: Chapter 15 shows some of our developed tools and animations, while Chapter 16 draws the final conclusion and summarizes the ideas and techniques developed in this research work.
Part I

A Framework for Large-Scale Visualization
In this part, we will detail our framework for large-scale visualization, which includes a number of representations and models for large-scale rendering as well as some techniques for accelerating the rendering performance. Since the devised models and techniques in this part are independent of the rendering context, the proposed framework is applicable not only for virtual astronomy, but could in principle be applied to bio-molecular environments as well as physical environments (molecular, atomic, or Planck-length scale), etc.

Chapter 2 will first review existing computer graphics technology, especially graphics hardware problems related to large-scale rendering. Then, Chapter 3 will present the power homogeneous coordinate representation for modeling large spatial quantities such as distances, vectors, and positions in large-scale environments.

Based on the power homogeneous coordinate representation, Chapters 4 and 5 will enhance the transformation unit in the context of OpenGL so that we can avoid graphics hardware problems related to large-scale rendering. Finally, Chapters 6 and 7 present two performance-accelerating criteria for large-scale environments so that we can cache faraway objects as images and cull objects that are too small.
Existing Graphics Pipeline

2.1 Problems in Existing Graphics Hardware

In section 1.5, we listed some of the difficulties in implementing a large-scale astronomical system. One of these is the rendering architecture of existing graphics hardware, which limits the navigation scale for continuous motions; three problems in the graphics hardware dominate our concerns:

1. Floating point representation. The first problem arises from the floating point representation. In particular, for the IEEE floating point number representation used for a float in C, there are two restrictions. First, the representation only guarantees 6 digits of precision (refer to the definition FLT_DIG in “limits.h” for Unix or “float.h” for MS Windows). Second, the representation supports ranges from 1.175e—38 to 3.403e38 only (refer to the definitions FLT_MIN and FLT_MAX). If we use a float to store distances as large as $10^{26}$ meters, multiplying or squaring values like this will cause numeric overflow. We could of course choose double precision to increase the numeric accuracy as well as the numeric range. However, due to the fact that most graphics hardware uses floating point arithmetic for higher performance, the limiting float arithmetic is usually implemented internally in
2. Existing Graphics Pipeline

the hardware. As a result, we have to assume float arithmetic is the only guaranteed numeric representation.

2. Transformations. The second problem concerns the transformations [22] built into most conventional graphics hardware. The transformation unit is a pipeline model consisting of a series of transformations that map user-input vertices to screen coordinates in a sequence: The unit includes the modelview (or view-world) transformation, the projection transformation, the perspective division, and the viewport transformation. Nevertheless, because this model assumes explicit inputs for spatial quantities such as coordinates and translations, we have to provide coordinates explicitly as \((X, Y, Z)\) to the transformation unit. As a result, we have to directly supply float values corresponding to 3D positions in the astronomical environment; thus, naive approaches can easily produce overflow or underflow in the internal computation of transformations.

3. Precision in Depth Buffering. The last problem occurs at the end of the rendering pipeline, during depth buffering. Basically, depth buffering [28, 76] is a technique designed to resolve visibility, determining which polygon (or which part of it) is visible from the camera position when projected onto the screen. This method keeps track of a per-pixel depth value to determine the object fragment closest to the eye; the memory holding these depth values is called the depth buffer. Depth buffering is the most common approach used in graphics hardware to resolve visibility due to its simplicity. However, because perspective division is non-linear in depth, the depth value stored is non-linear with respect to the actual depth value in the view-world space (the three-dimensional space). As a result, if we are mapping depth values from large-scale scenes into the depth buffer, we can lose significant precision in the stored depth values and visibility may not be solved correctly. The situation becomes worse in older generation graphics hardware that supports only 8-bits or 16-bits per-pixel precision in the depth buffer. Though most recent PC graphics hardware
systems such as the NVidia GeForce series and Quadro series support 16-bits or 32-bits of depth buffer precision, this is still inadequate for depth ranges encountered in astronomical environments.

In this chapter, we will first review the transformation unit in the fixed T&L (transformation and lighting) rendering pipeline, which is built into almost all existing graphics hardware made by vendors like SGI, NVidia, ATI, etc. Note that the word “fixed” means that the rendering pipeline is hardcoded in the graphics hardware and is not subject to change. Then, we will look at problem #3 in more detail and analyze the efficiency of depth buffering if we are using the fixed T&L rendering pipeline directly. At the end of the chapter, we will briefly describe some recent advances (vertex shading [110], fragment (or pixel) shading [140], and Cg programming [114]) in graphics hardware, which generalize the fixed T&L rendering pipeline by offering more programmability at the hardware level. This capability can make transformations in large-scale environments much more efficient.

2.2 Transformation Unit in the Fixed T & L Pipeline

![Diagram of Transformation Unit](image)

Figure 2.1: Transformation unit: coordinate transformation sequence.

In this section, we review the transformation unit in the fixed T&L rendering pipeline in detail. Basically, the transformation unit of the rendering pipeline includes the modelview (or view-world) transformation, the projection transformation, the perspective division, and
the viewport transformation as depicted in Figure 2.1. Since we are concerned with large-scale transformations in 3D only, the viewport transformation, which takes care of two-dimensional screen coordinates, is omitted in our discussion. Readers familiar with the mathematics of the transformation unit may skip this section.

To make the discussion concrete, we will use OpenGL [149, 125] commands and terminology from now on. Note that OpenGL is a standard device-independent graphics library, widely used for delivering real-time graphics applications in both academics and industry. Other graphics programming packages such as VRML [12], Performer [145], and Open Inventor [96] are actually higher-level packages, often built on top of OpenGL. Although Microsoft developed Direct 3D [120] to compete with OpenGL, Direct 3D [120] nevertheless follows the same fixed $T\&L$ rendering pipeline as OpenGL without any major differences; we can therefore discuss the transformation unit in terms of OpenGL without any significant loss of generality.

Furthermore, it is worth mentioning that in addition to the graphics programming packages we noted above, there is one more popular package called RenderMan [138] introduced by Pixar in 1989. Instead of targeting real-time graphics applications, this package focuses on generating high-quality images for industrial animations, and so we omit its discussion.

**Modelview Transformation**

The basic entry point for all OpenGL transformations is the `glVertex` call, through which all 3D geometry is passed to the rendering engine. (We include by implication the parallel functions like `glDrawArrays`, `glDrawElements`, etc.) In this transformation model, the arguments to `glVertex4f(x, y, z, w)` form a $4 \times 1$ world-coordinate column
Existing Graphics Pipeline

2. Existing Graphics Pipeline

matrix,

\[
\bar{v}_{\text{world}} = (x_{\text{world}}, y_{\text{world}}, z_{\text{world}}, w_{\text{world}})^T, \\
\bar{v}_{\text{world}} = (x_{\text{world}}, y_{\text{world}}, z_{\text{world}}, w_{\text{world}})^T, \\
\text{which is a homogeneous coordinate representing the three-dimensional position}
\]

\[
\begin{pmatrix}
    x_{\text{world}} \\
    y_{\text{world}} \\
    z_{\text{world}} \\
    w_{\text{world}} \\
\end{pmatrix}.
\]

Note that using this representation, a single three-dimensional position can be expressed using many alternative homogeneous coordinates with different scaling by \( w_{\text{world}} \). Also, using homogeneous coordinates, we can represent positions at infinity by having \( w_{\text{world}} \) set to zero. In most cases, OpenGL programmers leave \( w_{\text{world}} = 1 \) for simplicity.

After getting the scene geometry from the \texttt{glVertex} call, the OpenGL kernel right-multiplies the input column-vector, \( \bar{v}_{\text{world}} \), by the 4x4 modelview (or view-world) matrix to transform the coordinate from the world space to the eye space. The resulting 4x1 vertex is called the eye coordinate, which is also a homogeneous coordinate. Note that expressing three-dimensional transformations as 4x4 matrices facilitates efficient transformations with homogeneous coordinates.

\textit{Notation:} For notational convenience, we will normally drop the explicit transpose when writing vectors in the following unless it is specifically required for clarity: thus \( \bar{x} = (x, y, z, w) \) should be understood in general as equivalent to the column vector \( \bar{x} = (x, y, z, w)^T \), and \( \bar{x} = (x, y, z) \) is equivalent to \( \bar{x} = (x, y, z)^T \).

In OpenGL eye space, the eye, which also may be thought of as the camera or viewpoint, is fixed at the origin, looking towards the negative \( z \)-axis with the positive \( y \)-axis as its view-up direction. Details of this camera model can be found in the OpenGL Programming Guide [125].

To illustrate the mathematics, let \( M_v \) be the 4x4 modelview matrix. In general, this matrix is composed, in right to left order, of a scaling, a rotation, and a translation and so
2. Existing Graphics Pipeline

may be written as \( M_v = TRS \); it is then left-multiplied on a homogeneous vector. If we define \( A \) as the \( 3 \times 3 \) top-left sub-matrix of \( M_v \) and \( T = (T_x, T_y, T_z)' \) as the translational component, we may write

\[
M_v = \begin{pmatrix}
A & T \\
0 & 1
\end{pmatrix}.
\]

Note that \( A \) normally consists solely of scaling and rotation (though it may possibly include also shear and reflection), and that the bottom row of \( M_v \) has three leading zeros followed by a one.

For simplicity, we put \( u_{\text{world}} = 1 \), so \( \vec{v}_{\text{world}} = (x_{\text{world}}, y_{\text{world}}, z_{\text{world}}, 1) \) and

\[
\vec{v}_{\text{eye}} = M_v \vec{v}_{\text{world}} = \left( A \begin{pmatrix} x_{\text{world}} \\ y_{\text{world}} \\ z_{\text{world}} \end{pmatrix} + T, 1 \right).
\]

Here, \( T \) can be understood as the displacement vector from the eye space origin to the world space origin measured in the eye space. Note that the advantage of using \( 4 \times 4 \) homogeneous matrix is that transformations can be carried out in the form of matrix multiplication not only for scaling and rotation but also for translation. As a result, the computation becomes simpler and easier to implement in graphics hardware.

Figure 2.2 illustrates the transformation process. The car in Figure 2.2(a) is located in its own object space. After the modelview transformation, it is put in front of the camera (eye or viewpoint) at the origin (Figure 2.2(b)).

**Projection Transformation**

After the modelview transformation, we project the eye coordinates to the virtual film plane by left-multiplying the vector using the \( 4 \times 4 \) projection matrix, \( M_p \). The resultant coordinate is again a homogeneous coordinate called the **clip coordinate**, denoted by
2. Existing Graphics Pipeline

Figure 2.2: Modelview Transformation.

\[ \vec{v}_{\text{clip}} = (x_{\text{clip}}, y_{\text{clip}}, z_{\text{clip}}, w_{\text{clip}}); \] this space is called the **clip space**.

\[ \vec{v}_{\text{clip}} = M_p \vec{v}_{\text{eye}} = M_p M_v \vec{v}_{\text{world}}. \]

In conventional graphics systems, we define a **viewing frustum** to specify a 3D volume relative to the viewpoint in such a way that objects are only visible on the camera image if they are inside the frustum. The frustum is basically a truncated pyramid limited by six planes in the eye space, depicted as the wireframe in front of the camera in Figure 2.2(b): The front and back clipping planes are called the near and far planes respectively. In order to carry out the inside-outside test, we perform clipping in the **clip space** by comparing \( x_{\text{clip}}, y_{\text{clip}}, \) and \( z_{\text{clip}} \) against \( w_{\text{clip}} \). This is known as **homogeneous clipping** in the graphics literature: A clip coordinate is inside the viewing frustum if

\[
\begin{align*}
| x_{\text{clip}} | & \leq w_{\text{clip}} \\
\text{and} \quad | y_{\text{clip}} | & \leq w_{\text{clip}} \\
\text{and} \quad | z_{\text{clip}} | & \leq w_{\text{clip}} .
\end{align*}
\]

When we set up the near and far planes defined by OpenGL functions such as \( \text{glOrtho}, \) \( \text{glFrustum}, \) \( \text{gluOrtho2D}, \) \( \text{gluPerspective}, \) etc, we are putting these two planes in the eye space along the negative z-axis at \( z = -z_{\text{near}} \) and \( z = -z_{\text{far}}, \) respectively. After the
projection transformation, they are transformed to \(-z_{\text{near}}\) and \(+z_{\text{far}}\) in the clip space, as documented in the OpenGL Programming Guide [125].

**Perspective Division**

Finally, after the projection transformation and the clipping operation, we divide the clip coordinates by \(w_{\text{clip}}\) to obtain the **normalized device coordinates** (NDC):

\[
\left( x_{\text{NDC}} , y_{\text{NDC}} , z_{\text{NDC}} \right) = \left( \frac{x_{\text{clip}}}{w_{\text{clip}}} , \frac{y_{\text{clip}}}{w_{\text{clip}}} , \frac{z_{\text{clip}}}{w_{\text{clip}}} \right).
\]

It is important to note that at the near and far clipping planes on the clip space \(z\)-axis, the \(z_{\text{clip}}\) values are \(-z_{\text{near}}\) and \(+z_{\text{far}}\), while the \(w_{\text{clip}}\) values are \(z_{\text{near}}\) and \(z_{\text{far}}\) respectively. Thus, after the perspective division, the near and far planes are mapped to \(-1\) and \(+1\) in the NDC space, respectively. The limits of the \(x_{\text{NDC}}\) and \(y_{\text{NDC}}\) coordinates are also mapped to \([-1, +1]\), so the transformed viewing frustum in NDC space is a \(2 \times 2 \times 2\) cube with limits \([-1, +1]\) in all three dimensions. Finally, the NDC coordinates are used to map the scene to the pixels in the \((x, y)\) viewport using the viewport transformation, and to the depth buffer using the linear range of the \(z_{\text{NDC}}\) coordinates.

Note that \(z_{\text{NDC}}\) may contain a division by \(z_{\text{world}}\), which is normally contained in \(w_{\text{clip}}\) for the case of perspective projection, and thus is not necessarily linear in \(z_{\text{world}}\); this fact is a major source of the problems that cause inadequate precision in the depth buffer. We will give concrete examples to illustrate this problem shortly in the next section. The use of \(z_{\text{NDC}}\) in the depth buffer has far-reaching consequences that we will investigate further below.
Examples: Perspective and Parallel Projections

In this section, we give explicit examples of phenomena that may cause problems when objects with extreme scale values and ranges are rendered in a computer graphics scene using conventional methods. As before, the OpenGL system is used for definiteness, but the same phenomena would be present in other programming packages, given the same reference hardware.

We begin by demonstrating the effects of the projection transformation and perspective division, moving from eye space to clip space, and then to NDC space. We choose a sample scene with $41 \times 41$ points at uniform grid positions in the $yz$-plane of eye space. In Figure 2.3, the positive $y$-axis is the vertical direction, the $x$-axis is perpendicular to the paper, and the $z$-axis is horizontal with negative $z$ going to the right. Therefore, the eye point (viewpoint or camera), at the eye space origin, is looking towards the right along the negative $z$-axis. We use the following example projection matrices, illustrating both perspective and parallel transformations, and show the corresponding viewing frusta as wireframes in the Figure:

\[
\text{Perspective Projection Matrix} = \begin{pmatrix}
1.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 1.00 & 0.00 & 0.00 \\
0.00 & 0.00 & -2.00 & -9.00 \\
0.00 & 0.00 & -1.00 & 0.00 \\
\end{pmatrix}
\]

\[
\text{Parallel Projection Matrix} = \begin{pmatrix}
0.10 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.10 & 0.00 & 0.00 \\
0.00 & 0.00 & -0.33 & -2.00 \\
0.00 & 0.00 & 0.00 & 1.00 \\
\end{pmatrix}.
\]

Note that the near and far planes in eye space are at $z = -3$ and $z = -9$, respectively, as shown by the wireframe viewing frustum. Perspective projection maps these to $z = -3$.
Figure 2.3: Eye → Clip → NDC (without shearing). Positive $z$ is to the left, Positive $y$ points upward, and white wireframes indicates the viewing frusta.
and $z = 9$. The parallel projection matrix normally sets $w_{\text{clip}} = 1$, and so maps the depth boundary values immediately to the NDC coordinates $z = -1$ and $z = 1$, respectively. Perspective projection requires the additional perspective division, which finally maps all points inside the viewing frustum into the $[-1, +1]$ box in NDC space.

Figure 2.3 shows intuitively that the perspective transformation from eye space to clip space is a mirror reflection about the near plane, followed by an appropriate scaling around the fixed point where the $z$-axis intersects the near plane. We also see that the perspective division to NDC space involves a non-uniform division (scaling) putting negative infinity ($z = -\infty$ in eye space) at a finite position on the $+z$ side of NDC space, i.e., the point of convergence in NDC. For parallel projection, we also have a mirror reflection about the near plane, but the scaling afterwards is different; it brings the near and far planes to $[-1, +1]$ instead. Also, the clip space basically has the same structure as the NDC space for parallel projection because $w_{\text{clip}}$ is usually just one. Furthermore, we can shear viewing frusta (both perspective and parallel) in the eye space with the following modified projection matrices (see Figure 2.4).

<table>
<thead>
<tr>
<th>Perspective Projection Matrix</th>
</tr>
</thead>
</table>
| $\begin{pmatrix}
1.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.81 & -0.59 & 0.00 \\
0.00 & 0.00 & -2.00 & -9.00 \\
0.00 & 0.00 & -1.00 & 0.00
\end{pmatrix}$ |

<table>
<thead>
<tr>
<th>Parallel Projection Matrix</th>
</tr>
</thead>
</table>
| $\begin{pmatrix}
0.10 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.10 & 0.00 & 0.40 \\
0.00 & 0.00 & -0.33 & -2.00 \\
0.00 & 0.00 & 0.00 & 1.00
\end{pmatrix}$ |
Figure 2.4: Eye → Clip → NDC (with shearing). Positive z is to the left, positive y points upward, and white wireframes indicates the viewing frusta.
2.3 Analysis: Transformation Unit

In this section, we will devise various useful formulas employed in the transformation unit and analyze the efficiency of depth buffering.

Computing \( n \) and \( f \) from the Projection Matrix

Let \( M_p \) be a projection matrix, where \( M_p[i] \) is the \( i \)-th entry out of the 16 entries according to the OpenGL column-major layout:

\[
\text{projection matrix} = \begin{pmatrix}
\end{pmatrix}
\]

Now let \( n \) and \( f \) be the \( z \)-distances from the eye to the near and far planes in the eye space, respectively. Denote the intersections between these two planes with the \( z \)-axis in eye space as \( P_{\text{near}} = (0,0,-n,1) \) and \( P_{\text{far}} = (0,0,-f,1) \) respectively. By (left) multiplying \( M_p \) with \( P_{\text{near}} \) and \( P_{\text{far}} \), we get these clip coordinates:

\[
\begin{aligned}
\end{aligned}
\]

Note that we do not care about \( x_{\text{clip}} \) and \( y_{\text{clip}} \) in the computation.

Because of the near-plane and far-plane clipping, the corresponding coordinates in NDC space after perspective division are \( (0,0,-1) \) for \( n \) and \( (0,0,1) \) for \( f \), respectively.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Therefore, we have

\[
\]
\[
\]

\[
\]
\[
\]

Notice that the above computation is independent of the type of projection; it is true for both perspective and parallel projections.

**Distinguishing Perspective or Parallel Projection from values in the Projection Matrix**

As we saw in the previous section on standard OpenGL, \(w_{\text{clip}}\) depends strongly on the last row of the projection matrix.

**Case 1:** If all entries in the last row are zero, \(w_{\text{clip}}\) is zero and according to the homogeneous coordinate convention, and nothing will be viewable. This is a **null projection**.

**Case 2:** If the last row of \(M_p\) has only the final entry non-zero, say \(M_p[15] = 1\), then \(w_{\text{clip}}\) is a constant (provided \(w_{\text{eye}}\) is a constant, which is usually true). Then, \(w_{\text{clip}}\) is independent of \(x_{\text{eye}}\), \(y_{\text{eye}}\), and \(z_{\text{eye}}\) and the NDC coordinates have perspective division by a constant: \(x_{\text{NDC}}\) and \(y_{\text{NDC}}\) are proportional to \(x_{\text{eye}}\) and \(y_{\text{eye}}\) and this is a **parallel projection**.

**Case 3:** Conversely, if any of the first three entries of the last row are non-zero (typically the 3rd entry), the NDC coordinates involve perspective division of \(x_{\text{eye}}\) and \(y_{\text{eye}}\) by a linear expression \((az_{\text{eye}} + b)\) where \(a\) and \(b\) are real. This is a **perspective projection**.
If the programmer follows these conventions, we can distinguish parallel and perspective projections by examining the last row of the projection matrix.

**Where are the Eye point and $z = \pm \infty$?**

In the eye space, the eye point and $z = \pm \infty$ are located at homogeneous coordinates $E_{\text{eye}} = (0, 0, 1, 0)$, $I_{\text{eye}}^+ = (0, 0, 1, 0)$, and $I_{\text{eye}}^- = (0, 0, -1, 0)$ respectively. Note that $E_{\text{eye}}$ is at the origin, $I_{\text{eye}}^+$ is at $z_{\text{eye}} = +\infty$, and $I_{\text{eye}}^-$ is at $z_{\text{eye}} = -\infty$ of the eye space. In this section, we examine where they move to in the clip space and NDC space.

Again, let $n$ and $f$ be the absolute depths at the near and far planes, so that for a particular field-of-view, the analogs of the matrices for perspective and parallel projection (without shearing) used in the previous section can be written as the following:

$$
\text{Perspective Projection Matrix} = \\
\begin{pmatrix}
1.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 1.00 & 0.00 & 0.00 \\
0.00 & 0.00 & \frac{n+f}{n-f} & \frac{2fn}{n-f} \\
0.00 & 0.00 & -1.00 & 0.00 \\
\end{pmatrix}
$$

$$
\text{Parallel Projection Matrix} = \\
\begin{pmatrix}
0.10 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.10 & 0.00 & 0.00 \\
0.00 & 0.00 & \frac{2}{n-f} & \frac{n+f}{n-f} \\
0.00 & 0.00 & 0.00 & 1.00 \\
\end{pmatrix}
$$

The corresponding homogeneous clip coordinates for $E_{\text{eye}}$, $I_{\text{eye}}^+$, and $I_{\text{eye}}^-$ become

**Perspective Projection**

$E_{\text{clip}} = (0, 0, \frac{2fn}{n-f}, 0)$

$E_{\text{clip}}^+$ and $E_{\text{clip}}^- = (0, 0, -\frac{f+n}{f-n}, -1)$ and $(0, 0, \frac{f+n}{f-n}, 1)$

**Parallel Projection**

$E_{\text{clip}} = (0, 0, -\frac{f+n}{f-n}, 1)$

$E_{\text{clip}}^+$ and $E_{\text{clip}}^- = (0, 0, -\frac{2}{f-n}, 0)$ and $(0, 0, \frac{2}{f-n}, 0)$
Furthermore, in the NDC space,

\[
\begin{align*}
\text{Perspective Projection} & \\
E_{\text{NDC}} & = (0, 0, -\infty) \\
I^+_{\text{NDC}} \text{ and } I^-_{\text{NDC}} & = (0, 0, \frac{f + n}{f - n})
\end{align*}
\]

\[
\begin{align*}
\text{Parallel Projection} & \\
E_{\text{NDC}} & = (0, 0, -\frac{f + n}{f - n}) \\
I^+_{\text{NDC}} \text{ and } I^-_{\text{NDC}} & = (0, 0, -\infty, 0) \text{ and } (0, 0, \infty, 0)
\end{align*}
\]

Therefore, we can see that under perspective projection, the eye point goes from the origin in eye space to $-\infty$ in the NDC space, while the $z_{\text{eye}} = \pm \infty$ both map to the same finite point, \( z = \frac{f + n}{f - n} \), in the NDC space. This is the point of convergence on the positive $z$-axis (which points to the left) shown in Figure 2.3; since \( f > n > 0 \), \( \frac{f + n}{f - n} > 1 \) and this point of convergence is always more positive (to the left) than the upper limit of the the NDC box. On the other hand, for parallel projection, the eye point shifts from the origin to \( -\frac{f + n}{f - n} \), while \( \pm \infty \) flips to \( \mp \infty \) respectively.

**Conversion between \( z_{\text{eye}} \) and \( z_{\text{NDC}} \)**

In this subsection, we compute the function relating \( z \) in the eye space to \( z \) in the NDC space. First, let $-z_{\text{eye}}$ and $z_{\text{NDC}}$ be the corresponding $z$ values in eye space and NDC space, respectively. We add "−" before $z_{\text{eye}}$ so that $z_{\text{eye}}$ is positive along the line of sight (negative $z$-axis) and $z_{\text{NDC}}$ has the range $[-1, +1]$. We start from $P_{\text{eye}} = (0, 0, -z_{\text{eye}}, 1)$, in the eye space and transform it using the projection matrix $M_p$ as follows:

\[
M_p P_{\text{eye}} = (\ldots, \ldots, -M_p[10]z_{\text{eye}} + M_p[14], -M_p[11]z_{\text{eye}} + M_p[15])
\]

\[
z_{\text{NDC}} = \frac{M_p[10]z_{\text{eye}} - M_p[14]}{M_p[13]z_{\text{eye}} - M_p[15]}
\]
2. Existing Graphics Pipeline

With perspective projection, $M_p[11] = -1.0$ and $M_p[15] = 0.0$, we have

$$z_{\text{NDC}} = \frac{M_p[10]z_{\text{eye}} - M_p[14]}{-z_{\text{eye}}}$$

$$= -M_p[10] + \frac{M_p[14]}{z_{\text{eye}}}$$

$$z_{\text{eye}} = \frac{M_p[14]}{z_{\text{NDC}} + M_p[10]}.$$

With parallel projection, $M_p[11] = 0.0$ and $M_p[15] = 1.0$, we have

$$z_{\text{NDC}} = -M_p[10]z_{\text{eye}} + M_p[14]$$

$$z_{\text{eye}} = \frac{M_p[14] - z_{\text{NDC}}}{M_p[10]}.$$

Thus, $z_{\text{NDC}}$ is linear in $1/z_{\text{eye}}$ for perspective projection and linear in $z_{\text{eye}}$ for parallel projection.

Utilization of the Depth Buffer in Perspective Projection

The depth buffer is accessed linearly by the NDC coordinate $z_{\text{NDC}}$, but, as noted in the previous subsection, $z_{\text{NDC}}$ may have various dependences on $z_{\text{eye}}$, and hence on the world distance of a scene object from the eye point. We can now study the behavior of the depth buffer as a function of $z_{\text{eye}}$. First we note that the $z$ value of the conventional OpenGL depth buffer hardware range, requires $z_{\text{depth}}$ to be in the interval $[0, 1]$; thus

$$z_{\text{NDC}} = 2z_{\text{depth}} - 1,$$

$$z_{\text{eye}} = \frac{M_p[14]}{z_{\text{NDC}} + M_p[10]}.$$

(from previous section)

$$= \frac{M_p[14]}{2z_{\text{depth}} + M_p[10] - 1}.$$
Normally, for perspective projection, \( M_p[10] = \frac{n+f}{n-f} \) and \( M_p[14] = \frac{2fn}{n-f} \), so

\[
z_{\text{eye}} = \frac{\frac{2nf}{n-f}}{2z_{\text{depth}} + \frac{n+f}{n-f} - 1}
= \frac{2nf}{2(n-f)z_{\text{depth}} + (n+f) - (n-f)}
= \frac{nf}{(n-f)z_{\text{depth}} + f}.
\]

In Figure 2.5, we present a table and a graph showing the percentage utilization of the depth buffer as a function of \( z_{\text{eye}} \) for an assortment of different ratios \( (f/n) \) from 10 up to 10^7. From the table, we can see that the utilization of the depth buffer is highly non-linear and hyperbolic. Typically, almost half of the depth buffer is used for the range \( z_{\text{eye}} \in [n, 2n] \) in the eye space. Also, when the depth range is large, for example when \( f = 10000000 \), about \( \frac{20}{10000000} \times 100\% = 0.0002\% \) of the \( z \) range in the eye-space viewing frustum occupies 95% of the depth buffer.

To find out exactly how poorly the depth buffer is utilized, we can make use of the graph in Figure 2.5. The vertical axis denotes the % utilization of the depth buffer, while the horizontal axis is the relative position in the \( z \) range in the eye space, so that different near-far ranges, \([1, 10]\), \([1, 100]\), etc., are all compressed together.

We observe that if the depth range is huge, depth buffer precision will be quickly lost when objects are far from the eye point relative to the value \( n \) of the near cutoff plane. Because of this, animators often sacrifice correctness in scale and size to make sure their objects at different scales are viewable at the same time. This is undesirable in general, and specifically problematic for applications such as scientific visualization that demand precise distance representation.

To further understand the resolution of the depth buffer, we can take partial derivatives of \( z_{\text{depth}} \) with respect to \( z_{\text{eye}} \), using \( M_p[14] = (2fn)/(n-f) \), to calculate the sensitivity to
2. Existing Graphics Pipeline

<table>
<thead>
<tr>
<th>z-Range</th>
<th>[1, 10]</th>
<th>[1, 100]</th>
<th>[1, 1000]</th>
<th>[1, 10000]</th>
<th>[1, 100000]</th>
<th>[1, 1000000]</th>
<th>[1, 10000000]</th>
</tr>
</thead>
<tbody>
<tr>
<td>z-buffer</td>
<td>Z-eye</td>
<td>Z-eye</td>
<td>Z-eye</td>
<td>Z-eye</td>
<td>Z-eye</td>
<td>Z-eye</td>
<td>Z-eye</td>
</tr>
<tr>
<td>0</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>5</td>
<td>1.0471</td>
<td>1.0526</td>
<td>1.0526</td>
<td>1.0526</td>
<td>1.0526</td>
<td>1.0526</td>
<td>1.0526</td>
</tr>
<tr>
<td>10</td>
<td>1.0969</td>
<td>1.1110</td>
<td>1.1110</td>
<td>1.1110</td>
<td>1.1110</td>
<td>1.1110</td>
<td>1.1110</td>
</tr>
<tr>
<td>15</td>
<td>1.1561</td>
<td>1.1764</td>
<td>1.1764</td>
<td>1.1764</td>
<td>1.1764</td>
<td>1.1764</td>
<td>1.1764</td>
</tr>
<tr>
<td>20</td>
<td>1.2135</td>
<td>1.2497</td>
<td>1.2497</td>
<td>1.2497</td>
<td>1.2497</td>
<td>1.2497</td>
<td>1.2497</td>
</tr>
<tr>
<td>25</td>
<td>1.2803</td>
<td>1.3333</td>
<td>1.3333</td>
<td>1.3333</td>
<td>1.3333</td>
<td>1.3333</td>
<td>1.3333</td>
</tr>
<tr>
<td>30</td>
<td>1.3599</td>
<td>1.4286</td>
<td>1.4286</td>
<td>1.4286</td>
<td>1.4286</td>
<td>1.4286</td>
<td>1.4286</td>
</tr>
<tr>
<td>35</td>
<td>1.4599</td>
<td>1.5333</td>
<td>1.5333</td>
<td>1.5333</td>
<td>1.5333</td>
<td>1.5333</td>
<td>1.5333</td>
</tr>
<tr>
<td>40</td>
<td>1.5625</td>
<td>1.6667</td>
<td>1.6667</td>
<td>1.6667</td>
<td>1.6667</td>
<td>1.6667</td>
<td>1.6667</td>
</tr>
<tr>
<td>45</td>
<td>1.6807</td>
<td>1.8182</td>
<td>1.8182</td>
<td>1.8182</td>
<td>1.8182</td>
<td>1.8182</td>
<td>1.8182</td>
</tr>
<tr>
<td>50</td>
<td>1.8182</td>
<td>1.9802</td>
<td>1.9802</td>
<td>1.9802</td>
<td>1.9802</td>
<td>1.9802</td>
<td>1.9802</td>
</tr>
<tr>
<td>55</td>
<td>1.9802</td>
<td>2.1934</td>
<td>2.1934</td>
<td>2.1934</td>
<td>2.1934</td>
<td>2.1934</td>
<td>2.1934</td>
</tr>
<tr>
<td>60</td>
<td>2.1739</td>
<td>2.4631</td>
<td>2.4631</td>
<td>2.4631</td>
<td>2.4631</td>
<td>2.4631</td>
<td>2.4631</td>
</tr>
<tr>
<td>65</td>
<td>2.4096</td>
<td>2.8556</td>
<td>2.8556</td>
<td>2.8556</td>
<td>2.8556</td>
<td>2.8556</td>
<td>2.8556</td>
</tr>
<tr>
<td>70</td>
<td>2.7027</td>
<td>3.3256</td>
<td>3.3256</td>
<td>3.3256</td>
<td>3.3256</td>
<td>3.3256</td>
<td>3.3256</td>
</tr>
<tr>
<td>75</td>
<td>3.0769</td>
<td>3.8855</td>
<td>3.8855</td>
<td>3.8855</td>
<td>3.8855</td>
<td>3.8855</td>
<td>3.8855</td>
</tr>
<tr>
<td>80</td>
<td>3.5714</td>
<td>4.8077</td>
<td>4.8077</td>
<td>4.8077</td>
<td>4.8077</td>
<td>4.8077</td>
<td>4.8077</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 2.5: The Utilization of the Depth Buffer.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Therefore, for perspective projection, the variation in $z_{\text{depth}}$ decreases rapidly with $z_{\text{eye}}$ as $|z_{\text{eye}}|$ increases from its minimum value $n$ to its maximum value $f$: progressively less depth buffer precision is provided as $|z_{\text{eye}}|$ increases from $n$ to $f$.

Our solution to this problem is to introduce Power Homogeneous Coordinates, described in detail in the next chapter, in order to facilitate accessing the depth buffer precision in a more flexible manner. Another method proposed to solve this problem is to replace the value of $z$ in the depth buffer by its inverse, $1/z$, leading to the "OOZ" or "one-over-$Z$" buffer, also called the W-buffer (see, e.g., http://developer.nvidia.com/docs/IO/1331/ATT/ZBuffering2.doc by NVidia and http://sjbaker.org/steve/omniv/love_your_z_buffer.html by Steve Baker). The W-buffer method improves the utilization of the depth buffer, but requires modifications of the hardware and the driver program to support it, although some PC graphics hardware does support this feature now. Even so, though the W-buffer enhances the utilization of the depth buffer, it still has insufficient flexibility for the extreme scale ranges that we will see in astronomical applications.
2.4 Programmable Graphics: Shaders and Cg

Recent advances in programmable hardware such as FPGA (Field Programmable Gate Array) chips, introduce programmability at the hardware level. Vendors of graphics hardware have brought this technology into their graphics boards and generalized the fixed rendering pipeline model on the hardware with the aim of improving the visual quality of rendering results in real-time graphics, especially for the games industry. This advance in graphics hardware has led to the so-called shader programming, which gives graphics programmers more flexibility when processing vertices (vertex shading [110]) and performing sophisticated shading when filling screen pixels (pixel/fragment shading [140]). These advances were realized in the NVidia GeForce series and ATI Radeon around 2001–2002 and the API (Application Programming Interface) for both OpenGL and Direct 3D have been updated to support these features.

When working with vertex and pixel shaders, programmers have to write something like assembly code so that this code can be loaded and executed directly on the graphics hardware (this is done at run-time). However, writing assembly code like this is very inconvenient, bug prone, and difficult to debug. In 2003, NVidia and Microsoft proposed to improve this situation by supporting the Cg programming (C for Graphics) [114] and HLSL (High-level Shader Language), respectively, as high level languages for writing shaders. And in fact, this hardware realization brings real-time graphics closer to the shading philosophy in RenderMan [138] and the goal of this trend is that the rendering quality produced by RenderMan can eventually be realized directly in OpenGL hardware.

Vertex Program (or Vertex Shader)

The introduction of the OpenGL vertex shader (or vertex program) (in OpenGL 1.3 or above) allows us to write low-level assembly-like programs (vertex programs) to replace
the standard Transform & Lighting (T&L) unit in OpenGL. Strictly speaking, OpenGL-
version of the assembly code is called "vertex program" while the Direct 3D-version (or
Direct-X) is called "vertex shader", since we follow OpenGL conventions, we will use
"vertex program", but in reality, "vertex shader" means the same thing, with only minor
differences in the syntactic style of the coding.

As illustrated in Figure 2.6, OpenGL vertex programs wrap around the standard OpenGL
modelview and projection matrix multiplication operations; that is, after gl\textit{Vertex} or other
parallel OpenGL routines are called, gl\textit{Vertex} invokes any existing binding for an OpenGL
vertex program and passes its world coordinate arguments as input. Then, the OpenGL ver-
tex program performs its internal computation on the input world coordinate (together with
its vertex attributes such as color, normal, etc.), computes the resultant homogeneous clip
coordinates, and feeds the result into the graphics hardware for clipping, perspective divi-
sion, rasterization, and so on.

Figure 2.6: OpenGL Pipeline: (1) Transform & Lighting (T&L) Unit in Conventional
OpenGL (1.2 or before) and (2) Shader Enabled OpenGL (1.3 or after).
Mimicking Vertex Programs on Older Graphics Hardware

In essence, if we are executing a vertex program on the graphics hardware, the program is actually executing in the GPU (Graphics Processing Unit) specialized for handling graphics instructions in vertex and pixel/fragment processing. However, if the graphics hardware does not support vertex programming (e.g., we are using older graphics hardware), it is possible that we can still run the vertex program on the CPU (Central Processing Unit) to mimic the GPU. This will of course make the rendering process slower.

From Figure 2.6, we can see that vertex programs incorporate the standard modelview and projection matrix transformations, compute the resultant homogeneous clip coordinates, and output them to the clipping, perspective division, and rasterization units.

In this way, it is trivial to see that if we set both the modelview and projection matrices to be the identity matrix and wrap `glVertex` with our own subroutine, say `my_gIVertex`, we can perform the necessary vertex programming inside our own subroutine. As a result, the mimicked vertex program is executed on the CPU and if we call `glVertex` inside our `my_gIVertex` at the end, we can pass our vertex-program-modified clip coordinates directly to the clipping circuit, since we have both modelview and projection matrices set to the identity already.

Pixel/Fragement Shading and Cg: C for Graphics

In addition to vertex programs introducing programmability to vertex processing, pixel/fragment shading [140] was introduced into graphics hardware at the same time in 2001; basically, pixel/fragment shading is intended to introduce programmability for the per-fragment (or per-pixel) operations when the hardware is filling screen pixels. In essence, after transforming individual vertices, the whole polygon (triangle or quadrilateral) is rasterized on the screen and per-vertex attributes such as color, depth, texture coordinates, etc.,
are interpolated over the screen polygon. Then, color for each pixel will be computed based on the interpolated value (as well as texture lookup, if any) and the computed color will then be filled into the frame buffer. The advantage of having pixel/fragment shader support is that it allows programmers to control the computation of the per-pixel color value based on the per-pixel color, texture coordinates, etc., which are interpolated during the rasterization. Note that pixel shading can be regarded as a parallel process over all pixels belonging to the on-screen polygon.

Nevertheless, as mentioned in the introduction of this chapter, writing vertex programs and pixel/fragment shaders involves writing low-level assembly code. This is very inefficient, bug prone, and difficult to debug. To address this problem, researchers are developing various higher-level shading languages as supporting systems residing above the low-level vertex and pixel/fragment processors: The Stanford RTSL (real-time shading languages) system [140], VectorC system in Song Playstation 2 [34], HLSL (High-level Shader Language) [35] by Microsoft, and Cg (C for Graphics) [114, 63, 36] by NVidia are examples of these higher-level shading languages. Among all these, Cg is the most popular shading language since it is supported in the latest NVidia graphics board for PC’s.

Basically, Cg means C for graphics, and as its name implies Cg is a trimmed-down version of the C language. Following the philosophy of the C language, Cg is a hardware-oriented, general-purpose language; the syntax of Cg is similar to the syntax of C. Moreover, when compiling a Cg program, the Cg compiler will automatically generate the assembly code for the vertex and pixel processors, which greatly improves the efficiency for writing shading programs. Details on Cg programming can be found in [63, 36].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Power Homogeneous Coordinate (PHC)

3.1 Definition

In this section, we introduce power homogeneous coordinates (PHC), which provide a coordinate representation specifically tailored to handle the demands of scenes with extreme spatial scales. Similar in form to homogeneous coordinates, power homogeneous coordinates are four-tuples, \((x, y, z, s)\), representing the three-dimensional world point 

\((x, y, z) \times k^s,\)

where \(k\) is any positive exponent base, normally chosen to be 10 or 2.

The basic idea of the PHC representation is to extract the exponential scale out of the 3D coordinates into the 4th component so that positions of stars or galaxies at huge scales or sub-atomic particles at microscopic scales can be handled in a uniform fashion. In this way, we can have the advantage of effectively representing positions and vectors with excessively large or excessively small spatial scales by means of the 4th coordinate in PHC.

Distances. PHC analysis of a vertex begins by extracting the exponential scales out of distances. However, since distances and lengths are only one-dimensional, we do not need
3. Power Homogeneous Coordinate (PHC)

<table>
<thead>
<tr>
<th>Object</th>
<th>Powers of 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planck length</td>
<td>-35</td>
</tr>
<tr>
<td>A Proton</td>
<td>-15</td>
</tr>
<tr>
<td>Hydrogen atom</td>
<td>-10</td>
</tr>
<tr>
<td>Virus</td>
<td>-9</td>
</tr>
<tr>
<td>Cell</td>
<td>-5</td>
</tr>
<tr>
<td>Human</td>
<td>0</td>
</tr>
<tr>
<td>Earth</td>
<td>7</td>
</tr>
<tr>
<td>Solar system</td>
<td>14</td>
</tr>
<tr>
<td>Milky Way</td>
<td>21</td>
</tr>
<tr>
<td>Local galaxies</td>
<td>23</td>
</tr>
<tr>
<td>Super-cluster</td>
<td>25</td>
</tr>
<tr>
<td>Known Universe</td>
<td>27</td>
</tr>
</tbody>
</table>

Table 3.1: Base 10 logarithms of scales of typical objects in the physical Universe in units of meters. To convert to other common units, note that $1\,\text{au} = 1.50 \times 10^{11}\,\text{m}$, $1\,\text{ly} = 9.46 \times 10^{15}\,\text{m}$, and $1\,\text{pc} = 3.26\,\text{ly} = 3.08 \times 10^{16}\,\text{m}$, where $\text{au} =$ astronomical unit, $\text{ly} =$ light year, and $\text{pc} =$ parsec. (see table A.17 in appendices for detail)

Normalized Power Homogeneous Coordinates. If the $(x, y, z)$ part of a power homogeneous coordinate forms a unit vector, then $(x, y, z, s)$ is said to be a normalized power homogeneous coordinate, with the property that the distance between its three-dimensional position and the origin depends solely on $s$. We denote normalized power homogeneous coordinates using the hat sign, $\hat{p}$.

For extreme scales in astronomical environments, it is conventional to choose $k = 10$ for the base of power homogeneous coordinates. Here, if we choose units of meters, we see that $s = 0$ corresponds to the human scale of one meter, and $s = 7$ to about the diameter of the Earth ($1.3 \times 10^7\,\text{m}$). Table 3.1 gives a rough picture of some typical scales in these units.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
3.2 Analysis

In this section, we relate Power Homogeneous Coordinates (PHC) to Homogeneous Coordinates (HC) and then compare them mathematically.

Conversions

In general, power homogeneous coordinates \((x, y, z, s)\) are equivalent to the ordinary homogeneous coordinates \((x, y, z, w)\) with

\[
k^s = \frac{1}{w} \quad \text{where } k > 0.
\]

Case I: HC \(\rightarrow\) PHC

To convert homogeneous coordinates to power homogeneous coordinates, we use the following equation:

\[
s = -\log_k w \quad \text{(provided that } w > 0 \text{ and } k > 0). \]

We can see that \(w\) has to be positive, or else the logarithm of \(w\) will be undefined.

Case I(a): \(w < 0\)

In this case, we can rewrite the given homogeneous coordinate \((x, y, z, w)\) as \((-x, -y, -z, -w)\); note that changing the sign of all four components does not change the actual three-dimensional position the coordinate represents. Therefore, we can use the above formula to convert the homogeneous coordinate \((-x, -y, -z, -w)\) to the power homogeneous coordinate \((-x, -y, -z, -\log_k(-w))\) since \(-w > 0\).
3. Power Homogeneous Coordinate (PHC)

Case 1(b): $w = 0$

However, if $w = 0$, we cannot apply the same trick to make the 4th component of the homogeneous coordinate to be positive. As a result, we cannot convert homogeneous coordinates with $w = 0$ into power homogeneous coordinates. And in fact, in terms of homogeneous coordinate representation, $w = 0$ implies points at infinity; this means that power homogeneous coordinates cannot represent points at infinity in the usual fashion.

Case II: PHC $\rightarrow$ HC

The conversion from power homogeneous coordinates to homogeneous coordinates is more straightforward:

$$w = \frac{1}{k^s} \quad \text{(provided that } k > 0\text{).}$$

As long as $k > 0$, there exists a $w$ for any given $s$. It means that any power homogeneous coordinate $(x, y, z, s)$ can be converted into a homogeneous coordinate $(x, y, z, \frac{1}{k^s})$ corresponding to the same three-dimensional position.

Due to the fact that homogeneous coordinates make use of $w = 0$ to represent points at infinity, in the homogeneous coordinate space, if $x = y = z = w = 0$, zero divided by zero will end up with undefined values. Thus, $(0, 0, 0, 0)$ is a singular point in the homogeneous coordinate space. However, for any power homogeneous coordinate, if $k > 0$, $(x, y, z, s)$ will always map to a unique three-dimensional position for any given $x, y, z,$ or $s$.

As a result, to represent the origin in the three-dimensional space, we can have $x = y = z = 0$, and any value of $s$ for power homogeneous coordinates, but for homogeneous coordinates, we need to have $x = y = z = 0$ and $w \neq 0$. Moreover, except at the origin, any finite three-dimensional position is mapped exactly to a carefully normalized power homogeneous coordinate or homogeneous coordinate.
Linear VS Exponential

Another major difference between power homogeneous coordinates and homogeneous coordinates is that the 4th component of power homogeneous coordinates is exponential in the spatial scale, while the 4th component of homogeneous coordinates is linear in the spatial scale. We now analyze this property using the following example:

Consider a power homogeneous coordinate \((x, 0, 0, s)\) and a homogeneous coordinate \((x, 0, 0, w)\). Given the fact that any non-singular three-dimensional coordinate can be expressed as more than one power homogeneous coordinate or homogeneous coordinate, if we take \((1, 0, 0)\) and choose \(k = 10\), we can express \((1, 0, 0)\) as different homogeneous coordinates: \((1, 0, 0, 1), (2, 0, 0, 2), \ldots, \) etc. or different power homogeneous coordinates: \((1, 0, 0, 0), (10, 0, 0, -1), \ldots, \) etc. In this way, we can plot \(x\) against \(w\) and \(x\) against \(s\) for the two coordinates and obtain the following graphs (Figure 3.1).

![Expressing (1,0,0) as different HCs](image)

![Expressing (1,0,0) as different PHCs](image)

Figure 3.1: Expressing the three-dimensional point \((1, 0, 0)\) as homogeneous coordinates (left) and power homogeneous coordinates (right): plot \(w\) against \(x\) and plot \(s\) against \(x\).

From the figure, we can see that power homogeneous coordinates have the advantage
that the 4th coordinate responds exponentially with respect to \( x \), which represents the spatial distances; modeling large-scale distances, vectors, or positions become more effective and intuitive for the case of astronomical environments.

Summary: Differences between PHC and HC

The table below (Table 3.2) summarizes the differences between power homogeneous coordinates and homogeneous coordinates.

<table>
<thead>
<tr>
<th></th>
<th>Power Homogeneous Coordinates (PHC)</th>
<th>Homogeneous Coordinates (HC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>can represent points at infinity?</td>
<td>no</td>
<td>yes when ( w = 0 ) and at least one ( x, y, z \neq 0 )</td>
</tr>
<tr>
<td>any singularity?</td>
<td>no</td>
<td>yes, at ( (0, 0, 0, 0) )</td>
</tr>
<tr>
<td>w.r.t. spatial scale</td>
<td>exponential</td>
<td>linear</td>
</tr>
<tr>
<td>origin</td>
<td>more than one representation ( x = y = z = 0, ) ( \text{any} \ s )</td>
<td>more than one representation ( x = y = z = 0, ) ( \text{any} \ w \neq 0 )</td>
</tr>
</tbody>
</table>

Table 3.2: Differences between PHC and HC.

3.3 Arithmetic for Power Homogeneous Coordinates

In this section, we will look at some basic arithmetic operations involving power homogeneous coordinates. First of all, let \( p \), \( \hat{p}_1 \), and \( \hat{p}_2 \) be three power homogeneous coordinates
such that $\tilde{p}_1$ and $\tilde{p}_2$ are normalized power homogeneous coordinates,

\[
p = (x, y, z, s)
\]
\[
\tilde{p}_1 = (x_1, y_1, z_1, s_1)
\]
\[
\tilde{p}_2 = (x_2, y_2, z_2, s_2)
\]

and let $d = k^{s_2}$ be a scalar, expressed as an exponential scale, $s_2$, with base $k$.

1. Normalization

To normalize a power homogeneous coordinate, say $p$, we first compute its norm:

\[
\text{norm of } p = ||p|| = \sqrt{x^2 + y^2 + z^2}.
\]

If $||p||$ is non-zero, we can compute normalized $p$:

\[
\hat{p} = \left( \frac{x}{||p||}, \frac{y}{||p||}, \frac{z}{||p||}, s + \log_k ||p|| \right).
\]

If $||p||$ is zero, the value of $s$ becomes irrelevant since $p$ is not necessarily a normalized power homogeneous coordinate: The reason for this is that since $x = y = z = 0$, $p$ is located at the origin regardless the value of $s$. However, according to the definition of normalized power homogeneous coordinates, the distance between the origin and the three-dimensional position it represents depends solely on its $s$. In order to agree with this rule, we have to put $s$ to be $-\infty$ or $NaN$ in case we are normalizing $p$.

In the actual implementation, when deciding whether $||p||$ is zero or not, we compare it against a small quantity such as $1\epsilon - 7$. Also, in the actual implementation when normalizing a zero power homogeneous coordinate, instead of putting $s = -\infty$ or $NaN$, we may put $s = -\text{FLT}_\text{MAX}$ for ease of implementation.
2. Addition and Subtraction

Furthermore, given two normalized power homogeneous coordinates, \( \hat{p}_1 \) and \( \hat{p}_2 \), we can perform addition and subtraction between them using the following equations.

**Addition:**

\[
\hat{p}_1 + \hat{p}_2 = \begin{cases}
\hat{p}_1 & \text{if } s_1 \gg s_2 \\
\hat{p}_2 & \text{if } s_1 << s_2 \\
( x_1 + x_2 k^{\delta s} , y_1 + y_2 k^{\delta s} , z_1 + z_2 k^{\delta s} , s_1 ) & \text{otherwise}
\end{cases}
\]

**Subtraction:**

\[
\hat{p}_1 - \hat{p}_2 = \begin{cases}
\hat{p}_1 & \text{if } s_1 \gg s_2 \\
-\hat{p}_2 & \text{if } s_1 << s_2 \\
( x_1 - x_2 k^{\delta s} , y_1 - y_2 k^{\delta s} , z_1 - z_2 k^{\delta s} , s_1 ) & \text{otherwise}
\end{cases}
\]

where \( \delta s = s_2 - s_1 \).

The trick here is that since \( \hat{p}_1 \) and \( \hat{p}_2 \) are normalized, their norms depend solely on the 4th component only, i.e., \( s_1 \) and \( s_2 \). Thus, if the difference between \( s_1 \) and \( s_2 \) is too large, adding or subtracting something too small will not make any change to the larger value since the IEEE floating point representation has finite precision.

Typically, in case we are using a float representation, if \( |\delta s| \) is greater than or equal to 7, we can perform this trick and skip the computation: we know that float values only have 6 digits of precision (refer to FLT_DLG).

Moreover, it is worth noting that we are actually transforming the exponential base of \( \hat{p}_2 \) to \( s_1 \) and perform addition and subtraction at scale \( k^{s_1} \) in the above formulas. On the other hand, we can normalize \( p_1 \) to \( s_2 \), perform these operations at scale \( k^{s_2} \) or normalize both \( p_1 \) and \( p_2 \) to \( \frac{1}{2}(s_1 + s_2) \), and perform these operations at scale \( k^{\frac{1}{2}(s_1+s_2)} \).
3. Multiplication and Division

Note that there is no simple multiplication nor division between two power homogeneous coordinates, we only discuss multiplication and division between power homogeneous coordinates and scalar values:

\[ p_1 \cdot d = \left( x_1, y_1, z_1, s_1 + s_3 \right) \]
\[ \frac{p_1}{d} = \left( x_1, y_1, z_1, s_1 - s_3 \right) \]

Notice that, in terms of power homogeneous coordinate representation, multiplication and division become addition and subtraction on the 4th component.

4. Dot Product

When computing the dot product between two power homogeneous coordinates, we have to compute the \( s \) and \((x, y, z)\) parts separately:

\[ p_1 \cdot p_2 = \left( \left( x_1, y_1, z_1 \right) \cdot \left( x_2, y_2, z_2 \right) \right) \cdot \left( s_1 + s_2 \right) \]

**Ordinary 3D dot product**

\[ = \left( x_1 x_2 + y_1 y_2 + z_1 z_2 \right) \cdot \left( s_1 + s_2 \right) \]

Note that the result of this dot product operation is a scalar value.

5. Cross Product

For the cross product, we can carry out an ordinary cross product on the \((x, y, z)\) parts of the two power coordinates, and add up their exponential scales:

\[ p_1 \times p_2 = \left( \left( x_1, y_1, z_1 \right) \times \left( x_2, y_2, z_2 \right) \right) \cdot \left( s_1 + s_2 \right) \]

**Ordinary 3D cross product**

Note that the result of this cross product operation is also a power homogeneous coordinate.
6. Interpolation

Interpolation between two power homogeneous coordinates is an important tool for navigation across large-scale environments. Here, let $t \in [0, 1]$ be the parameter interpolating from coordinate $\hat{p}_i$ to $\hat{p}_2$. We can have the following types of interpolation:

**Linear:**

We linearly interpolate between the two three-dimensional positions:

$$p(t) = \hat{p}_1 (1 - t) + \hat{p}_2 t$$

$$= (x(t), y(t), z(t), s(t)),$$

where $x(t) = x_1 (1 - t) + x_2 k^{(s_2 - s_1)} t$,

$$y(t) = y_1 (1 - t) + y_2 k^{(s_2 - s_1)} t,$$

and $z(t) = z_1 (1 - t) + z_2 k^{(s_2 - s_1)} t$.

**Linear-Separate:**

We linearly interpolate separately between the $(x, y, z)$ portion and the $s$ part of the two power homogeneous coordinates. This interpolation method is useful when users want to travel across huge scales exponentially.

$$p(t) = (x(t), y(t), z(t), s(t)),$$

where $x(t) = x_1 (1 - t) + x_2 t$,

$$y(t) = y_1 (1 - t) + y_2 t,$$

$$z(t) = z_1 (1 - t) + z_2 t,$$

and $s(t) = s_1 (1 - t) + s_2 t$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Conclusion

A number of powerful multi-scale rendering techniques such as level-of-detail methods can be naturally supported by power homogeneous coordinates (PHC): we can easily make transitions from too-small-to-render objects (single-point images), to normal-scale three-dimensional models, to too-large-to-render or too-far-to-notice-the-depth-buffer situations. Moreover, we can make use of the standard OpenGL framework to express PHC's directly by using the 4th coordinate ($w$) in an extension to the OpenGL vertex subroutine `glVertex4[dfis]()`. We will demonstrate how to achieve these goals in the next chapter.
4

PHC Transformation: the Method

4.1 Introduction

In Section 2.1, we presented three problems in conventional graphics hardware that hinder large-scale visualization; in Chapter 3, we developed the concept of power homogeneous coordinates (PHC) for representing large spatial quantities. Now, in Chapters 4 and 5, we will respond to the three problems introduced in Chapter 2 and provide a unified solution for incorporating the concept of PHC into the transformation unit so as to facilitate large-scale visualization. The three problems are as follows:

- Floating point number representation has limited precision and range
- The transformation unit in conventional graphics hardware is limited by the internal floating point number precision
- Limited precision in depth buffering

The concept of power homogeneous coordinates (PHC) plays a core role in the techniques described in Chapters 4 and 5. As illustrated in Chapter 3, the basic idea of PHC is to
extract the exponential scale out of three-dimensional coordinates into the 4th component, \( s \), so that we gain extra range for the representation of large three-dimensional coordinates.

Following the same concept, in this Chapter, we will generalize the transformation mechanism we reviewed in Chapter 2 and extract exponential scales out of the transformation unit (both in the modelview and projection transformations) by incorporating the power homogeneous coordinate (PHC) representation into it. In this way, we gain extra precision for the transformation: we generalized the approach to PHC modelview transformations and PHC projection transformations in this chapter. Subsequently, in the next chapter, we will discuss how we implement these enhanced transformations within the OpenGL context; we will provide an integrated API (Application Programming Interface) as well as a small demonstration program to show how this works.

### 4.2 Using PHC in Transformation Unit

As reviewed in Chapter 2, the existing transformation unit carries out its computation from world to eye and clip spaces based on homogeneous coordinates: the world coordinates input to the modelview transformation, and eye coordinates output from the modelview and input to the projection transformation, as well as the clip coordinates after the projection transformation, are all homogeneous coordinates.

![Figure 4.1: PHC Modelview and Projection Transformations.](image)

Therefore, in order to incorporate power homogeneous coordinates (PHC) into this
existing transformation mechanism, our very first step is to re-derive the entire transformation mechanism based on power homogeneous coordinates (PHC). Figure 4.1 illustrates the situation: we represent world-space and eye-space coordinates in terms of PHC and reformulate the two transformations in terms of PHC. In the next two sections, we will present the two extended transformations in detail.

4.3 PHC Modelview Transformation

The Model: Embedded Exponential Scale of the Camera

In the conventional modelview transformation, we left-multiply the world-space coordinates, \( v_{\text{world}} \), by the \( 4 \times 4 \) modelview matrix, \( M_v \), and produce eye-space coordinates, \( v_{\text{eye}} \). Both \( v_{\text{world}} \) and \( v_{\text{eye}} \) are homogeneous coordinates. From Subsection 2.2, we have,

\[
\begin{align*}
v_{\text{eye}} & = M_v v_{\text{world}} \\
& = \begin{pmatrix} A & T \\ 0 & 1 \end{pmatrix} v_{\text{world}} \\
& = \begin{pmatrix} A (x_{\text{world}}, y_{\text{world}}, z_{\text{world}}^t + T, 1) \end{pmatrix}.
\end{align*}
\]

Since the modelview matrix is always invertible, we can reverse the transformation between the eye space and the world space. The situation here is like special relativity: in the eye space, we fix the camera at the origin and transform all objects from world to eye space (or camera space) through the modelview matrix. Inversely, instead of transforming all objects from world to eye space, we can transform the camera from eye to world space through the inverse modelview matrix and look at everything in the world space. In other words, we can transform the camera from eye to world and take a picture of objects in the world space; or we can transform all objects to the eye space and take the picture in the
eye space. The two pictures are exactly the same. This is a kind of duality between the eye space and the world space.

\[ v_{\text{world}} = M_v^{-1} v_{\text{eye}} \]

\[ = \begin{pmatrix} A^{-1} - A^{-1} T \\ 0 & 1 \end{pmatrix} v_{\text{world}} \]

\[ = \begin{pmatrix} A^{-1} (x_{\text{eye}}, y_{\text{eye}}, z_{\text{eye}})^t - A^{-1} T, 1 \end{pmatrix} \]

Furthermore, as described in Chapter 2, the camera (eye or viewpoint) is fixed at the origin of the eye space according to the OpenGL convention. Therefore, if we put \( v_{\text{eye}} = (0, 0, 0, 1) \), which is the camera position, into the above inverse equation, we can obtain the 3D location of the camera in the world space, which is \( -A^{-1}T \).

In this sense, we can clearly see that the modelview matrix is actually embedding the orientation as well as the position of the camera in the world space. As a result, to generalize this modelview matrix in agreement with the PHC language, we introduce an additional variable, \( s_{\text{mv}} \), to denote the exponential scale of the camera's position. In short, we generalize the \( T \) vector embedded in the modelview matrix as a power homogeneous coordinate, \((T_x, T_y, T_z, s_{\text{mv}})\), yielding

\[
\text{modelview translation: } T = (T_x, T_y, T_z)^t \times k^{s_{\text{mv}}},
\]

where \( k \) is the exponent base in PHC. As a rule of thumb, this \( s_{\text{mv}} \) can be understood as a scaling factor from the world space to the eye space: when \( s_{\text{mv}} \) increases, everything in the world coordinates shrinks accordingly when mapped to the eye space. We will give an example of varying \( s_{\text{mv}} \) and the PHC enhanced modelview transformation in Subsection 4.3.
Mathematics for PHC Modelview transformation

As proposed previously in Subsection 4.2, we would like to have power homogeneous coordinates (PHC) as the representation of three-dimensional coordinates for input and output through the PHC modelview transformation; here we define the input PHC as

\[
\tilde{\mathbf{v}}_{\text{world}} = (x_{\text{world}}, y_{\text{world}}, z_{\text{world}}) \times k^{\text{world}},
\]

where \( k \) is the exponent base in PHC.

We can now substitute the above two PHC representations, \( \tilde{\mathbf{v}}_{\text{world}} \) and \( T \) (from the previous section), into the modelview matrix computation, to yield

\[
\tilde{\mathbf{v}}_{\text{sys}} = |M_v| \cdot |\tilde{\mathbf{v}}_{\text{world}}| = \begin{vmatrix}
A \begin{pmatrix}
x_{\text{world}} \\
y_{\text{world}} \\
z_{\text{world}}
\end{pmatrix}
& k^{\text{world}} + \begin{pmatrix}
T_x \\
T_y \\
T_z
\end{pmatrix}
& k^{\text{mv}}, 1 
\end{vmatrix},
\]

where "\( \cdot \)" means the multiplication is not an ordinary matrix-vector multiplication, but is the generalized PHC transformation defined in the equation.

Note that we ignore the last row of \( M_v \) in the computation, assuming that it will produce the value one as the standard 4th homogeneous coordinate does (see assumptions in Subsection 5.2).

Furthermore, to ensure safety in the computation and avoid numeric problems, we first normalize \( A(x_{\text{world}}, y_{\text{world}}, z_{\text{world}})^T k^{\text{world}} \) and \( T \) into two normalized PHCs, say \( \hat{\mathbf{P}}_a \) and \( \hat{\mathbf{P}}_t \), where \( \hat{\mathbf{P}}_a = (x_a, y_a, z_a, s_a) \) and \( \hat{\mathbf{P}}_t = (x_t, y_t, z_t, s_t) \). In this way, we can make use of the addition formulae we devised in Section 3.3 to add up \( \hat{\mathbf{P}}_a \) and \( \hat{\mathbf{P}}_t \):

\[
\hat{\mathbf{P}}_a + \hat{\mathbf{P}}_t = \begin{cases}
\hat{\mathbf{P}}_a & \text{if } s_a >> s_t \\
\hat{\mathbf{P}}_t & \text{if } s_a << s_t \\
(x_a + x_t k^{\text{ds}}, y_a + y_t k^{\text{ds}}, z_a + z_t k^{\text{ds}}, s_a) & \text{otherwise}
\end{cases}
\]
where $\delta s = s_t - s_0$.

In addition, it is worth noting that we can pre-compute the normalization of $T = (T_x, T_y, T_z)^T \times k^{s_{mv}}$ for all subsequent PHC modelview computations (see above) unless the modelview matrix is modified.

At the end of the PHC modelview transformation, we denote the resultant eye space vertex $\vec{u}_{\text{eye}}$ in PHC form as

$$
\vec{u}_{\text{eye}} = (x_{\text{eye}}, y_{\text{eye}}, z_{\text{eye}}, s_{\text{eye}}),
$$

which is ready for the next transformation: the PHC projection transformation.

**Effects of $s_{mv}$**

In this subsection, we will present an example of varying $s_{mv}$ in an astronomical scene with $k$ set to 10; in practice, this will result in a powers-of-ten transition, which is a log-scale transition as opposed to linear scale transition.

Figures 4.2 demonstrates such an animation sequence. The camera (viewpoint or eye) starts out at a distance $10^{12}$ meters from the Sun (top-left picture in the figure), i.e., $s_{mv} = 12$ and $||T_x, T_y, T_z|| = 1$. At this moment, we are at the scale around the inner planets in our Solar system: Inner planets include Mercury, Venus, Earth, and Mars. Then, without changing the conventional 4 x 4 modelview matrix and locations of objects in the scene, we vary the exponential scale, $s_{mv}$, which is a scaling factor in the modelview translation in $T$. By increasing it to 13, the whole Solar system comes into our view: Outer planets include Jupiter, Saturn, Uranus, Neptune, and Pluto. Furthermore, when $s_{mv}$ reaches 14, the whole Solar system diminishes, and next it collapses into a small point at $s_{mv} = 15$. As we further increase $s_{mv}$ up to 16 ... 19, we can see that we arrive at the scale of nearby stars, and stars appear to move towards the center of our view as $s_{mv}$ further increases.
Figure 4.2: Effect of varying $s_{mv}$ results in Powers-of-ten transition.
This small animation sequence demonstrates the mathematical nature of $s_{mv}$: It is a scaling factor from the world space to the eye space that always keeps the modelview matrix representable at extreme scales. Thus, we never need to place the camera at extreme scales using only the plain $4 \times 4$ modelview matrix; we gain extra precision to specify the camera position by using $s_{mv}$.

### 4.4 PHC Projection Transformation

**The Idea: Depth Rescaling Method (DRM)**

As examined in Chapter 2, the functionality of projection is to map eye coordinates to clip coordinates so that we can decide whether a vertex is inside the viewing frustum or not. This is called *homogeneous clipping*. Furthermore, after perspective division, we can obtain the corresponding screen coordinates as well as depth values to be stored in the depth buffer.

This projection mechanism has been studied intensively in Sections 2.2 and 2.3: the relationship between the eye space $z$-value and the depth value in the depth buffer (Subsection 2.3). We learned from the analysis that we should not allow a large separation between the near and far planes; otherwise, we lose depth precision for objects close to the far plane and foul up the projection transformation, which depends on floating point numbers internally.

To get rid of this problem, we present the following observation:

*Instead of enlarging the separation between the near and far planes to include far-away objects at huge scale, we can shrink the eye-space distances to far-away objects so that they move into the finite region between the near and far planes.*
Figure 4.3: Depth Rescaling: By distorting vertices along their lines of sight uniformly, we can bring far-away objects into the viewing frustum.

Figure 4.3 illustrates the idea: given an object far beyond the far plane, if we shrink all its vertices along their lines of sight towards the eye point, the object will decrease in effective size and enter the viewing frustum. As a result, it becomes viewable at the camera without having to extend the far plane. At the same time, since all vertices are distorted uniformly along their lines of sight, we will not notice any difference at the camera since the camera is fixed at the origin, at which all lines of sight intersect. In summary,

*If we distort all eye-space coordinates along their lines of sight uniformly in the same manner, we can preserve their depth orders relative to the eye point and bring far-away objects into the viewing frustum for visualization.*

We call this technique the **Depth Rescaling Method** (DRM); the underlying mathematical model to distort eye-space coordinates is the **Depth Rescaling Model**, which will be discussed in detail in the next subsection.
The Model: Depth Rescaling Model

In our depth rescaling model, we begin by adding extra information in addition to the near and far planes. As illustrated in Figure 4.4, we provide two supplementary planes between the classical near and far planes: the near cutoff plane and the far cutoff plane. While this is sufficient for most practical purposes, there is no reason we could not support an arbitrary number of supplementary planes.

Figure 4.4: The Depth Rescaling Model.

The reason behind this projection model is that since the utilization of the depth buffer is strongly influenced by the ratio between near-far-plane distances, the depth buffer fails to work properly when the ratio is too large. With the two additional planes, we segment the ordinary viewing region into three sub-regions and map them onto the depth buffer individually. For the two outer regions, one between the near plane and the near cutoff plane, the other between the far plane and far cutoff plane, we map eye-space coordinates logarithmically to the front and rear parts of the depth buffer. These two regions are called the near...
safety region and the far safety region because they allow us to safely render objects that are too near or too far from the eye point for conventional depth buffering to function reliably. On the other hand, we map the middle region, the linear cutoff region, to the middle part of the depth buffer either linearly or using standard OpenGL hardware-supported NDC coordinates. Note that the depth coordinates in this middle region can still be enormous, even if the relative distance between objects at the limits of the \([\text{nearRatio}, \text{farRatio}]\) bounds are small; thus, while the mapping in PHC coordinates may be linear in \(z_{\text{NDC}}\), OpenGL may not be able to handle it directly, and the normalization of the range must be verified before choosing the scale of coordinates passed to the OpenGL pipeline. Using this projection model, the programmer has access to detailed controls allocating variable portions of the depth buffer for different scene regions, and allowing important objects to remain in the middle linear region for precision rendering.

To support effective computation with this model, the positions of the near plane, the far plane, and the two cutoff planes should be intrinsically defined using the exponential PHC scale format instead of the linear scale of the standard OpenGL environment; the PHC method is used throughout to represent vertices and to implement their transformations.

In Figure 4.4, we define \(10^s_{\text{near}}\) and \(10^s_{\text{far}}\) to be the distances to the near and far planes respectively. Also, we define \(10^s_{\text{near,cutoff}}\) and \(10^s_{\text{far,cutoff}}\) to be the distances from the eye (the origin of the eye space) to the cutoff planes for the near and far safety regions respectively. Again we set \(\text{base}\) or \(k\) equal to 10 in practice; these four planes are perpendicular to the eye-space \(z\)-axis, and \(s_{\text{near}} \leq s_{\text{near,cutoff}} \leq s_{\text{far,cutoff}} \leq s_{\text{far}}\). In addition, to map the two cutoff planes onto the depth buffer, we define \(\text{nearRatio}\) and \(\text{farRatio}\) with \(0.0 \leq \text{nearRatio} \leq \text{farRatio} \leq 1.0\) such that the near safety region, the linear cutoff region, and the far safety region are allocated to \([0.0, \text{nearRatio}]\), \([\text{nearRatio}, \text{farRatio}]\), and \([\text{farRatio}, 1.0]\) of the depth buffer respectively, assuming the conventional depth buffer range \([0, 1]\).
Note on Generalized Model: The Near/Linear/Far Model worked out in detail previously has the advantage that it is well-adapted to straightforward implementation in software, vertex shaders, and fairly simple hardware. However, our overall concept of Depth Rescaling for large-scale-range scene-rendering enhancements is more general than the explicit Near/Linear/Far Model. The Generalized Model, which includes the Near/Linear/Far Model as a special case, essentially takes the chosen representation of the projection depth for each object (typically in PHC for the practical cases we have implemented), and considers the family of scene objects as having fixed projections rendered into the screen, but arbitrarily movable depth buffer values rendered into the depth buffer.

Thus, one may perform arbitrary rescaling before depth buffering, depending on the program’s prior knowledge of scene depth ranges, and depending on the computational resources available to perform detailed computations identifying potential depth problems within groups of objects in the scene. A partial list of strategies available for the generalized model would then include the following:

- **No rescaling.** Use the ordinary depth values of vertices to access the depth buffer, effectively using only what we have termed the “linear” region.

- **Near/Linear/Far.** Identify a large-scale trouble region, and rescale into the area reserved for Far depth buffer; similarly, identify possible troubles for the Near region, and use one or both of the special rescaling regions to allow all objects to be rendered without anomalies.

- **Application-based scene groups.** Generalizing the concept of near, far, and linear, we can divide the depth buffer into an arbitrary set of regions, possibly mixing linear and rescaled regions arbitrarily. There can be a different layer of the depth buffer for each of a set of arbitrarily chosen groups of objects, and each layer can treat its group’s scaling in a different appropriate manner.
4. PHC Transformation: the Method

- **Recycling entire depth buffer for each portion of sorted scenes.** If reliably depth-sorted groups of scene objects are available in a given application, the depth buffer can even be cleared and reused at a new scale range as each group is rendered into the image buffer from back to front. Then a much larger numerical range of the depth buffer becomes available for the sorted subscenes, since parts of the scene no longer need to have a depth range reserved for them.

- **Optimal Partitions.** If complex scene management strategies are feasible, then the depth buffer can be treated as a statistically allocated resource for the whole range of scales, and the density of depth values to be rendered (determined however possible) is used to select, e.g., an equal-density partition of the resource, thus achieving maximal use of the precision available for a particular scene (or scene segment, if recycling is exploited).

**Mathematics for PHC Projection transformation**

In this subsection, we will detail the mathematics for implementing the depth rescaling method. Unlike the simple mathematical model we have for the PHC modelview transformation, the PHC projection transformation is much more complicated; it basically consists of four steps (Figure 4.5):

1. Z-normalization of the eye-space coordinate
2. Mapping the z-normalized exponential scale into the depth buffer based on the given Depth Rescaling Model
3. Mapping the $z_{NDC}$ inversely to eye space, $z_{eye}'$, based on the conventional $4 \times 4$ projection matrix
4. Rescale (distort) the eye-space coordinate by $z_{eye}'$
4. PHC Transformation: the Method

Figure 4.5: The PHC Projection Transformation.

Note that we employ the Near/Linear/Far depth rescaling model in step 2 to map the eye-space depth value to the depth value to be stored in the depth buffer; in fact, this step can be customized for different depth rescaling models accordingly. When we map \( z_{NDC} \) back to \( z_{eye} \) based on the conventional projection matrix, we need to know the type of projection (perspective or parallel) as well as the projection parameters embedded in the conventional \( 4 \times 4 \) projection matrix. We make use of the techniques we devised in Section 2.3 to complete this task.

1) Z-Normalization (\( \vec{v}_{eye} \rightarrow z'_{eye} \)). The end of the PHC Modelview transformation results in a power homogeneous coordinate, \( \vec{v}_{eye} \), which is in eye space. In order to know in which region (the near safety region, the linear cutoff region, etc) \( \vec{v}_{eye} \) is located, we must normalize \( \vec{v}_{eye} \) and compare its log distance from the origin (the eye) against the log distances to each separating plane in our near/linear/far model.

Since distances to these separating planes are measured along eye-space \( z \)-axis, we should not apply ordinary PHC normalization on \( \vec{v}_{eye} \) to unify (normalize) its first three components \((x_{eye}, y_{eye}, z_{eye})\). Instead, we re-normalize \( \vec{v}_{eye} \) \((x_{eye}, y_{eye}, z_{eye}, s_{eye})\) to \( \vec{v}'_{eye} \).
(x'_eye, y'_eye, z'_eye, s'_eye) so that,

\[
\begin{align*}
    z'_\text{eye} &= \pm 1 \quad \text{if } z_{\text{eye}} \neq 0, \\
    \max (|x'_{\text{eye}}|, |y'_{\text{eye}}|) &= +1 \quad \text{if } z_{\text{eye}} = 0 \text{ and either } x_{\text{eye}} \text{ or } y_{\text{eye}} \text{ is non-zero}, \\
    s'_\text{eye} &= -\infty \text{ (or } -\text{FLT}_\text{MAX}) \quad \text{if } x_{\text{eye}}, y_{\text{eye}}, \text{ and } z_{\text{eye}} \text{ are all zero}.
\end{align*}
\]

This specialized normalization method is called \textit{z-normalization} and the \textit{z}-distance of the resultant PHC coordinate depends solely on \(s'_\text{eye}\). Though it seems that it may not be completely necessary to \textit{z-normalize} a PHC if its \(z_{\text{eye}} = 0\), it is a necessary operation so that we can ensure correct clipping.

After the \textit{z-normalization}, we obtain \(\bar{v}'_{\text{eye}} = (x'_{\text{eye}}, y'_{\text{eye}}, z'_{\text{eye}}, s'_{\text{eye}})\), which represents the same three-dimensional coordinate as \(v_{\text{eye}}\) in the eye space, but with \(z'_{\text{eye}}\) normalized to \(-1, +1, \text{ or } 0\). We will split our discussion into two cases, one for perspective projection (Subsections A-II, A-III, and A-IV) and one for parallel projection (Subsections B-II and B-III) because the related mathematics is different.

[Case A: Perspective Projection]

\textit{A-II) Mapping } s'_{\text{eye}} \textit{into the Depth Buffer } (s'_{\text{eye}} \rightarrow z_{\text{NDC}}). \textit{Given the fact that the eye space } z\text{-distance depends solely on } s'_{\text{eye}}, \text{ we can compare } s'_{\text{eye}} \text{ against the distances to the planes in the Near/Linear/Far model, } s_{\text{near}}, s_{\text{near_cutoff}}, s_{\text{far_cutoff}}, \text{ and } s_{\text{far}}, \text{ to find out in which region } v'_{\text{eye}} \text{ lies, i.e.,}

\[
\bar{v}_{\text{eye}} \text{ is } \begin{cases} 
    \text{before the near plane} & \text{if } s'_{\text{eye}} \in [-\infty, s_{\text{near}}] \\
    \text{in the near safety region} & \text{if } s'_{\text{eye}} \in [s_{\text{near}}, s_{\text{near_cutoff}}] \\
    \text{in the linear cutoff region} & \text{if } s'_{\text{eye}} \in [s_{\text{near_cutoff}}, s_{\text{far_cutoff}}] \\
    \text{in the far safety region} & \text{if } s'_{\text{eye}} \in [s_{\text{far_cutoff}}, s_{\text{far}}] \\
    \text{beyond the far plane} & \text{if } s'_{\text{eye}} \in [s_{\text{far}}, +\infty]. 
\end{cases}
\]
Consequently, we can map $s'_{\text{eye}}$ to the depth buffer based on the depth rescaling model depicted in Figure 4.4. Assuming the depth buffer value $z_{\text{depth}}$ is in the range $[0, 1]$, we have,

$$z_{\text{depth}} = \frac{z_{\text{NDG}} + 1}{2} \begin{cases} < 0.0 & \text{if } s'_{\text{eye}} \in [-\infty, s_{\text{near}}] \\ [0.0, \text{nearRatio}] & \text{if } s'_{\text{eye}} \in [s_{\text{near}}, s_{\text{near cut off}}] \\ [\text{nearRatio}, \text{farRatio}] & \text{if } s'_{\text{eye}} \in [-\infty, s_{\text{near}}] \\ \text{nearRatio} & \text{if } s'_{\text{eye}} \in [s_{\text{near cut off}}, s_{\text{far cut off}}] \\ \text{farRatio} & \text{if } s'_{\text{eye}} \in [s_{\text{near cut off}}, s_{\text{far}}] \\ > 1.0 & \text{if } s'_{\text{eye}} \in [s_{\text{far}}, +\infty] \end{cases}$$

To reduce computational overhead, we combine the first two cases and the last two cases; thus, we have three cases left:

- **Case (1) - Inside or Before the Near Safety Region ($s'_{\text{eye}} < s_{\text{near cut off}}$)**

In this case, we interpolate $s'_{\text{eye}}$ in the eye space $\log(z)$ to find $z_{\text{depth}}$, which is in $[-\infty, \text{nearRatio}]$.

$$z_{\text{depth}} = \frac{s'_{\text{eye}} - s_{\text{near}}}{s_{\text{near cut off}} - s_{\text{near}}} \cdot \text{nearRatio}$$

- **Case (2) - The Linear Cutoff Region ($s_{\text{near cut off}} \leq s'_{\text{eye}} < s_{\text{far cut off}}$)**

In this region, the mapping is linear from $s'_{\text{eye}}$ to $[\text{nearRatio}, \text{farRatio}]$ such that,

$$z_{\text{depth}} = \text{nearRatio} + \frac{10^{s'_{\text{eye}}} - 10^{s_{\text{near cut off}}}}{10^{s_{\text{far cut off}}} - 10^{s_{\text{near cut off}}}} \cdot (\text{farRatio} - \text{nearRatio})$$
Since $s'_{\text{eye}}$, $s_{\text{near\_cutoff}}$, and $s_{\text{far\_cutoff}}$ can be arbitrary large, computing exponents of these values can pose precision problems during floating point number computations. We therefore re-arrange the above equation as

$$z_{\text{depth}} = \text{nearRatio} + \frac{10(s'_{\text{eye}} - s_{\text{near\_cutoff}}) - 1}{10(s_{\text{far\_cutoff}} - s_{\text{near\_cutoff}}) - 1} \cdot (\text{farRatio} - \text{nearRatio})$$

It is important to note that this linear cutoff region behaves like the ordinary OpenGL model; that is, we have to make sure that the two cutoff planes are not too far apart from each other: the value of $s_{\text{far\_cutoff}} - s_{\text{near\_cutoff}}$ should not be too large.

- **Case (3) - Inside or Beyond the Far Safety Region ($s'_{\text{eye}} \geq s_{\text{far\_cutoff}}$)**

Similar to case (1), we interpolate $s'_{\text{eye}}$ in the eye space $\log(z)$ to find $z_{\text{depth}}$, which is in $[\text{farRatio}, +\infty]$.

$$z_{\text{depth}} = \text{farRatio} + \frac{s'_{\text{eye}} - s_{\text{far\_cutoff}}}{s_{\text{far}} - s_{\text{far\_cutoff}}} \cdot (1 - \text{farRatio})$$

Finally, we convert $z_{\text{depth}}$ to $z_{\text{NDC}}$ using the following relation:

$$z_{\text{NDC}} = 2z_{\text{depth}} - 1.$$

Interestingly, careful readers may see that the mapping from $s'_{\text{eye}}$ to $z_{\text{depth}}$, say $h$, is continuous (though not differentiable), even at boundaries between the above three cases. In this way, we guarantee any value of $s'_{\text{eye}}$ is mapped to a continuous range of legal values of $z_{\text{depth}}$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
In addition, $h$ is strictly increasing so that,

$$
\text{if } s'_{\text{eye}1} < s'_{\text{eye}2},
\text{then } h(s'_{\text{eye}1}) < h(s'_{\text{eye}2}).
$$

This property preserves the depth ordering of eye-space coordinates after they are mapped to the depth buffer. Therefore, this guarantees correct visibility resolution. In general, these two properties are basic criteria necessary for the generalized model described in Subsection 4.4.

**A-III) Inverse Mapping:** \( z_{\text{NDC}} \rightarrow z_{\text{eye}} \). We have now transformed \( s'_{\text{eye}} \) to \( z_{\text{NDC}} \), so we can find \( z_{\text{NDC}} \) and \( y_{\text{NDC}} \) by applying ordinary projection to \( x_{\text{eye}} \) and \( y_{\text{eye}} \). This gives NDC coordinates \((x_{\text{NDC}}, y_{\text{NDC}}, z_{\text{NDC}})\). Although Blinn has shown that it is possible to perform clipping in NDC space, the homogeneous clipping built into OpenGL is performed in clip space instead of NDC space; in order to have reliable and correct clipping compatible with the OpenGL architecture, we have to supply eye or clip coordinates to OpenGL.

As a result, we need to map \( z_{\text{depth}} \) or \( z_{\text{NDC}} \) inversely back to the eye space so that we can fit our rescaling method into the OpenGL framework. Indeed, if we are allowed to modify the underlying graphics hardware or the OpenGL architecture, this step is not necessary and the overall calculation can be greatly simplified.

As shown in Figure 4.6, we need to map \( z_{\text{depth}} \) to \( z_{\text{eye}} \) and then rescale \( s'_{\text{eye}} \) (actual position) along the line of sight so that the actual position moves into the view frustum defined by the near and far planes specified by standard OpenGL/GLU commands such as \texttt{glOrtho}, \texttt{glFrustum}, \texttt{gluOrtho2D}, or \texttt{gluPerspective} -- that is the actual viewing frustum corresponding to the conventional \( 4 \times 4 \) projection matrix. To have proper rescaling, after we convert \( s'_{\text{eye}} \) to \( z_{\text{depth}} \) and \( z_{\text{NDC}} \) in the previous step, we make use of the \( z_{\text{eye}} \leftrightarrow z_{\text{NDC}} \) conversion derived in Subsection 2.3 to find the rescaled \( z_{\text{eye}} \) in the view frustum. In this way, we can...
find the rescaled position simply by multiplying the \(Z\)-normalized PHC, \((x'_{\text{eye}}, y'_{\text{eye}}, z'_{\text{eye}})\), by \(z_{\text{eye}}\) so the three-dimensional rescaled position is inside the actual view frustum.

A precautionary step is needed when the actual eye coordinate is further away than \(s_{\text{far}}\): when \(s'_{\text{eye}} > s_{\text{far}}\), \(z_{\text{NDC}}\) is greater than one. According to our study of NDC space for perspective projection, it is possible that \(z_{\text{NDC}}\) may go beyond the convergence point and cause mapping errors. We can see in Figure 2.3 (rightmost subfigure in the top row) that \(z_{\text{NDC}}\) at the convergence point does not map continuously back to eye space. Instead, this point is mapped to NDC space from both \(z_{\text{eye}} = +\infty\) and \(z_{\text{eye}} = -\infty\). Therefore, for perspective projection, we can find the rescaled \(z_{\text{eye}}\) using

\[
\bar{z}_{\text{eye}} = \begin{cases} 
\frac{4 f}{z_{\text{NDC}} + 4 f} & \text{if } s'_{\text{eye}} \leq s_{\text{far}} \\
2 z_{\text{NDC}} & \text{if } s'_{\text{eye}} > s_{\text{far}}
\end{cases}
\]
where $f$ is the distance to the actual far plane defined conventional $4 \times 4$ projection matrix. This will ensure a continuous inverse mapping.

**A-IV) Rescaling followed by Projection and Clipping.** Finally, we can obtain the rescaled eye coordinates in homogeneous coordinate form (for perspective projection) as

$$\hat{P}_{\text{eye}} = (x'_{\text{eye}}, y'_{\text{eye}}, z'_{\text{eye}}, 1).$$

Then, we can let the OpenGL kernel multiply the ordinary OpenGL projection matrix onto this homogeneous coordinate, $\hat{P}_{\text{eye}}$, to obtain the corresponding rescaled clip coordinates. After that, OpenGL will apply perspective division to obtain the NDC, which is the $z_{\text{NDC}}$ we computed before. Note that using this approach, we can properly rescale any coordinate along the line of sight, and thus, this method allows OpenGL to handle clipping correctly afterwards. Also, this approach avoids the need to handle complications in the projection matrix when shear is present.

---

**[Case B: Parallel Projection]**

**B-II) Mapping $s'_{\text{eye}}$ onto the Depth Buffer ($s'_{\text{eye}} \rightarrow z_{\text{NDC}}$).** The story for parallel projection is quite different. There is no convergence limit in the NDC space like that in perspective projection, and $z_{\text{eye}}$ is linear in $z_{\text{NDC}}$ (see Figure 2.3). However, since we are projecting all eye coordinates towards $z = -\infty$ (parallel projection) instead of the origin (perspective projection), the line of sight to a vertex position, say $\vec{v}_{\text{eye}}$, is parallel to the $z$-axis through $\vec{v}_{\text{eye}}$, instead of being on a line from the eye space origin to $\vec{v}_{\text{eye}}$. Therefore, to rescale the position of $\vec{v}_{\text{eye}}$, we have to rescale the position of $\vec{v}_{\text{eye}}$ along the $z$-parallel line through $\vec{v}_{\text{eye}}$; thus, we only rescale (distort) the $z$-coordinate of $\vec{v}_{\text{eye}}$.

The mechanism for depth rescaling with parallel projection is illustrated in Figure 4.7. In this case, the first thing to do is to map the $\log(z)$, i.e., $s'_{\text{eye}}$, of the actual position ($\vec{v}_{\text{eye}}$) to
4. PHC Transformation: the Method

Figure 4.7: Depth Rescaling (Parallel Projection).
the depth buffer: we have to compute the corresponding \( z_{\text{ND}} \). From the analysis section, we can see that in the NDC space (see Figure 2.3), the eye space origin \((z_{\text{eye}} = 0)\) is mapped to \(-\frac{f+n}{f-n}\). Since \(f > n, \frac{f+n}{f-n} > 1\) and so, the eye space origin is mapped to \(-\frac{f+n}{f-n}\), which is on the negative side of \(-1\) in the NDC space. Observing this, we find,

\[
\begin{align*}
z_{\text{ND}} \text{ maps to } & \quad \begin{cases} 
-\frac{f+n}{f-n}, & \text{if } z'_{\text{eye}} < s_{\text{near}} \\
1, & \text{otherwise}
\end{cases} \\
& \quad \text{if } s'_{\text{eye}} < s_{\text{near}}
\end{align*}
\]

As illustrated in Figure 4.8, if \(s'_{\text{eye}} \geq s_{\text{near}}\) (beyond the near plane), we can apply the mapping we used in perspective projection (Subsection 4.4) to map this range to \(x_{\text{depth}}\) and then \(z_{\text{ND}}\) in the NDC space \([-1, +\infty)\). On the other hand, for \(s'_{\text{eye}} < s_{\text{near}}\) (between the \(z_{\text{eye}} = 0\) and the near plane), we map it to \([-\frac{f+n}{f-n}, -1)\) (the shaded region in the figure), as follows:

\[
z_{\text{ND}} = \left(\frac{f+n}{f-n} - 1\right)(k'(s'_{\text{eye}} - s_{\text{near}}) - 1) - 1.
\]

So, if \(s'_{\text{eye}} = s_{\text{near}}\), then \(z_{\text{ND}} = -1\), and if \(s'_{\text{eye}} \rightarrow -\infty\), then \(z_{\text{ND}} \rightarrow -\frac{f+n}{f-n}\). Note that \((\frac{f+n}{f-n} - 1)\) is the distance in the range \([-\frac{f+n}{f-n}, -1]\) in the NDC space (shaded region in the figure).
4. **PHC Transformation: the Method**

Furthermore, we should take care with the sign of $z'_{\text{eye}}$. Notice that when $z'_{\text{eye}}$ is positive, the actual input vertex is behind the $z_{\text{eye}} = 0$ plane in eye-space, which is out of the viewing frustum and should not be visible. However, since $s_{\text{eye}}'$ does not tell us on which side of $z = 0$ the actual vertex is located, or even whether $z_{\text{eye}}' > 0$, it is possible that $z_{\text{NDC}}$ may map into the viewing region $[-1, +1]$ in the NDC space, which is wrong. To remedy this, as illustrated in Figure 4.8, if $z_{\text{eye}}' > 0$, we flip the computed $z_{\text{NDC}}$ about the value $z = -\frac{f+n}{f-n}$ in the NDC space.

\[ z_{\text{NDC}(\text{new})} = -z_{\text{NDC(\text{original})}} - 2\left(\frac{f+n}{f-n}\right) \quad \text{if} \quad z_{\text{eye}} > 0. \]

After this flipping, we can safely map $z$ in eye space one-to-one into NDC space, so that $z = 0$ in eye space is mapped to $z = -\frac{f+n}{f-n}$ in NDC space. This mapping satisfies the depth rescaling model. Moreover, allocating NDC space like this allows OpenGL to handle clipping across the near plane.

**B-III) Projection to Clip space.** So far, we have computed $z_{\text{NDC}}$ so that $[-1, 1]$ in the NDC space corresponds to the viewing region. Then, as for perspective projection, it follows that the next step we should do is to map it back to the eye space to find $z_{\text{eye}}$. However, the inverse mapping step is not necessary for parallel projection because parallel projection is linear from clip to NDC space, so clipping in NDC or clip is the same (see Figures 2.3 and 2.4).

For parallel projection, $w_{\text{clip}}$ is always one; thus, we always divide clip coordinates by one to go from clip to NDC space and thus, $z_{\text{NDC}}$ and $z_{\text{clip}}$ are equal. Furthermore, since $w_{\text{clip}}$ is always one, the homogeneous clipping is uniform for all $x_{\text{clip}}$, $y_{\text{clip}}$, and $z_{\text{clip}}$. In this way, we can fix the $z$ and $w$ components of the output clip coordinate to be the $z_{\text{NDC}}$ we just computed and 1 respectively. Note that this output is the final output directed to the OpenGL hardware for clipping, perspective division, etc. Because of this simplicity, we
have the advantage that we do not need to find the rescaled eye space distance, $z_{\text{eye}}$, and project the rescaled eye position to clip space as we did in perspective projection.

Finally, we need to fix $x_{\text{clip}}$ and $y_{\text{clip}}$. In parallel projection, since the line of sight is along $z$-dimension, the $x$ and $y$ dimensions stay fixed. Therefore, for $x_{\text{clip}}$ and $y_{\text{clip}}$, we just need to apply ordinary projection. Let $p_{\text{f}1}$ and $p_{\text{f}2}$ be the first and second rows of the projection matrix, i.e., $p_{\text{f}1}$ is $(M_p[0], M_p[4], M_p[8], M_p[12])$ and $p_{\text{f}2}$ is $(M_p[1], M_p[5], M_p[9], M_p[13])$ in the standard $4 \times 4$ $M_p$ we outlined in Subsection 2.3. Then, we can expand the $z$-normalized eye PHC, $\vec{v}_{\text{eye}} = (x'_{\text{eye}}, y'_{\text{eye}}, z'_{\text{eye}}, s'_{\text{eye}})$, as a three-dimensional coordinate and perform parallel projection,

$$
\begin{align*}
  x_{\text{clip}} &= p_{\text{f}1} \cdot (x'_{\text{eye}}, y'_{\text{eye}}, z'_{\text{eye}}) 10^{z'_{\text{eye}}}, 1) \\
  y_{\text{clip}} &= p_{\text{f}2} \cdot (x'_{\text{eye}}, y'_{\text{eye}}, z'_{\text{eye}}) 10^{z'_{\text{eye}}}, 1)
\end{align*}
$$

Nevertheless, a precautionary step must be taken in the above computation. Note from the equations that if $x'_{\text{eye}} 10^{z'_{\text{eye}}}$ is too large, much bigger than the viewing frustum, numerical error may occur in the expansion. Note that for parallel projection, the viewing frustum is a rectangular box with top, bottom, left, and right limits. To avoid the numerical problem, we can first check these limits and, if $x'_{\text{eye}} 10^{z'_{\text{eye}}}$ is too large compare to the box limit, we can rescale it to a closer position. Furthermore, we cannot simply ignore this vertex, since this vertex may connect to some other vertices inside the view frustum and produce a clipped segment. In particular, if the point is more than three orders of magnitude outside the viewing box limit, we move it in so it is not more than three to four orders of magnitude away – this is a kind of rescaling in the $x$-$y$ space. In this way, we can ensure clipping and vertex interpolation function normally.

Finally, we will discuss why we have to flip the NDC for parallel projection but not perspective projection. In perspective projection, the computed $z_{\text{NDC}}$ is not put into the output clip coordinate directly as in parallel projection; instead, we first find the corresponding $z_{\text{eye}}$ (inversely) from this $z_{\text{NDC}}$, and then we rescale the actual coordinate in eye space. It
is thus possible that two mirror eye-space coordinates, with the same \( s'_{\text{eye}} \), but on different sides of the origin in eye space can have the same \( z_{\text{NDC}} \). Following the inverse mapping in perspective projection, they can have the same rescaled \( z_{\text{eye}} \), but after multiplying this \( z_{\text{eye}} \) with \((x'_{\text{eye}}, y'_{\text{eye}}, z'_{\text{eye}})\), the two mirror coordinates remain opposite to one another because they have different values of \( z'_{\text{eye}} \) in eye space. However, for parallel projection, we need to flip \( z_{\text{NDC}} \) since we are doing the depth rescaling in NDC space without going back to eye space. Nevertheless, we can ignore the NDC flipping in parallel projection and follow the same tactic as the perspective projection. However, since eye and clip spaces are linear for parallel projection, going to the eye space and projecting back to NDC afterwards is redundant and therefore unnecessary.

In conclusion, most complicated mathematics described in this section results from the fact that we have to fit the PHC transformation approach into the OpenGL architecture and the underlying graphics hardware. Thus, after we use the depth rescaling model to compute the NDC depth value, \( z_{\text{NDC}} \), we need to map it back to the eye space so that we agree with the OpenGL architecture by providing clip coordinates for the hardware clipping. In the long run, if the OpenGL architecture or commodity graphics hardware becomes more flexible in the handling of the clipping operation, we can further simplify the calculations described in this section.
5

PHC Transformation: Implementation and Demonstration

5.1 Introduction

In the previous chapter, we went through the PHC transformation method in detail; we revised both the modelview and projection transformations and made them compatible with PHC. In this chapter, we introduce our library API, the PHC API, for realizing PHC transformation in the context of OpenGL. Two versions of the implementation, based on the traditional T&L pipeline and on the vertex programming pipeline, have been developed: this means that we can execute the computations devised in the previous chapter either on the CPU or the GPU. Here, we describe the demonstration program that we developed to illustrate the capabilities of the PHC API for large-scale visualization. In practice, we show in the demonstration program that we can effectively render and navigate among objects continuously from the scale of $10^{78}$ down to the scale of $10^{-80}$ in world space, which is not possible using any other conventional method.
5.2 Implementation: The PHC API

The purpose of the PHC library API is to realize and explore the capability of PHC transformations in the context of OpenGL. In particular, we would like to illustrate the practical details by adopting the community standard OpenGL graphics framework and by constructing subroutines in this API as a set of OpenGL "glExt" extensions that provide the implementation.

As mentioned in Chapter 2, the basic entry point for all OpenGL transformations is the glVertex call, through which all 3D geometry is passed to the rendering engine; glVertex4[] takes four parameters, \((x, y, z, w)\), in the form of a homogeneous coordinate. Thus, the very first goal of the PHC API is to provide analogous to glVertex function for specifying power homogeneous coordinates (PHC):

**Synopsis:** We accomplish this goal by defining a modified vertex procedure, e.g., of the form glVertex4f\((x, y, z, s)\), that allows us to place a polygon vertex at the three-dimensional position \((x, y, z) \times (\text{base})^s\) in such a way that programmers can render an individual scene without encountering depth buffering precision anomalies, problems with the near or far depth buffer limits, or rendering anomalies due to exceeding the machine precision limits of floating point numbers in vertex definitions and transformation matrices. This is realized by the glPHCVertex4f subroutine in the library API.

**Description of the PHC API**

Altogether, there are five sets of subroutines in this library API. The first three sets of subroutines control parameters in PHC transformation, e.g., the exponential scale, \(s_{mv}\), related to the modelview matrix, parameters in the depth rescaling model, etc. The last
two sets of subroutines are for rendering, including the glVertex4f-like functions. Detailed
descriptions of each set of subroutines are given below:

1. Control \( s_{nv} \) in PHC Modelview Transformation

// Set the current exponential scale of the translation component in the
// modelview matrix (scaling factor from the view-world to clip space)

// DEFAULT:
// - 0.0, i.e. no scaling on mat[12] - mat[14]
//
// INPUT:
// - smv (exponential scale of the translation component in the modelview matrix)
//
// NOTE:
// - Do not call between glPHC_Begin_EXT and glPHC_End_EXT

extern void glPHC_setModelViewScale_EXT (GLdouble smv);
extern GLdouble glPHC_getModelViewScale_EXT();

The above subroutines control the parameter \( s_{nv} \) in the PHC modelview transformation
(see Section 4.3 for details of \( s_{nv} \)).

2. Control parameters of the Depth Rescaling Model in PHC Projection

// glPHC_DensityRange_EXT (GLfloat sNear, GLfloat sFar)
//
// - Defines the density range (in log10 scale) mapped to the
// depth buffer such that depth values \( 10^{-sNear} \) and
// \( 10^{-sFar} \) can be regarded as mapping to the "zNear"
// and "zFar" defined by the projection matrix
5. PHC Transformation: Implementation and Demonstration

// e.g. If we call
// > glPerspective ( 60.0, 1.0, 0.1, 1000.0 ) ;
// to initialize the projection matrix and then call
// > glPHC_DepthRange_EXT ( 0.0, 30.0 ) ,
// we can re-scale actual depth values ranged [10^0.0,10^30.0] 
// to depth values ranged [0.1,1000.0] in non-decreasing order.

// DEFAULT:
// - sNear: 0.0
// - sFar: 30.0

// INPUT:
// - sNear is the log10(z) value at near plane from eye
// - sFar is the log10(z) value at far plane from eye

// NOTE:
// - Do not call between glPHC_Begin_EXT and glPHC_End_EXT

extern void glPHC_DepthRange_EXT ( GLfloat sNear, GLfloat sFar ) ,

// glPHC_DepthCutoff_EXT ( GLenum cutoff_op ) ;
// glPHC_DepthNearCutoff_EXT ( GLfloat sNearCutoff, GLfloat nearRatio ) ;
// glPHC_DepthFarCutoff_EXT ( GLfloat sFarCutoff, GLfloat farRatio ) ;

// - Define the near and far cutoff planes (linear range) in log-Scale and how we allocate these regions in the depth buffer, such that

// (a) Cutoff positions
// Case (1):
// If "cutoff_op" == GL_PHC_ABSOLUTE
// actual_sNearCutoff = sNearCutoff ;
// actual_sFarCutoff = sFarCutoff ;
// Case (2):
// If "cutoff_op" == GL_PHC_RELATIVE_SNEAR
// actual_sNearCutoff = sNearCutoff + sNear ;
// actual_sFarCutoff = sFarCutoff + sNear ;
// Case (3):
// If "cutoff_op" == GL_PHC_RELATIVE_SMV
// actual_sNearCutoff = sNearCutoff + smv ;
// actual_sFarCutoff = sFarCutoff + smv ;
// (b) Cutoff method
// Case (1):
// Depth value with log-scale within [sNear, actual_sNearCutoff]
// is mapped to the depth buffer range [0.0, nearRatio] in log scale.
// Case (2):
// Depth value with log-scale within [actual_sNearCutoff, actual_sFarCutoff]
// is mapped to the depth buffer range [nearRatio, farRatio] in ordinary
// linear scale.
// Case (3):
// Depth value with log-scale within [actual_sFarCutoff, sFar]
// is mapped to the depth buffer range [farRatio, 1.0] in log scale.
// provided the depth buffer ranges [0.0, 1.0].
// (c) No cutoff method (i.e. omit the linear region in the near/linear/far model)
// If GL_PHC_LINEAR_RANGE is disabled by glPHC_Disable_EXT,
// depth value with log-scale ranged [sNear, sFar]
// is mapped to the whole depth buffer range in log scale.
// DEFAULT:
// - cutoff_op is PHC_ABSOLUTE
// - sNearCutoff is 0.5
// - nearRatio is 0.2
// - sFarCutoff is 3.5
// - farRatio is 0.8
// INPUT:
// - "cutoff_op" can be
//   GL_PHC_ABSOLUTE
//   GL_PHC_RELATIVE_SHEAR
//   GL_PHC_RELATIVE_SMV
// - sNearCutoff defines the near cutoff plane
// - nearRatio defines the proportion of the depth buffer for logZ value
//   [sNear, actual_sNearCutoff]
// - sFarCutoff defines the far cutoff plane
// - farRatio defines the proportion of the depth buffer for logZ value
//   [sNear, actual_sFarCutoff]
// NOTE:
// - Make sure 0.0 <= nearRatio <= farRatio <= 1.0
// else rendering result is undefined.
5. PHC Transformation: Implementation and Demonstration

// Also, make sure 'sNear <= actual_sNearCutoff <= actual_sFarCutoff <= sFar',
// else rendering result is undefined.

// If nearRatio == farRatio or GL_PHC_LINEAR_RANGE is disabled,
// we map all depth values onto the depth buffer in log-scale order.
// If nearRatio == 0.0 and farRatio == 1.0,
// we map all depth values onto the depth buffer in ordinary linear scale.

// - MAKE SURE "sFarCutoff-sNearCutoff" is NOT TOO LARGE as the
// actual depth buffer precision is affected by this difference and
// we map depth values linearly in this range

// - Do not call between glPHC_Begin_EXT and glPHC_End_EXT

extern void glPHC_DepthCutoff_EXT ( GLenum cutoff_op ) ;
extern void glPHC_DepthNearCutoff_EXT ( GLfloat sNearCutoff, GLfloat nearRatio ) ;
extern void glPHC_DepthFarCutoff_EXT ( GLfloat sFarCutoff, GLfloat farRatio ) ;

Basically, these subroutines set up parameters in the Near/Linear/Far depth rescaling model we presented in Subsection 4.4. These parameters include $s_{near}$, $s_{near\_cutoff}$, $s_{far\_cutoff}$, $s_{far}$, nearRatio, and farRatio. The meanings of these parameters in the library API are exactly the same as those defined in the previous chapter. In addition, we provide the cutoff\_op option to position the two cutoff planes: relative to $s_{mv}$, relative to $s_{near}$, or at absolute positions.

3. Control PHC Status

// Enable and Disable PHC status
// - use of vertex shaders in the PHC computation (GL_PHC_SHADER)
// - use of cutoff planes in the PHC computation (GL_PHC_LINEAR_RANGE)

// DEFAULT:
// - GL_PHC_SHADER is disabled
// - GL_PHC_LINEAR_RANGE is enabled

// INPUT: 'cap' can be
// - GL_PHC_SHADER
// - GL_PHC_LINEAR_RANGE

// OUTPUT:
// - glPHC_Enable_EXT returns true on success
//   e.g. vertex program enabled successfully

// KEY:
// - Do not call between glPHC_Begin_EXT and glPHC_End_EXT

extern bool glPHC_Enable_EXT(GLenum cap);
extern void glPHC_Disable_EXT(GLenum cap);

// Check if vertex programming is supported in current system

// OUTPUT:
// - returns true or false

// NOTE:
// - Do not call between glPHC_Begin_EXT and glPHC_End_EXT

extern bool glPHC_IsShaderSupported();

// Allows programmers to provide hint about safety (accuracy) against
// efficiency in the PHC computation

// DEFAULT:
// - GL_PHC_SAFETY_FASTEST

// INPUT:
// - Hint can be
//   - GL_PHC_SAFETY_DONT_CARE
//   - GL_PHC_SAFETY_FASTEST (default)
//   - GL_PHC_SAFETY_NICEST

// NOTE:
5. PHC Transformation: Implementation and Demonstration

extern void g1PHC_Hint_EXT ( GLuint hint );

The above set of subroutines controls the PHC status:

1. Enable / Disable the use of vertex programming.

2. Enable / Disable the linear cutoff range in the depth rescaling model.

3. Provide the hint to trade off accuracy (safety) and efficiency in the PHC computation.

4. Begin and End a PHC Session

// NOTE:
// - For efficiency in PHC computation, some matrix-dependent
//   operations are carried out inside glPHC_Begin_EXT().
//   THEREFORE, DO NOT call any GL function that may alter the
//   modelview or projection matrix (i.e. glRotate, glTranslate,
//   glScale, etc.) after a glPHC session began.
// - Usually, we call glPHC_Begin_EXT() before glBegin() and
//   glPHC_End_EXT() after glEnd() to maintain a better
//   programming structure.
// - If glPHC_Begin_EXT was not called before glPHC_Vertex*,
//   glPHC_Vertex* work like glVertex* correspondingly and
//   the coordinate will become homogeneous coordinate, but
//   not power homogeneous coordinate, which would result in
//   unpredictable error.
// - In addition, since we are building this API outside the
//   OpenGL kernel, these glPHC subroutines cannot be stored in
5. **PHC Transformation: Implementation and Demonstration**

This pair of subroutines must enclose all conventional OpenGL "`glBegin-glEnd`" pairs. When `glPHC_Begin_EXT` is called, certain matrix-dependent operations will be carried out, e.g., normalizing the translation component in the modelview matrix, examining parameters and type of the projection matrix, etc. Therefore, altering the modelview or projection matrix inside a PHC session will cause serious errors in the PHC computation. In addition, vertex programs will be loaded into the graphics hardware for execution when the `GL_PHC_SHADER` option is enabled.

Note that the "`glBegin-glEnd`" pair defines the geometry in standard OpenGL; `glVertex` can only be called inside this pair (similarly for `glPHC_Vertex*`). Thus, we must have the correct calling sequence: all "`glBegin-glEnd`" pairs that use `glPHC_Vertex*` should be enclosed by the "`glPHC_Begin_EXT-glPHC_End_EXT`" pair (illustrated in the comment of the next subroutine below).

### 5. Vertex calling (replace `glVertex` in `glBegin-glEnd`)

```c
// CALL glPHC_Begin_EXT before glBegin and then call these
// functions in the following sequence:
// i.e. -> glPHC_Begin_EXT()
//     -> glBegin()
//     -> glPHC_Vertex*()
//     -> glEnd()
//     -> glBegin()
//     -> glPHC_Vertex*()
//     -> glEnd()
```
As noted in the synopsis described earlier in the chapter, we provide this specialized vertex subroutine to replace the `glVertex` function. This subroutine is tailor-made to accept power homogeneous coordinates instead of homogeneous coordinates if the corresponding `glBegin-glEnd` pair enclosing the `glPHC_Vertex*` subroutine is inside a `glPHC_Begin_EXT-glPHC_End_EXT` pair. In practice, the PHC computation is performed either inside this subroutine or inside a vertex program; a call to the standard `glVertex` is made at the end of the `glPHC_Vertex*` subroutine so that `glPHC_Vertex*` can pass control to the OpenGL kernel for further graphics operations. Detail about how vertex programming is carried out with this subroutine are discussed later in Subsection 5.2.

**Note on this API**

1. First, the safety (accuracy) of the computation is controlled by `glPHC_Hint_EXT`. This relates to the PHC modelview transformation described in Subsection 4.3. Here, we provide three levels of safety: *don't care*, *fastest*, and *nicest*. Basically, *nicest* implements the full detail we presented in Subsection 4.3 and computes $P_a + P_t$ based on the PHC addition formulae:

$$
\hat{P}_a + \hat{P}_t = \begin{cases} 
\hat{P}_a & \text{if } s_a \gg s_t \\
\hat{P}_t & \text{if } s_a \ll s_t \\
(x_a + x_t k^6 s, y_a + y_t k^6 s, z_a + z_t k^5 s, s_a) & \text{otherwise}
\end{cases}
$$
where \( \delta s = s_t - s_n \).

But for fastest, it ignores the three-case conditioning and computes \( P_a + P_t \) using only the equation below:

\[
P_a + P_t = \left((x_a, y_a, z_a) + (x_t, y_t, z_t)k^{a-t}\delta a, s_a\right).
\]

This may produce precision problems in the floating point computation if \( s_t \) (or \( s_{mv} \)) is large. Finally, for don't care, we expand the two PHCs, \( P_a \) and \( P_t \), and treat them as ordinary three-dimensional vectors,

\[
P_a + P_t = \left((x_a, y_a, z_a)k^{a}, (x_t, y_t, z_t)k^{t}, 0.0\right).
\]

As we can see, "don't care" is not faster than "fastest" in terms of the number of arithmetic operations, but it poses the greatest hazard in the computation (it can be regarded as disabling the PHC computation). Therefore, we put this option in the API for comparison.

2. Another feature of this API allows us to turn on or off the linear cutoff region by calling \texttt{glPHC.Enable.EXT} and \texttt{glPHC.Disable.EXT} with the corresponding argument \texttt{GL.PHC.LINEAR.RANGE}. When the cutoff region is turned on, the depth rescaling model reserves the linear mapping region so that vertices with \( \log(z) \) between \([s_{near.cutoff}, s_{far.cutoff}]\) are mapped linearly to the corresponding NDC coordinates (see Subsection 4.4).

However, when the linear cutoff region is turned off, the near safety region and the far safety region are merged together into one continuous region because they are both exponential in scale. Thus, the three conversion cases in Subsection 4.4 can be reduced to one single case.
3. The other feature is that we can enable or disable OpenGL vertex programs (shaders) by calling `glPHC_Enable_EXT` and `glPHC_Disable_EXT` with the corresponding argument `GL_PHC_SHADER`. We will show how to implement the PHC transformation with or without OpenGL vertex programs in Subsection 5.2.

4. The other useful feature of the API is the ability to reposition the linear cutoff region. Using the subroutine `glPHC_DepthCutoff_EXT`, we can arrange the near and far cutoff planes so that the input values `sNearCutoff` and `sFarCutoff` (arguments of `glPHC_DepthNearCutoff_EXT` and `glPHC_DepthFarCutoff_EXT`), have an absolute position relative to the near plane, or relative to the modelview translation scale (`s_mv`) (see the API for detail). This gives us to have more freedom to position the linear cutoff region.

**Assumptions in this Implementation**

Next we list the assumptions made by the PHC transformation method that are required for the API to run properly under OpenGL.

- **The Modelview Matrix**
  First, the modelview matrix set up by the programmer, say `M_v`, should consist of translation, scaling, and rotation, etc., without any projection components. Explicitly, the projection components, `M_v[3]`, `M_v[7]`, and `M_v[11]`, are assumed to be zero whereas `M_v[15]` is assumed to be one. In the implementation, values of `M_v[3]`, `M_v[7]`, `M_v[11]`, and `M_v[15]` in the input modelview matrix are ignored and assumed to be `(0, 0, 0, 1)`.

- **The Projection Matrix**
  Likewise, the projection matrix set up by the programmer, say `M_p`, should consist
of projection only, without any modelview translation, rotation, scaling, etc. For example, $M_p$ satisfies this requirement if it is set up using functions like `glOrtho`, `glFrustum`, `gluOrtho2D`, `gluPerspective`, etc. In short, the modelview matrix is solely for view-world transformations whereas the projection matrix is solely for projections.

- Distinguishing Perspective and Parallel Projections

Thirdly, we assume that we can use the method described in Subsection 2.3 to examine the projection matrix set up by the programmer to determine if the matrix represents a perspective or a parallel projection. Also, we assume that the projection matrix is not a null projection.

- Exponential Base of PHC

For the base in the exponential scale of PHC, we normally choose 10 as the base so that it is natural for most human applications. Nevertheless, this base can be changed by substituting different values into the original equations. However, for efficiency in the computation, this base is hardcoded internally and we do not provide function calls to change this base value.

**With/Without Vertex Programming**

**Implementation with vertex programs**

The introduction of the OpenGL vertex program (in OpenGL 1.3 or above) allows us to write low-level assembly-like programs (vertex programs) to replace the standard Transform & Lighting (T&L) unit in OpenGL. As illustrated in Figure 2.6 of Section 2.4, OpenGL vertex programs wrap around the standard OpenGL modelview and projection matrix multiplication operations; that is, after `glVertex` or other parallel OpenGL routines are called, `glVertex` invokes any existing binding for an OpenGL vertex program and passes
its world coordinate arguments as input. Then, the OpenGL vertex program performs its internal computation on the input world coordinate (together with its vertex attributes such as color, normal, etc.), computes the resultant homogeneous clip coordinate, and feeds it into the graphics hardware for clipping, perspective division, rasterization, and so on.

By implementing the PHC transformation with vertex programs, we are actually mapping the arithmetic in the PHC transformation into low-level assembly code. In the PHC transformation, there are basically four or five computational steps, depending whether it is a parallel or perspective projection:

1. PHC Modelview Transformation (Subsection 4.3),
2. Z-Normalization (Subsection 4.4 (I)),
3. Computing $z_{NDC}$ (Subsections 4.4 (A-II) and 4.4 (B-II)),
4a. Perspective projection: inverse mapping and rescale $\vec{v}_{eye}$ (Subsections 4.4 (A-III) and 4.4 (A-IV)),
4b. Parallel projection: ordinary projection (Subsection 4.4 (B-III)).

Thus, to make use of OpenGL vertex programs to carry out PHC transformations, we can map each of the steps to the vertex program language. Altogether, there are three different safety states: don’t care, fastest, or nicest; the linear cutoff region can be enabled or disabled, and there are two kinds of projections: perspective and parallel. We note that vertex program performance is greatly affected by the number of instructions in the coding. Instead of doing more computation inside the vertex program for all these choices, we create twelve variants of the vertex program for all the $3 \times 2 \times 2$ combinations of cases and bind the corresponding vertex program before calling glVertex (inside glPHC.Begin.EXT). In this way, we can reduce the number of instructions in each vertex program.

The twelve variants are:
When a user calls `glPHC_Begin_EXT` (see the API subsection), `glPHC_Begin_EXT` will select the appropriate vertex program from the twelve cases and bind the vertex program using `glBindProgramNV` (and load the vertex program by `glLoadProgramNV` if the vertex program has not been loaded previously). Subsequently, when the user call `glVertex` or other parallel OpenGL functions, the selected OpenGL vertex program will be executed to perform PHC Transformation properly. Note that if the user wants to make the programs compatible with the non-vertex-programming case, he/she should call `glPHC_Vertex*` instead of `glVertex`. In fact, `glPHC_Vertex*` encloses `glVertex` call: if the vertex program is enabled, `glPHC_Vertex*` will just call `glVertex` internally; otherwise, `glPHC_Vertex*` will
need to execute some computations internally before calling \texttt{glVertex} (see the next part of this subsection).

In addition, \texttt{glPHC\_Begin\_EXT} also tracks the modelview and projection matrices, carries out some pre-computations (relating to the matrices or the depth rescaling model), and stores the pre-computed results in the OpenGL vertex program registers before executing the vertex program. Details for vertex programming can be found in [66, 52]. In Appendix B, we give explicit listings of the implementations of \texttt{vp\_nicest\_cutoff.vp} and \texttt{vp\_fastest\_nocutoff\_parallel.vp} for reference. The vertex program \texttt{vp\_nicest\_cutoff.vp} specifies "nicest" as the safety mode, enables the linear cutoff region, and assumes perspective projection, whereas the vertex program \texttt{vp\_fastest\_nocutoff\_parallel.vp} specifies "fastest" as the safety mode, disables the linear cutoff region, and assumes parallel projection.

**Implementation without vertex programs**

Careful readers may notice that the names of functions for loading and for binding vertex programs end with suffix NV. NV stands for the company NVidia, which is one of the major vendor for PC graphics hardware. However, since these vendors target their products for the PC market, vertex programming is currently supported only on some advanced graphics boards for PCs. Therefore, in order to carry out the PHC transformation without vertex programs, we have to emulate vertex programs with standard OpenGL. As discussed in Subsection 2.4, if we can set the modelview matrix to be the identity, the input world coordinate will be the same as the eye coordinate after the modelview transformation. Likewise, if we set both the modelview and projection matrices to be identity matrices, we can input clip coordinates directly to \texttt{glVertex} as world coordinates.

Using this trick, if users disable vertex programs by using \texttt{glPHC\_Disable\_EXT}, or if vertex programming is not supported on the current graphics system, \texttt{glPHC\_Begin\_EXT}
will step into the non-vertex-programming track. This function first remembers the contents of both the modelview and projection matrices and loads identities into them. Then, when we call the extended API function \texttt{glPHC\_Vertex\*} instead of the bare \texttt{glVertex}, \texttt{glPHC\_Vertex\*} will perform the same four or five PHC transformation steps implemented in the corresponding vertex program and compute the homogeneous clip coordinate; at the end, \texttt{glPHC\_Vertex\*} will invoke \texttt{glVertex\*} with the resultant clip coordinates. Since both the modelview and projection matrices has been set to identities, the input homogeneous clip coordinates go directly to the clipping step of the rendering pipeline and the emulation is accomplished.

**Cautionary note on \texttt{glPushMatrix} and \texttt{glPopMatrix}**

Note that \texttt{glPHC\_Vertex} is programmed in the bare C language and so the actual PHC computation is carried out on the CPU rather than on the GPU in the graphics hardware. Finally, \texttt{glPHC\_End\_EXT} restores the matrices and ends the PHC session. Therefore, by wrapping \texttt{glVertex\*} inside \texttt{glPHC\_Vertex\*}, we can perform this trick and make the implementation work with or without vertex programming. In addition, it is important to note that we should avoid using \texttt{glPushMatrix} and \texttt{glPopMatrix} to remember matrix contents in the implementation. The situation is that OpenGL implements Modelview and Projection with stacks so that programmers can push or pop matrices to or from the corresponding stacks and OpenGL uses the topmost matrices in the two stacks to carry out vertex transformations. However, the number of available slots (depth) in these stacks depends on the vendor implementation: Typically, on a PC with an NVidia GeForce3 graphics board, the depth of projection and modelview stacks are only 4 and 32, respectively, while the depth of projection stack is only 2 for most SGI machines. As a result, if we use \texttt{glPushMatrix} and \texttt{glPopMatrix} in our API implementation, stack overflow may result and it is hard for users of this API to find out the error since the internal implementation of this API is hidden.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
5.3 Demonstration Programs

The Demonstration Program

**PHC Transformation Demonstration Program:** Figure 5.1 depicts the interface of the demonstration program illustrating how we perform PHC transformations and experiment with parameters in the library API interactively. This program is written in OpenGL with GLUT and GLUI interfaces; it demonstrates how we make use of the PHC library API to render objects in scenes with huge spatial scales. In the program, we can look at the divided near/cutoff/far regions of the depth buffer as defined in the depth rescaling model and clearly visualize how depth values are allocated in the depth buffer.

The right-hand side of the interface is the control panel. It contains five panels used to modify the PHC status, to control the scene, to animate the views, to measure frame rates, etc. The left-hand side contains three subwindows: the screen view (top left), the depth buffer view (top right), and the command manipulator window (bottom). The screen view at the top-left region shows the rendered view of the world scene itself; the depth buffer view (to the right) depicts the corresponding depth buffer as a unit cube with a green outline. The two red-outlined planes in the cube correspond to the near cutoff plane and the far cutoff plane separating the near safety region, the linear cutoff region, and the far safety region. The command manipulator view shows the PHC commands we are currently using to render the screen view. Since we cannot fit all the PHC commands into a single page in the command manipulator view, we split the commands into two pages (see the top and bottom pictures in the figure).
Figure 5.1: Interface of the Demonstration Program (Pages 1 and 2).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
**Depth Buffer Errors:** Figure 5.2 depicts the contents of the depth buffer when we render objects across large spatial scales simply by setting the $z_{far}$ to a huge value in bare OpenGL, using no depth-rescaling features (no PHC transformation). Figure 5.2(a) tells us the spatial scale by putting textured digits on polygons. In this case, we are rendering objects with depths from $10^{17}$ up to $10^{37}$. We can see in the depth buffer window that most of the objects cluster near the far plane; the depth buffer is thus inefficiently utilized and depth buffering anomalies appear in the screen view rendering due to limited precision in the depth buffer. Figure 5.2(b) shows similar depth buffer errors for a tiled texture: we can see incorrect texture assignments inside the red square.

Turning on the PHC transformation (depth-rescaling method) features corrects these problems. As illustrated in Figure 5.3, we can see that the depth buffer is much better utilized and the rendering anomalies are now absent. Moreover, comparing Figure 5.2(b) and Figure 5.3(b), we can see that the PHC transformation method can render additional scales of tiled textures (on the left side of the screen) because it can render objects with scale above $10^{38}$, which is the limit for ordinary single precision floating point computation supported in typical OpenGL implementations.

**Varying the Modelview $log(z)$ Scale:** Using the PHC modelview transformation, that is by varying the Modelview $log(z)$ scale ($s_{mv}$), we can simultaneously render objects across extreme spatial scale ranges, including microscopic scales, or even down to the Planck length, appearing in the viewing frustum. Figure 5.4(a-d) shows the rendering of the objects from $10^{76}$ down to $10^{-80}$ in world space. This is not possible using any conventional method.
Figure 5.2: Examples showing Depth Buffer errors for bare OpenGL at large scales.
Figure 5.3: Correcting the Depth Buffer errors using the depth rescaling technique.
Figure 5.4: Going from $10^{26}$ down to $10^{-80}$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
With/Without Linear Cutoff Region: Figure 5.5 shows the rendering results with and without linear cutoff region (spatial scale from $10^0$ up to $10^{30}$). Note that disabling the linear cutoff region maps objects using logarithmic distance units onto the viewing frustum. Although screen views of both figures 5.5 (a) and (b) look the same, the way the depth buffer is actually utilized is different.

(a) With Linear Cutoff Region.

(b) Without Linear Cutoff Region.

Figure 5.5: The use of the Linear Cutoff Region.
**Perspective and Parallel Projections:** Figure 5.6 shows the difference between the renderings of the world using perspective and parallel projections.

![Perspective Projection](image1.png)

(a) Perspective Projection.

![Parallel Projection](image2.png)

(b) Parallel Projection.

Figure 5.6: Rendering with different projections.
5. **PHC Transformation: Implementation and Demonstration**

### Results

To study the performance of the PHC library, we compare four attributes (refer to the API – Subsection 5.2):

- Disabling or Enabling of Vertex Programs (Shader),
- Disabling or Enabling of Linear Cutoff Region (Cutoff/Nocutoff),
- Safety status (Don’t Care, Fastest, or Nicest),
- Projection Type (Perspective or Parallel).

To understand the variations in experimental performance on different platforms supporting vertex program features, we carry out the experiments on three platforms: 1) GeForce4 440 Go (Mobile) Laptop with 1.66GHz CPU, 2) GeForce4 Ti 4400 Desktop with 1.5GHz, and 3) GeForce3 Ti 500 Desktop with 1.5GHz CPU.

In the experiment, we put in 858 vertices (glPHC_Vertex* calls) defining 284 texture-mapped polygons. These polygons range in scale and eye-distance from $10^{-30}$ up to $10^{30}$ in OpenGL world coordinates. To measure the frame rate for each case, we rendered 6400 frames ranging in scene scale from $10^{-30}$ up to $10^{30}$, and then averaged the total time to compute the average frame rates. The results are listed in the table below.
### Table 5.1: Average Frame Rates for the PHC/Depth Rescaling Library rendered with/without Vertex Programs (in frames per second).

<table>
<thead>
<tr>
<th></th>
<th>Disable Shader (CPU)</th>
<th>Enable Shader (GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Don't Care</td>
<td>Fastest</td>
</tr>
<tr>
<td><strong>Perspective</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perspective</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cutoff</td>
<td>559.33</td>
</tr>
<tr>
<td></td>
<td>nocutoff</td>
<td>635.51</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Parallel</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cutoff</td>
<td>484.06</td>
</tr>
<tr>
<td></td>
<td>nocutoff</td>
<td>549.79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>GeForce4 Ti 4400</strong></th>
<th>Disable Shader (CPU)</th>
<th>Enable Shader (GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Don't Care</td>
<td>Fastest</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Perspective</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cutoff</td>
<td>518.27</td>
</tr>
<tr>
<td></td>
<td>nocutoff</td>
<td>582.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Parallel</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cutoff</td>
<td>440.64</td>
</tr>
<tr>
<td></td>
<td>nocutoff</td>
<td>483.29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>GeForce3 Ti 500</strong></th>
<th>Disable Shader (CPU)</th>
<th>Enable Shader (GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Don't Care</td>
<td>Fastest</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Perspective</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cutoff</td>
<td>500.51</td>
</tr>
<tr>
<td></td>
<td>nocutoff</td>
<td>509.79</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Parallel</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cutoff</td>
<td>394.32</td>
</tr>
<tr>
<td></td>
<td>nocutoff</td>
<td>438.52</td>
</tr>
</tbody>
</table>
Observations and Discussions:

- First of all, if vertex program is disabled, we are running vertex program emulators (bare C in *glPHC.Vertex*) on the CPU instead of using the GPU on the GeForce graphics board. Thus, the notebook with the 1.66GHz CPU performs the fastest among the three experimental platforms.

- Secondly, if the vertex program is enabled, we can see that the *GeForce4 Ti 4400* runs the fastest since it is equipped with the more advanced GeForce technology (parallelized vertex programs in the graphics board, called *infinitieFX II Engine*) in 2002. On the other hand, the *GeForce4 440 Go (Mobile)* runs the slowest, and in fact the vertex shader hardware is generally a factor of five slower than the CPU, and thus has no benefit.

- We can see that for the *Ti* members of the GeForce family (the *GeForce4 Ti 4400* and the *GeForce4 Ti 500*), the vertex programs run much faster than the emulator. Therefore, running vertex programs on the GPU does help to speed up the performance. Unfortunately, the laptop *GeForce4 440 Go* vertex shader programs have a frame rate just 1/8 the frame rate on the other two *GeForce Ti* platforms.

- For perspective and parallel projections, if vertex programs are disabled, rendering with perspective projection is always faster. However, if vertex programs are enabled, rendering with parallel projection is faster for *GeForce4 440 Go* and the rendering speed is almost the same on the other two platforms.

- Disabling the linear cutoff region makes the program run faster because it simplifies the mapping from $s'_{eye}$ to $z_{NDC}$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
5.4 Conclusion

We have presented and implemented the PHC transformation and the Depth Rescaling Method combined with Power Homogeneous Coordinates in the context of the OpenGL architecture so that scenes containing extreme scales and extreme scale differences can be handled transparently by the programmer. We accomplished this by defining a complete library API. Here, a modified vertex procedure, glPHC_Vertex4f(x, y, z, s), allows programmers to place a polygon vertex at the three-dimensional position $(x, y, z) \times k^s$ while managing the relative object depths to support complete control and suppression of depth buffering precision anomalies, problems with the near or far depth buffer limits, and other rendering following from finite vertex precision and finite depth buffer precision.

In addition to realizing the Depth Rescaling Method design features in software as OpenGL extensions, we also showed how the PHC transformation can be implemented in vertex program extensions, and, by implication, directly running on graphics accelerator hardware.

So long as the relative depths of numerous features can be apportioned to distinct depths in the depth buffer, scale ranges of 100 orders of magnitude can be supported. When too many objects are close to one another in depth, it is still possible to have familiar depth buffer anomalies; it is up to the programmer, as always, to control for exceptionally challenging scene configurations; however, for many situations of interest, particularly the huge scales with sparse data encountered in virtual astronomy and “Powers of Ten” applications, the method can be configured to require little programmer attention.
6

Environment Caching Criterion

In Chapters 4 and 5, we gave an analysis based on power homogeneous coordinates to
derive PHC transformations as well as the depth rescaling method, so that we can correct
rendering problems posed by the graphics hardware when working with large-scale visual-
alization. Now, in Chapters 6 and 7, we focus on accelerating large-scale visualization;
again, we make use of the concept of power homogeneous coordinates and devise two cri-
teria: the environment caching criterion in Chapter 6 to cache faraway objects, and the
object disappearance criterion in Chapter 7 to cull objects that are too small.

6.1 Observation

We are motivated by the following simple observation (the Celestial sphere concept) in
developing the techniques in this chapter:

*Stars appear to be fixed on a big celestial sphere no matter what the time and
place of our observation from Earth.*

This is an ancient observation establishing the Celestial sphere (or the star ball) con-
cept, independently proposed by Aristotle in Greece as well as written in "hun tian shuo"
(淵天說) [152] in China (see Chapter 1). Although astronomers abandoned this concept after Friedrich Bessel discovered parallax of the star "61 Cygnus" in the constellation Cygnus, this concept is useful for accelerating large-scale rendering.

Figure 6.1 outlines Bessel's method for measuring stellar parallax. We note that parallax means the apparent displacement of an object as seen from two different points that are not on a line with the object, we see that there are essentially two important parameters in the method:

1. the resolution power of the telescope,

2. the spatial distance between the two observation locations when measuring the directions to the star.

In measuring parallax, we have to maximize the resolving power of the telescope so that we can maximize the precision for resolving angular displacement of stars. In addition, we should measure directions to the star at different times of the year, usually half a year apart; the reason for this is that since the Earth is orbiting around the Sun at a period of one year, this strategy maximizes the spatial distance between the two observation locations. Hence, the observed parallax can be maximized.

Figure 6.1: Looking at 61 Cygnus at different times of the year.
Nevertheless, it is important to note that Bessel’s goal was to observe stellar parallax by measuring the displacement of stars; he needed to maximize the above two parameters to ensure a better measurement. However, the target of this research work is different: we would like to find out the limiting condition of these parameters to determine when the parallax of distant objects cannot be observed. In this case, we can use constant images to represent distant objects because users navigating within a finite region far away from these objects cannot notice any aberration. We will study the geometric model and quantify this technique in the next section.

6.2 The Environment Caching Criterion

The Safety Region

Figure 6.2 depicts the situation when observing stellar parallax. When a camera moves to a new position, the direction to the star is changed accordingly. The angular displacement of the star resulting from the change in camera location is referred to as stellar parallax.

Figure 6.2: Parallax of a star as observed from two different camera locations.

To quantify this geometric model, we define:
10^L units to be the distance from the initial camera location to the star,
10^S units to be the distance between the two camera locations,
θ to be the parallax (the angular displacement).

Representing distances in this way agrees with the base-10 power homogeneous coordinate representation: we can easily obtain distances in this form by normalizing coordinates or vectors taking the form of PHC. Furthermore, if the two directions from the new camera location to the initial camera location and to the star are at right angles to each other, the parallax (angular displacement), θ, will be maximized at a given s.

\[
\sin(\theta_{\text{max}}) = \frac{10^S}{10^L} = 10^{S-L}.
\]

Therefore, taking the direction from the initial camera to the star as the reference direction, the angular displacement of the star will always be less than \(\theta_{\text{max}}\) if the camera
moves within $10^5$ units from the initial camera position. This region is called the safety region of the camera. Figure 6.4 shows the safety region. Furthermore, if the camera is free to move inside this safety region, maximum parallax is attained when the camera moves between tangential points P and Q, as depicted in the figure. Thus, the maximum parallax observable in this safety region is $2\theta_{\text{max}}$.

![Diagram of safety region](image)

Figure 6.4: The Safety Region.

In short, we have the following conclusion:

The maximum angular displacement of an object $10^L$ units away from the reference location is guaranteed to be smaller than $2\theta_{\text{max}}$ if the camera moves within a safety region of radius $10^5$ units.

**Angular size of a Pixel**

To solidify the safety region concept for computer graphics applications, we need to relate $2\theta_{\text{max}}$ to screen pixel units. In other words, we need to find out the angular size of
screen pixels, denoted as $\theta_{\text{pixel}}$. Interestingly, this $\theta_{\text{pixel}}$ is similar to the resolution power of telescopes.

![Diagram showing field of view and resolution relationship](image)

**Figure 6.5: Angular size of a Pixel, $\theta_{\text{pixel}}$.**

Figure 6.5 illustrates our approximation, if the resolution of the screen (window) is of size $w$ and the field-of-view of the virtual camera is $\text{fov}$, we can approximate the angular size of a pixel, $\theta_{\text{pixel}}$, by $\text{fov}/w$. Note that we can take $w$ to be the maximum of screen width and height in case screen width and height are different.

**The Criterion**

To build up the environment caching criterion, we can relate $2\theta_{\text{max}}$ to $\theta_{\text{pixel}}$:

*If* $2\theta_{\text{max}} < k \theta_{\text{pixel}}$, the object, which is $10^L$ units away, will not move more than $k$ pixel units on the screen if the camera is moving within the safety region.

In fact, the above statement is straightforward: Since $\theta_{\text{pixel}}$ is the angular size of a pixel, if the greatest possible parallax, $2\theta_{\text{max}}$, related to the safety region of size $10^S$ units, is smaller.
than $k\theta\text{pixel}$, the star, which is $10^L$ units away, will not move more than $k$ pixel units on the screen provided that the camera moves within the safety region.

From the previous subsection, we have,

$$\theta_{max} = \sin^{-1}(10^{S-L}) \cdot$$

Thus, we can expand and rearrange the inequality,

$$k \theta\text{pixel} > 2 \theta_{max}$$
$$= 2 \sin^{-1}(10^{S-L}) ,$$

$$\therefore S < L + \log_{10}(\sin\left(\frac{k \theta\text{pixel}}{2}\right)) .$$

Now, we define the following function,

$$\delta(k) = -\log_{10}\left(\sin\left(\frac{k \theta\text{pixel}}{2}\right)\right)$$
$$= -\log_{10}\left(\sin\left(\frac{k f\text{ov}}{2w}\right)\right) ,$$

for a given screen resolution, $w$, and a given field-of-view, $f\text{ov}$. Since $\theta\text{pixel}$ is always a small angle, taking sine and then log on small values results in negative numbers; therefore, putting the minus sign in the definition of $\delta(k)$ function ensures that the resultant value is always positive, which makes $\delta(k)$ more meaningful (we will see this shortly). Now, we can plug $\delta(k)$ into the criterion inequality above:

$$S + \delta(k) < L .$$

Note that this $\delta(k)$ tells us the limiting ratio of radii between the $10^S$ safety region and the distance to object. If the object (i.e., the star) is at a distance more than $10^{S+\delta(k)}$ from the center of the safety region, which is $10^S$ units, the object will not move more than $k$ pixel units on the screen.
Furthermore, we can generalize the above argument by saying that any object, which is at a distance greater than \(10^{S+\delta(k)}\) units from the center of the safety region, matches the criterion. This means that:

*Given a safety region, we can pre-render any object beyond \(10^{S+\delta(k)}\) units as a plain image. During the program runtime, if the camera is within the safety region, instead of rendering these objects in three-dimensions, we can replace them by the pre-rendered image at \(k\) pixel-unit accuracy.*

![Safety Region and Caching Boundary Diagram](image)

\[ S + \delta(k) < L \]

Figure 6.6: Safety region: \(10^S\) units and the Caching boundary: \(10^{S+\delta(k)}\) units.

The established inequality is called the **Environment Caching Criterion**. It allows us to cache faraway objects in the form of images at a certain pixel-unit accuracy. Figure 6.6 depicts the safety region (the \(10^S\) shell) and the corresponding caching boundary (the \(10^{S+\delta(k)}\) shell): according to the environment caching criterion, everything that is outside the caching boundary can be cached in the form of images. Note that \(10^{\delta(k)}\) is the ratio between the radii of the caching boundary and the safety region; this is reason why
we put the minus sign in the definition of $\delta(k)$. We will give more examples and present the corresponding rendering algorithm based on this criterion in the next section.

6.3 Analysis and Applications

Analyze the $\delta(k)$ function

In this subsection, we would like to analyze the $\delta(k)$ function since this function is the crucial quantity, telling us the range of objects that can be cached. In general, the smaller this $\delta(k)$, the more objects we can cache.

$$\delta(k) = -\log_{10} \left( \sin \left( \frac{k \text{fov}}{2w} \right) \right)$$

and

$$10^{\delta(k)} = \frac{\text{radius of caching boundary}}{\text{radius of safety region}}$$

Basically, there are three parameters in this function: $k$, $\text{fov}$, and $w$.

Varying $k$

In this case, we fix $w$ and $\text{fov}$, then we vary $k$ from 1 to 5,

<table>
<thead>
<tr>
<th>$k$</th>
<th>$w = 1024, \text{fov} = 90^\circ$</th>
<th>$w = 2048, \text{fov} = 90^\circ$</th>
<th>$w = 2048, \text{fov} = 45^\circ$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.115 (1303.8)</td>
<td>3.416 (2607.6)</td>
<td>3.717 (5215.2)</td>
</tr>
<tr>
<td>2</td>
<td>2.814 (651.9)</td>
<td>3.115 (1303.8)</td>
<td>3.416 (2607.6)</td>
</tr>
<tr>
<td>3</td>
<td>2.638 (434.6)</td>
<td>2.939 (869.2)</td>
<td>3.240 (1738.4)</td>
</tr>
<tr>
<td>4</td>
<td>2.513 (325.9)</td>
<td>2.814 (651.9)</td>
<td>3.115 (1303.8)</td>
</tr>
<tr>
<td>5</td>
<td>2.416 (260.8)</td>
<td>2.717 (521.5)</td>
<td>3.018 (1043.0)</td>
</tr>
</tbody>
</table>
6. Environment Caching Criterion

In the above table, note that values inside brackets correspond to \(10^{\delta(k)}\)'s. Looking at the columns, we can see that as \(k\) increases, \(\delta(k)\) decreases. The reason for this is that the larger the value of \(k\), the larger the tolerance we allowed for pixel-unit accuracy. When we allow a bigger parallax, we can cache more objects closer to the safety region. Consequently, the caching boundary is drawn closer to the safety region, i.e., \(\delta(k)\) decreases.

### Varying \(f_{ov}\) and \(w\)

Here, we fix \(k = 1\), then we vary \(f_{ov}\) from 30 to 150 degrees and \(w\) from 512 to 2560.

<table>
<thead>
<tr>
<th>(f_{ov})</th>
<th>(w = 512)</th>
<th>(w = 1024)</th>
<th>(w = 1536)</th>
<th>(w = 2048)</th>
<th>(w = 2560)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30°</td>
<td>3.291 (1955.7)</td>
<td>3.502 (3911.4)</td>
<td>3.766 (5867.1)</td>
<td>3.893 (7822.8)</td>
<td>3.990 (9778.5)</td>
</tr>
<tr>
<td>60°</td>
<td>2.990 (977.8)</td>
<td>3.291 (1955.7)</td>
<td>3.467 (2933.5)</td>
<td>3.592 (3911.4)</td>
<td>3.699 (4889.2)</td>
</tr>
<tr>
<td>90°</td>
<td>2.814 (651.9)</td>
<td>3.115 (1303.8)</td>
<td>3.291 (1955.7)</td>
<td>3.416 (2607.6)</td>
<td>3.513 (3259.5)</td>
</tr>
<tr>
<td>120°</td>
<td>2.689 (488.9)</td>
<td>2.990 (977.8)</td>
<td>3.166 (1466.8)</td>
<td>3.291 (1955.7)</td>
<td>3.388 (2444.6)</td>
</tr>
<tr>
<td>150°</td>
<td>2.592 (391.1)</td>
<td>2.893 (782.3)</td>
<td>3.069 (1173.4)</td>
<td>3.194 (1564.6)</td>
<td>3.291 (1955.7)</td>
</tr>
</tbody>
</table>

By definition, we have

\[
\theta_{\text{pixel}} = \frac{f_{ov}}{w}.
\]

Thus, if we decrease \(f_{ov}\) or increase \(w\), \(\theta_{\text{pixel}}\) will decrease. Noting that a decrease in \(\theta_{\text{pixel}}\) is equivalent to an increase in pixel precision, so, contrary to the argument in the previous discussion, we have to push the caching boundary (the \(10^{\delta+\theta(1)}\) shell) further away from the safety region, i.e., \(\delta(1)\) becomes larger.
Examples

Example 1: The CAVE at Indiana University, Bloomington.

In this example, we compute the angular size of a pixel, $\theta_{\text{pixel}}$, for the CAVE setting at Indiana University, Bloomington; Figure 6.7 shows the three-dimensional model of this CAVE. It has four walls: front, left, right, and bottom; and there are four corresponding projectors illuminating the back side of each semi-transparent wall so that the user inside the CAVE has an immersive experience inside the virtual environment. The resolution of all projectors in the CAVE is $1024 \times 1024$.

Since the field-of-view for each wall is $90^\circ$ (or $\pi/2$ radians), we have to render the virtual environment at this fixed field-of-view value so that when these rendered images are projected onto the walls, the user inside the CAVE has the right perspective for the virtual environment. In addition, the screen resolution, $w$, is $1024$. Therefore, we have

$$\theta_{\text{pixel}} = \frac{f_{\text{ov}}}{w} = \frac{\pi}{2048}. $$
Example 2: Taking pictures using a Digital Camera.

Given a digital camera with $w = 1024$ (resolution) and $f_{ov} = 90$ degrees (field-of-view), by taking $k = 1$, we can look up $\delta(k)$ the second table in Subsection 6.3: We have $\delta(1) = 3.115$ and $10^{3.115} = 1303.8$. Therefore, the ratio of radii between the safety region and the caching boundary is 1303.8.

This means that, given a building more than 1303.8 meters from the current camera position, the building will not move more than one pixel in the camera image if we move the camera within a circle of 1-meter radius. Note that we ignore atmospheric effects at the Earth’s surface.

Example 3: Looking at the nearest star: the Centaurus Triple Star system.

Figure 6.9: Looking at the nearest star, the Centaurus Triple Star system.
This is a virtual astronomy example: we are navigating inside the Solar system. Assuming that we are in the CAVE, we have $\theta_{\text{pixel}} = \frac{\pi}{2648}$ and $\delta(1) = 3.115$ (take $k = 1$) from the previous two examples.

On the other hand, the nearest star is the Centaurus Triple Star system (consisting of stars Rigel Kentaurus, Alpha Centauri A, Alpha Centauri B, and Proxima Centauri Alpha Centauri C) at about 4.39 light year from Earth; thus, by converting this value to logarithmic scale in meters, we have $10^{16.62}$ meters. Note that one light year is about $9.46 \times 10^{15}$ meters.

Since the Solar system diameter is around $10^{13}$ meters, we can compute the $\log_{10}$ of its radius, $S = \log_{10}(10^{13}/2) = 12.7$. Since $\delta(1) = 3.115$, this agrees with the environment caching criterion.

$$12.7 + 3.115 < 16.62$$
$$S + \delta(1) < L.$$ 

Thus all stars, including the nearest star system, are outside the caching boundary, and we can guarantee that all stars will not move more than one pixel on the CAVE wall if we are navigating within our Solar system.

The Rendering Algorithm: Environment Caching

Pre-rendering: The result of the previous example leads to the use of an environment map to cache everything outside the caching boundary, e.g., stars, when we are navigating inside the Solar system — this method is called Environment Caching. The first step of this method is to pre-render the environment and to project the rendering onto the six surfaces of a cube. To do this, we can fix the camera at the center of the safety region, and render all
objects outside the caching boundary (stars, galaxies, etc.) six times with a 90-degree field-of-view. During the six-pass rendering, we point the camera in six different directions: top, bottom, left, right, front, and back. In this way, we cached the environment as six images in the form of a cube; such a mapping is called a *cubemap* in the computer graphics literature, usually used for texture mapping.

![An unfolded cubemap](image1.png)  
(a) An unfolded cubemap

![A cubemap is a textured box centered at the viewpoint](image2.png)  
(b) A cubemap is a textured box centered at the viewpoint

Figure 6.10: Example cubemap caching stars and the Milky Way.

Figure 6.10 depicts the pre-rendering result: the left hand side of the figure shows an unfolded cubemap caching stars and the Milky Way galaxy. The six square images can be folded together to form a cube depicted on the right hand side of the figure.

**Program Runtime:** After caching the environment as a cubemap, if we are navigating within the Solar system during the program runtime, we perform the following steps:

Step 1: Render the cubemap as a textured box with

- depth buffer disabled;
6. Environment Caching Criterion

- the viewing position at the center of the cubemap box.

Step 2: Render models within the Solar system as usual.

In the above procedure, it is important to note that we have to disable depth buffering during the cubemap rendering so that the cubemap geometry is not marked on the depth buffer; otherwise, it may block the other objects to be drawn in step 2. Essentially, this is equivalent to putting the geometry at the far clipping plane, \( z = 1 \) in the depth buffer), where the most distant viewable objects are located. In addition, by centering the cubemap at the viewing position, no matter where the viewing position moves to, the user will have the illusion that objects on the cached images of the cubemap are far away from the navigation region. In addition, because of the caching criterion, we can guarantee the rendering correctness in terms of pixel-unit accuracy.

6.4 Conclusion

In conclusion, by using the environment caching criterion and the corresponding rendering algorithm, we can pre-render anything outside the caching boundary and quickly display the cubemap during the program runtime without actually rendering the cached objects in three-dimensions. This technique greatly increases the rendering performance when we are working with large-scale environments.

- Although this rendering method greatly accelerates the performance, we are actually trading off performance for memory. Typically, a cubemap of face resolution \( 1024 \times 1024 \) takes up \( 6 \times 1024 \times 1024 \times 3 \) bytes because it has six faces and each image pixel takes three bytes of red, green, and blue. Altogether, such a cubemap has size 18 MB, which is barely manageable on some conventional PCs.
6. Environment Caching Criterion

- On the other hand, it is interesting to note that in addition to cubes, we can use any other geometry that encloses the whole $360^\circ \times 180^\circ$ sky, as the intermediate caching surface, e.g., spheres.

- Another issue is that the cubemap resolution should be comparable to the screen resolution; otherwise, if the cubemap resolution is too low, cached objects may look too blurry and accuracy is lost. In the CAVE demonstration, in which we cached stars, we chose the face resolution of the cubemap to be (at least) $1024 \times 1024$, which is compatible to the CAVE screen resolution.

- Furthermore, the environment caching technique can be used together with the depth rescaling method. For instance, in the Solar system navigation example, we can turn on the depth rescaling model when we pre-render faraway objects such as stars onto the cubemap. Afterwards, when we are using the cubemap to substitute for star rendering, we can turn off the depth rescaling model and fully utilize the depth buffer for the Solar system, since we no longer need to render objects beyond the Solar system.

- Lastly, we can employ a hierarchical strategy: one cubemap for nearby stars can have a range from $10^{16}$ meters up to $10^{19}$ meters, another cubemap for the Milky Way galaxy can have a scale of about $10^{21}$ meters, and yet another cubemap can be used for galaxies beyond the Milky Way. While we are inside the Solar system, we can render these cubemaps from back to front with alpha blending (transparency). Then, when we move to the stellar scale, we can fade in three-dimensional stars, fade out the stars' cubemap, and still have the benefit of using the other larger-scale cubemaps. As a result, we never need to render everything expensively in three-dimensions. Nevertheless, using this hierarchical approach, texture memory consumption and I/O bandwidth for texture paging are currently very large, which makes it hard for conventional PCs to handle.
Object Disappearance Criterion

In Chapter 6, we presented the environment caching criterion that accelerates large-scale rendering by caching faraway objects on cubemaps. In this chapter, we will look at another criterion that accelerates large-scale rendering. It is called the Object Disappearance Criterion, which allows us to cull objects that are too small to be viewable at the camera image resolution. Again, we apply the concept of power homogeneous coordinates and use exponents to represent distances just as we did in the previous chapter. Since the criterion is similar to the environment caching criterion, readers may find the derivation familiar.

7.1 Observation

Like the criterion we developed in the previous chapter, the object disappearance criterion starts from a simple observation:

*Since the enormous Universe is full of empty space, most objects are too small to be visible most of the time; instead of rendering them all the time, we can ignore them when they are too small to be seen.*
Indeed, this criterion helps us to address the inadequacy of the previous criterion. With the environment caching criterion, while we are moving within the Solar system, we can ignore the three-dimensional renderings of stars and galaxies by replacing these objects with cubemaps. However, when we move to the stellar scale, we still have to render stars fully in three-dimensions. We cannot ignore their renderings all the time. Nevertheless, by the time we reach the stellar scale, the Solar system becomes too small to be visible. Thus, after the environment caching criterion is violated, we can make use of the object disappearance criterion to ignore the rendering of the Solar system. Similarly, when we reach the galactic scale, nearby stars become too small to be viewable; likewise, we can ignore their renderings. In the next section, we will present the mathematical derivation of the object disappearance criterion.

7.2 The Object Disappearance Criterion

Visual angles of Objects

Before we derive the criterion, we need to know the size of objects perceived at the camera. Basically, this is the term “visual angle” we learned from high school physics:

Visual angle is the angle subtended by an object at the eye (or camera),

and we denote the visual angle of an object as $\theta_{\text{object}}$. Figure 7.1 depicts how we measure the visual angle of the Earth subtended at the camera; in this example, the camera is located at a distance $10L$ units from the Earth, whose radius is $10R$ units. Then, by extracting the geometry in Figure 7.1, we obtain Figure 7.2.
Figure 7.1: Measuring the Visual Angle, $\theta_{\text{object}}$, of the Earth subtended at the camera.

Figure 7.2: Geometric model for Figure 7.1.
Hence, we can use simple geometry to derive the equation for $\theta_{\text{object}}$:

$$\sin \left( \frac{\theta_{\text{object}}}{2} \right) = \frac{10^R}{10^L} = 10^{R-L},$$

$$\theta_{\text{object}} = 2 \sin^{-1} \left( 10^{R-L} \right).$$

With this equation, we are ready to make up the object disappearance criterion.

**The Criterion**

In the previous chapter, we relate maximum possible parallax, $2\theta_{\max}$, to the pixel resolution, $\theta_{\text{pixel}}$, and derive the caching criterion. Now, we take a similar step by relating the visual angle, $\theta_{\text{object}}$, to the pixel resolution, $\theta_{\text{pixel}}$, and derive the object disappearance criterion. Note that $\theta_{\text{pixel}}$ is the same $\theta_{\text{pixel}}$ defined in Subsection 6.2 of the previous chapter. Here, it is interesting to see that

if $\theta_{\text{object}} < k \theta_{\text{pixel}}$, the object, whose radius is $10^R$ units, will have size less than $k$ pixel units on the screen if the camera is more than $10^L$ units away from it.

The derivation of the above statement is straightforward by making comparison in the angular domain. Likewise, we can plug in the equation for $\theta_{\text{object}}$,

$$\theta_{\text{object}} = 2 \sin^{-1} \left( 10^{R-L} \right),$$

and obtain the following inequality:

$$k \theta_{\text{pixel}} > \theta_{\text{object}} = 2 \sin^{-1} \left( 10^{R-L} \right),$$

$$: R < L + \log_{10} \left( \sin \left( \frac{k \theta_{\text{pixel}}}{2} \right) \right).$$
Fortunately, since we use the same definition for pixel resolution, $\theta_{\text{pixel}}$, we can reuse the $\delta(k)$ function defined in the previous chapter: by putting $\delta(k)$ into the inequality above, we obtain the criterion:

$$R + \delta(k) < L,$$

which looks similar to the environment caching criterion we devised in the previous chapter. We will explore their similarity later in this chapter. In addition, this $\delta(k)$ tells us the limiting ratio between the camera-object distance and the object radius:

*If the camera is at a distance more than $10R + \delta(k)$ units from the object center, whose radius is $10R$ units, the object will have size less than $k$ pixel units on the camera screen.*

The established inequality is called the **Object Disappearance Criterion**, which allows us to cull small objects quantitatively at a certain pixel unit accuracy.

![Figure 7.3](image.png)

**Figure 7.3:** Object radius: $10R$ units and the Culling boundary: $10R + \delta(k)$ units.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 7.3 depicts the object (radius: $10^{R}$) and the corresponding culling boundary (the $10^{R+δ(k)}$ shell): according to the object disappearance criterion, everything that is inside the $10^{R}$-unit object region can be culled. Note that $10^{R(k)}$ is the ratio between the culling boundary radius and the object radius. We will give some examples of this criterion in the next section.

### 7.3 Examples

We now give two examples: a daily life example and a virtual astronomy example.

**Example 1: A Daily Life Example.**

Given a digital camera with field-of-view 60 degrees (or $\pi/6$ radians), and resolution $1024 \times 1024$, taking $k = 1$, we can look up $δ(1)$ from the table in the previous chapter, so $δ(1) = 3.291$ and $10^{3.291} = 1955.7$.

![Digital Camera](image)

---

**Example 2: A Virtual Astronomy Example.**

As depicted in Figure 7.4, if a cabinet of size two meters (radius: one meter) is at a distance 1955.7 meters from the digital camera, it will be smaller than one pixel in the
captured picture. Although two thousand meters (2km) seems to be a large distance (or factor) in terms of daily life scale, this is not the case for astronomy.

Example 2: Looking at the Earth and the Solar system.

In this virtual astronomy example, we take the same digital camera in the previous example into outer space and look at the Earth from far away.

Figure 7.5: Object Disappearance Criteria: the Earth of radius \(10^{6.805}\) meters.

Since the camera setting is the same, we have \(\delta(1) = 3.291\) (take \(k = 1\)). In addition, given the fact that the diameter of the Earth equals \(1.2756 \times 10^7\) meters, its radius can be normalized as \(10^{6.805}\) meters. Thus, we can add up \(R\) and \(\delta(1)\) to find out the radius of the culling boundary, i.e., \(10^{R+\delta(1)} = 10^{6.805+3.291} = 10^{10.096}\). As a result, if the camera is over \(10^{10.096}\) meters from the Earth's center, the Earth will be smaller than one pixel on the captured image. Note that \(10^{10.096}\) meters is two thousand \((10^5\) times\) Earth's radius.

Given the fact that the orbit radii of Earth and Mars around the Sun are \(1.49 \times 10^{11}\) meters and \(2.28 \times 10^{11}\) meters, respectively, the shortest Earth-Mars distance is about \(7.9 \times 10^{10}\) meters, or \(10^{16.9}\) meters. Knowing that this exponent, 16.9, is greater than 10.096, if we take a picture of the Earth on Mars using the same digital camera setting as before, the Earth will be smaller than one pixel in the captured image.

Similarly, we can do the same calculation for our Solar system as seen from the nearest
7. Object Disappearance Criterion

star: The diameter of our Solar system is about $10^{13}$ meters, so its radius is about $10^{12.699}$ meters. Using the same digital camera setting, we can compute the radius of the culling boundary: $10^{R+\delta(1)} = 10^{15.996}$ meters. Thus, if we traveled to the nearest stars, the Centaurus triple star system, which is $\sim 10^{16.62}$ meters from Earth, our Solar system will be much smaller than one pixel in the captured image.

7.4 Discussion and Conclusion

Duality between the two Criteria

After presenting the environment caching criterion and the object disappearance criterion, careful readers will notice that the two criteria look very similar. Indeed, they are duals of each other:

\[
\begin{align*}
\text{Environment Caching Criterion:} & \quad S + \delta(k) < L \\
\text{Object Disappearance Criterion:} & \quad R + \delta(k) < L.
\end{align*}
\]

Table 7.1 summarizes related quantities in the two criteria. To simplify the discussion, we use abbreviations, ECC and ODC, to refer to the Environment Caching Criterion and the Object Disappearance Criterion, respectively, from now on.

At the beginning of the derivation of the two criteria (the two inequalities), our first step is to relate $2\theta_{\text{max}}$ and $\theta_{\text{object}}$ to the pixel resolution $\theta_{\text{pixel}}$. In the ODC, $\theta_{\text{object}}$ is the visual angle of the object as seen from the camera whereas in the ECC, $2\theta_{\text{max}}$ is the maximum parallax of a faraway object with respect to the safety region. However, if we look at Figure 6.4 again, we can see that $2\theta_{\text{max}}$ is also a visual angle: it is the visual angle of the safety region subtended at the faraway object. Thus, both $2\theta_{\text{max}}$ and $\theta_{\text{object}}$ are visual angles, i.e., angular sizes of the safety region in the ECC and the central object in the ODC respectively.
7. Object Disappearance Criterion

<table>
<thead>
<tr>
<th>Environment Caching Criterion (ECC)</th>
<th>Object Disappearance Criterion (ODC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2 \theta_{\text{max}}$</td>
<td>$\theta_{\text{object}}$</td>
</tr>
<tr>
<td>$S$ (radius of the safety region)</td>
<td>$R$ (radius of the object)</td>
</tr>
<tr>
<td>Safety region at the center</td>
<td>Object at the center</td>
</tr>
<tr>
<td>Caching boundary</td>
<td>Culling boundary</td>
</tr>
<tr>
<td>$S + \delta(k) &lt; L$</td>
<td>$R + \delta(k) &lt; L$</td>
</tr>
</tbody>
</table>

Table 7.1: Related terms in the Two Criteria.

Furthermore, in the ECC, we confine the camera inside the safety region at the center and look at objects outside the caching boundary, whereas in the ODC, the camera is put outside the culling boundary and looks at the object confined inside the object radius (or object region). The safety region and the object, the caching boundary and the culling boundary, the safety region radius $S$ and the object radius $R$ are actually duals of one another.

![Figure 7.6: Parallax is a relative quantity.](image)
In the end, it is important to note that parallax is indeed a relative quantity between the camera and the object being observed. As depicted in Figure 7.6, when we move the camera to a new position, we observe angular displacement of the star due to change in camera position. In a relative manner, instead of moving the camera, if we move the star in the opposite direction, we can observe the same angular displacement of the star without moving the camera. Therefore, when we move the camera to a new position, we can observe the same angular displacement of the camera at the star location. This is the situation relating the ECC and ODC: We are actually switching the camera and the object in the two criteria. We are talking about parallax of the object as seen from the camera in the ECC ($2\theta_{\text{max}}$) and the parallax of the camera as seen from the object in the ODC ($\theta_{\text{object}}$). Therefore, by limiting the resultant parallax (visual angles) against pixel resolution, we end up with the same form of the inequality with the same $\delta(k)$ function.

**Conclusion**

In conclusion, we devised two useful criteria for accelerating large-scale visualization. Using the Environment Caching Criterion, instead of rendering faraway objects in three-dimensions for every frame, if the criterion is satisfied, we can replace these objects by a pre-rendered cubemap so that the rendering performance can be greatly advanced. At the same time, we have the Object Disappearance Criterion so that we can avoid the rendering of small objects when they are too small to be seen at the camera. This helps to improve the rendering performance as well. In addition, we also discuss the interesting observation that the two criteria are in fact duals of each other. Following are some worthwhile notes about the Object Disappearance Criterion:

- The first issue when using the Object Disappearance Criterion is that we have to take apparent magnitudes into account for completeness in the rendering. A real example
is the stars of the night sky: If we take only physical radii of stars into account, all stars are indeed too small to be viewable with the naked eye on Earth. However, because of the apparent brightness of stars against the dark background, we still can see stars against the night sky. Therefore, when using the object disappearance criteria, we have to take the apparent magnitude into account. That is, when the object disappearance criterion is satisfied for some $k$, we have to compute the apparent size of the object based on its luminosity and distance to the camera. Then, we can draw the object as if it is a point light source similar to stars. The details for rendering with apparent magnitude will be presented in the star rendering chapter (Chapter 9).

- Another issue here is that the representation imposed by the two criteria can be regarded as level of detail in exponential scale. Indeed, while we are navigating inside the Solar system, stars (nearby stars from $10^{16}$ to $10^{19}$ meters) are too far away from our navigation region, and so we can use a cubemap (or environment map) representation for the stars. Then, when we approach the stellar scale, we can seamlessly fade in three-dimensional stars and represent them truly in three-dimensions. Subsequently, when we further move out to the Milky Way scale, these stars become too small and too faint to be visible. Thus, they can be culled. This is essentially a continuous level of detail representation for objects across huge exponential scales.

- Finally, we can impose a hierarchical structure on objects by clustering them using spheres. Then, we can cull them systematically. In practice, we explored the Vantage-Point tree (vp-tree) and the Metric tree (M-tree) structure and used this structure to group stars: In the tree, leaf nodes are individual stars, internal nodes are spheres enclosing child objects, and the root node of the tree encloses everything in the tree structure. In this way, we can apply the object disappearance criterion recursively from the root node. Details of the Vantage-Point and Metric tree method will be presented in the star rendering chapter (Chapter 9).
Part II

Modeling and Rendering for Virtual Astronomy
This part describes the visualization of astronomical objects such as the Solar system, stars, the Milky Way, and galaxies in virtual astronomy. In the following chapters, from Chapter 8 to Chapter 11, we will go through details concerning the modeling and rendering of these entities from the scale of the Earth up to the scale of the Universe:

- Chapter 8 is about the Earth, satellites orbiting around it, the Solar system, and the corresponding Orrery, which governs planetary motion;
- Chapter 9 is about nearby stars, which have true three-dimensional information, as well as associated extra-solar planets;
- Chapter 10 is about the Milky Way galaxy, its rendering as a panoramic image and as a fully three-dimensional model;
- and Chapter 11 is about galaxies beyond our Milky Way, e.g., the Large Magellanic Cloud (LMC), the Small Magellanic Cloud (SMC), Andromeda, etc.

The organization of each chapter is this: First, we present some background knowledge about the astronomical entities being discussed, including their physical information, related astronomical datasets, coordinate systems, etc. Then we present our modeling and rendering methods as well as how to interpret the related datasets.
8

Modeling and Rendering the Solar System

8.1 About our Solar System

Background

In this chapter, we describe the modeling and rendering of various bodies in our Solar system, of which our Earth is a member. There are numerous entities in the Solar system: the Sun, the nine planets, their associated moons and satellites, asteroids, comets, etc. The Sun, being the central body in the system, has mass about 330,000 times that of the Earth and diameter about 109 times the Earth's diameter. Table A.2 in Appendices summarizes information about our Sun. The nine planets, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, and Pluto, orbit about the Sun. Likewise, at the planetary scale, satellites and moons orbit around their associated planets, e.g., the Moon orbits about the Earth. Tables A.2 and A.6 in Appendix A summarize the Moon and planetary statistics.

In addition to the major planets, there are a large number of minor planets, commonly known as asteroids, orbiting around the Sun, mostly between the orbits of Mars and Jupiter.
This region is known as the asteroid belt. Typical asteroids are around 1,000 kilometers in diameter. For some time, astronomers have suspected that these asteroids could be fragments of a planet that was destroyed by a collision with another object. However, since the combined mass of these asteroids is much less than that of the Moon, it is impossible that these asteroids come from a source as large as a planet. The first four asteroids (Ceres, Pallas, Juno, and Vesta) were discovered between 1801 and 1807; presently, over 20,000 asteroids have been identified. Table A.7 of Appendix A lists some well-known asteroids. Notice that some asteroids are actually very close to the orbit of our Earth.

Other families of objects inside our Solar system include comets, meteors, and meteorites. Basically, comets are dirty ice-balls orbiting around the Sun along highly elliptical orbits. If the path of a comet is close to the Earth's orbit, after the comet passes by, some of its fragments (tiny grains of dust or rock fragments) can be left behind and attracted by the gravitation of the Earth. If these fragments fall into the Earth's atmosphere, the result is a meteor shower. Furthermore, if these fragments survive passing through the atmosphere and crash onto the ground, they are known as meteorites. A famous comet-meteor-shower pair is the comet 55p/Tempel-Tuttle and the Leonid meteor shower; the shower appears around November, reaching its peak activity every 33 to 34 years. Tables A.8 and A.9 in Appendix A list some selected comets and meteor showers.

After the discovery of more and more comets, astronomers began to divide them into two groups: short-period comets with orbital periods less than 200 years and long-period comets with periods of more than 200 years. They actually come from two different places: the Kuiper belt and the Oort Cloud, at a distance of 30-50 A.U. (around Neptune and Pluto) and up to ~ 200,000 A.U., respectively. A.U. is the "Astronomical Unit," the mean Earth-Sun distance (see table A.1 in Appendix A for details about astronomical units). Indeed, these regions can be regarded as reservoirs of icy objects; it is important to note that unlike the rocky or metallic asteroids in the asteroid belt, comets from the Oort cloud or the Kuiper
belt are mostly icy objects. When there is a perturbation in the gravitational field, some of these icy objects may move towards the Sun and become comets.

Beyond the Kuiper belt, at about 75-230 A.U., we have a structure, the Heliopause, that results from the out-flowing Solar wind of our Sun. Our Sun moves through the interstellar space in a way similar to a boat traveling through the water. As it moves, the Sun continuously sends out material known as the Solar wind; the out-flowing Solar wind then pushes against the interstellar medium (ISM) and creates a bow shock against the medium. The result is similar to the bow shock created by a traveling boat. The equilibrium boundary where the Solar wind pressure balances the ISM is called the Heliopause. We will show our rendering of the Heliosphere structure, including the bow shock and the Heliopause, later in this chapter.

In a nutshell, if we consider Pluto as the boundary of the Solar system, the diameter of the Solar system is about $\sim 10^{13}$ meters. However, the effective radius of the Solar system is much larger than that: the gravitational effect of our Sun ranges out to at least half of the distance to the nearest star, about one to two light years.

**Dataset**

To model the Solar system for virtual astronomy, the following datasets have been studied:

- Planetary and Lunar Ephemeris: JPL's DE-200, DE-405, and DE-406 [162, 163]
  The dataset provides coordinates of the Sun, Moon, and nine planets relative to the Earth over a period of time.

- Satellite positional information, "NORAD Two-line Element Sets," [92] and Satellite launching information, "Kunstman dataset."
8. Modeling and Rendering the Solar System

- The Heliosphere: Boundary of the Solar Wind [109, 108]
  This includes three-dimensional models of the Termination shock, the Heliopause, and the Bow shock; renderings can be found in Section 8.6.

- World Time Zones Map
  This dataset includes data points for country and time zone boundaries.

- Location of the International Date Line (IDL)
  It is basically a table, listing the coordinates (latitude, longitude) of the points marking the ends of the circular arcs defining the location of the International Date Line in major 20th-century atlases.

Details about these datasets can be found in Appendix A.3. Other datasets related to this research work but not yet explored include the ephemerides for asteroids and comets, the surface textures (image or/and elevation map) for various moons and planets.

Coordinate systems

For observations referred to the Solar system, there are basically two coordinate systems: the Equatorial (or Celestial) coordinate system and the Ecliptic coordinate system. The Equatorial coordinate system is related to the Earth’s rotation, whereas the Ecliptic coordinate system is related to the annual path of the Sun as seen from the Earth. Details about the two coordinate systems can be found in Appendix A.4.

8.2 Modeling and Rendering the Earth

Earth texture  To render a three-dimensional Earth, we make use of three different Earth textures: the Earth daytime texture, the Earth nighttime texture, and the Earth cloud texture. Figure 8.1 depicts the three textures.
Figure 8.1: Earth textures: Daytime (top), Nighttime (middle), and Cloud (bottom).
Day/Night Shading To render the Earth using these textures, we first build a three-dimensional unit sphere (polygonal model) and map these textures onto the sphere accordingly; then we can progressively composite these textures using OpenGL. In practice, two directional lights are employed in the rendering process: one directed from the Sun to the Earth ($\text{light}_1$) and the other in the opposite direction ($\text{light}_2$). The following pseudo-code outlines the rendering method:

```pseudo
> # Render Daytime Earth
> Cull Face ON
> Depth Buffering and Depth Test OFF
> Enable Light1
> Render unit sphere with Earth daytime texture
>
> # Render Nighttime Earth
> Disable Light1
> Enable Light2
> Set blending: SRC=GL_SRC_ALPHA and DST=GL_ONE
> Render unit sphere with Earth nighttime texture
>
> # Render Cloud Earth
> Disable Light2
> Enable Light1
> Depth Buffering and Depth Test ON
> Render unit sphere with Earth cloud texture
>
> # Reset OpenGL Status
> Disable Light1
> Cull Face OFF
> Set blending: SRC=GL_SRC_ALPHA and DST=GL_ONE_MINUS_SRC_ALPHA
```

Note on the pseudo-code:
• As a polygonal model, the unit sphere is composed of a number of triangles. When backface culling is enabled, the rendering of any triangle facing away from the viewing position will be ignored. Thus, we avoid the rendering of the back side of the Earth (see the OpenGL command `glCullFace`).

• In addition, since backface culling is enabled, it is no longer necessary to apply depth buffering to determine the visibility of triangles in the unit sphere. Therefore, we can disable depth buffering and the depth test. However, when we draw the last geometry, the Earth cloud cover, we have to enable depth buffering so that the depth buffer is filled with depth values associated with the Earth's geometry. In this way, when we draw satellites or the Moon later on, these objects can be correctly occluded by the Earth in case they are behind the Earth.

• Blending is related to transparency processing in OpenGL. If it is enabled, the SRC value will be multiplied with the incoming RGB values, whereas the DST value will be multiplied with the existing RGB values in the frame buffer. Then, the two resultant RGB values will be summed up and written back to the frame buffer. Therefore, using `SRC=SRC_ALPHA`, the alpha channel (the A channel of the RGBA texture) of the nighttime texture will be multiplied with the RGB values of the nighttime texture; then, with `DST=GL_ONE`, we keep the existing RGB values in the frame buffer (the daytime rendering) the same and add to it the previously multiplied RGB values (the nighttime texture). Therefore, the nighttime texture is overlaid on the daytime texture and the same happens for the cloud texture (see OpenGL command `glBlendFunc`).

• Furthermore, since we use `light1` to shine on the daytime Earth and `light2` to shine on the nighttime Earth, only the part of the daytime (or nighttime) texture facing `light1` (or `light2`) is revealed. The unlit sides are guaranteed to contribute nothing, as required.
Figure 8.2: Rendering the Earth: The top row shows only the daytime texture; the second row adds in the nighttime texture, whereas the third row adds in the cloud texture. The last row adds in the rendering of the atmosphere.
Figure 8.2 shows renderings of the Earth from different viewpoints: America and Asia. In the first row, we show only the daytime texture illuminated by sunlight (*light1*). Then, we add in the nighttime (illuminated by *light2*) and cloud textures, producing the second and third rows in the figure. Finally, we add in the rendering of the atmosphere, which we will discuss shortly.

**Atmospheric Shading**

In this subsection, we present our rendering method for the atmosphere, as shown in the last row of Figure 8.2. Our technique is a simplified version of the method in [128] for real-time rendering. Because of the scattering effect of the Earth's atmosphere, when we look at the Earth from the outer space, the thicker the atmosphere, the more sunlight it reflects (or scatters). Therefore, we shall expect a glowing specular highlight around the rim of the Earth facing the Sun direction (see renderings shown in the last row of Figure 8.2).

![Figure 8.3: Modeling the Earth Atmosphere.](image)
To model this phenomenon, we construct the atmosphere model shown in Figure 8.3. Assuming that the Earth radius and atmosphere thickness are uniform over the Earth, we can define the following quantities:

- $R$: the radius of the Earth.
- $dR$: the thickness of the atmosphere.
- $\theta$: the angular distance from the central viewing direction.
- $t$: the thickness of the atmosphere along the camera’s viewing direction with $x = (R + dR) \sin \theta$.

Thus, we have

$$t(x) = \begin{cases} 
    dR & \text{when } x = 0 \text{ or } \theta = 0 \\
    0 & \text{when } x = R + dR \text{ or } \theta = \frac{\pi}{2}.
\end{cases}$$

Note that we do not multiply $t(x)$ by two when $x \in [R, R + dR]$ so that we can make $t(x)$ continuous at $x = R$. Practically, this does not create too much difference in the rendering as $dR$ is small compared to $R$.

To model the atmosphere geometrically, we first build a polygonal model in the form of a hemisphere: the hemisphere model is depicted as a tilted wireframe in Figure 8.4. Then, similar to the billboard method in computer graphics, the orientation of this hemisphere is tilted in so that its tip always points towards the viewing position (camera or eye).

Hence, with this billboard strategy, we can make use of the atmosphere thickness equation $t(x)$ to mimic the reflectance of the atmosphere as seen from the outer space. Here, we define the reflectance as follows:

$$\text{reflectance} = \frac{t(x) - t(0)}{t(R) - t(0)} \cdot (\max_{\text{opacity}} - \min_{\text{opacity}}) + \min_{\text{opacity}}$$

$$\text{opacity} = \text{intensity} = \text{reflectance}.$$
In the rendering, the atmosphere color should be a little bit bluish because the atmosphere scatters blue light to a greater extent in the visible light spectrum. In addition, intensity and opacity of the atmosphere color (vertex color in the hemisphere model) changes according to the reflectance equation above. Moreover, this hemisphere model has to be illuminated by the sunlight (light intensity) so that the reflectance is modulated by the sunlight. Note that min_opacity and max_opacity are tuning parameters in this model; we typically set them to 0.1 and 0.8 respectively.

To demonstrate the Earth shading using both atmospheric rendering and Earth textures, we simulate the rendering of the Earth over a day at Bloomington, Indiana, United States. Figure 8.5 shows the animation sequence: we take eight snapshots of the Earth at a fixed viewpoint over the twenty-four hours starting at 6am, followed by 9am, noon, 3pm, ..., up to 3am on the next day. The top-left sub-figure corresponds to sunrise at 6am, the sub-figure below it corresponds to 9am and so on. We can clearly observe the change in daytime and nighttime as well as the change in reflectance of the atmosphere. Interestingly, since
the selected simulation date is in August, we can see that the Arctic region is always in
daytime.

In terms of implementation, if both vertex and pixel shading are available, we can
further speed up the rendering process by compositing textures using a pixel shader and by
applying a vertex shader to compute the reflectance equation over the atmosphere as well.
Nevertheless, if the graphics hardware does not support shaders, we can still use the above
rendering method in standard OpenGL for rendering the Earth.

Markers on Earth
Besides realistic rendering, we also plot the longitude/latitude grid, the country/time zone
boundaries, and the international date line over the surface of the Earth to make the display
more informative. Figure 8.6 shows the rendering of these entities from four different
viewpoints: Pacific region, Asian region, American region, and Europe region. Note that
the rendering of cloud texture and atmosphere is disabled for clarity.

8.3 Modeling and Rendering Satellites

In this section, we discuss the modeling and rendering of various satellites orbiting the
Earth. There are several types of satellites; to visualize the type, we use distinct colors and
models.
Figure 8.5: Simulation of an Earth day (Indiana time): from sunrise at 6am (top-left) to late night at 3am (bottom-right).
Figure 8.6: The Longitude/Latitude grid (markers and grid: yellow, Equator: red, Tropics of Cancer (23.5°N) and Capricorn (23.5°S): blue), the Country/TimeZone boundaries, and the International Date line on Earth model.
Using the NORAD dataset, we can obtain the three-dimensional locations of satellites around the Earth. Details of the implementation for manipulating the NORAD dataset are presented in Appendix A.3. During the rendering, we can display these satellites in four different modes:

1. Points — three-dimensional points in the above colors.

2. XYZ-Frames — XYZ (red/green/blue) axes depicting the Frenet frame (Tangent/Normal/Binormal) of the satellite trajectory.

3. Bounding box — rectangular boxes bounding the region occupied by each satellite.

4. 3D-Models — different types of satellites are shown as different polygonal models, which are stored in Wavefront .obj format.

Figure 8.7 shows the rendering of the satellites using colored points. Note that the yellow curve in the images depicts the trajectory of the International Space Station (ISS).
Figure 8.7: Different types of satellite have different colors.
Figure 8.8: Display mode for satellites: points, boxes, frames, and 3D-models.
Altogether, there are about 1,000 satellites in the NORAD dataset, with the number of satellites belonging to each type shown in the captions of Figure 8.7. Notice that more than half of the satellites are communication satellites, shown as yellow points. Figure 8.8 shows the four satellite display modes. Note that in order to show the satellites more clearly, we disabled the rendering of stars as well as simplifying the shading of the Earth.

**APOD Animation**

As a visualization, we made a short animation sequence focusing on the satellites. This animation was selected for APOD (Astronomy Picture of the Day), 14 July 2003 ([http://antwrp.gsfc.nasa.gov/apod/ap030714.html](http://antwrp.gsfc.nasa.gov/apod/ap030714.html)). Figure 8.9 shows some of the keyframes in the animation sequence.

In fact, there are over a thousand satellites in the sky orbiting the Earth; this small animation shows most of them at a given time and then zooms in to the International Space station (ISS) at the end. The frames are rendered using our software in which all objects are positioned and rendered correctly at the same given time (2:21pm 5th June (GMT)), including the Earth and its day/night shading, as well as the Sun. Again, since stars could distract people from distinguishing the satellites, we disabled star rendering.

In this animation, (from top-left to bottom-left, and then top-right to bottom-right), we start our journey in the evening in Africa, and then rotate our view to look at the United States, which is at about 9:21am (Eastern) in the afternoon. To position the satellites, we make use of the **SGP4/SDP4** satellite tracking software (see Appendix A.3) for near-Earth (period < 225 minutes) and deep-space (period >= 225 minutes) satellites respectively. At about 38,500 km from Earth's surface (about 3 times the Earth's diameter), we find the ring structure of geo-stationary (deep-space) satellites co-rotating with the Earth. There, we can see that most geo-stationary satellites are communication satellites, shown in yellow. Moving towards the Earth, we find another layer of near-Earth satellites, which are about
Figure 8.9: APOD 14th July 2003: Satellites over the Sky (GMT June 24th 2003 2:21pm) by Andrew J. Hanson, Philip Chi-Wing Fu, and Priscilla Frisch.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
300km to 1,500km from Earth’s surface. Note also that there are more satellites over the northern hemisphere since the communication over the northern hemisphere is more active. Finally, we move into this layer and zoom in to the ISS, which is also a low orbit object. The ISS orbits at about 389km above central United States at this time. The rendering can be done in real-time, and is fully interactive.

8.4 Modeling and Rendering the Sun

To model and render the Sun, we apply the billboard strategy: the Sun is modeled as a textured square such that the normal to the plane of the square always points to the viewing position. Because the Sun is spherical in shape, the viewer sees the expected shape, since the disk-shaped billboard Sun always faces the viewpoint.

![Figure 8.10: Rendering the Sun using three Gaussian textures.](image)

In addition, to simulate the brightness of the Sun in contrast to the dark background...
and make the Sun look glowing, we apply three different Gaussian textures to mimic the Sun's profile. Each Gaussian texture has different color, amplitude, and variance in the intensity distribution. Figure 8.10 shows the three Gaussian textures and the composited result: the rendering of the Sun. The first texture is brighter and redder, but has a smaller variance; it contributes to the central bright region of the rendered Sun. The second texture has a slightly larger variance than the first one, but is more yellow in color. By compositing these two textures, we obtain the central saturation and first falloff around the rim of the central saturation. Finally, the third texture is very pale in color and has a much larger variance, thus it creates the fadeout (corona) around the Sun. In the actual implementation, this is accomplished by using the particle rendering library for the modeling of the three-dimensional Milky Way (see Chapter 10 for details).

8.5 Modeling and Rendering the Ephemeris

In this section, we move beyond the Earth and Sun to the whole Solar system. As mentioned previously, we can make use of the JPL Ephemeris dataset to determine three-dimensional positions of various bodies (Moon, Sun, and other planets) in our Solar system relative to the Earth at a given time. Details concerning this dataset is presented in Appendix A.3.

Since the ephemeris is extremely accurate, we can use it to simulate Solar eclipses, and lunar eclipses, in addition to planetary trajectories and retrograde motions.

1. Simulating Solar Eclipses

Solar eclipses occur when the Moon moves to certain region between the Sun and the Earth. When this happens, the Sun's disk is blocked by the Moon and the Sun becomes
invisible (or partially visible) as seen from certain areas on Earth. Figure 8.11 plots the areas on Earth’s surface where recent Solar eclipses have occurred or will occur (1996–2020). Each labeled region in the figure corresponds to a specific Solar eclipse.

![Figure 8.11: Recent Solar Eclipses (1996–2020).](image)

Figure 8.11: Recent Solar Eclipses (1996–2020).

Figure 8.12 shows our simulation for the Solar eclipse of 8 April 2005. Referring to Figure 8.11, this Solar eclipse starts in the Pacific ocean and extends to Central America. Because of the accuracy of the JPL ephemeris dataset, we can precisely simulate the trace of Solar eclipses on Earth’s surface. Figure 8.12(a) depicts the situation when the Moon moves to a position between the Sun and the Earth. The time related to this simulation is 22:06 GMT 8th April 2005. We can outline the shadow volume behind the Moon using red and blue lines as shown in Figure 8.12(b). The red lines outline the umbra region where the Sun is totally blocked by the Moon, whereas the blue lines outline the penumbra region where the Sun is partially blocked.
Figure 8.12: Simulation of Solar Eclipses in 2005.

(a) Moon in-between Sun and Earth
(b) Shadow casted by the Moon
(c) Looking at the Earth
(d) Penumbra and the Path on Earth
(e) One hour earlier
(f) Two hours earlier
8. Modeling and Rendering the Solar System

Then, in Figure 8.12(c), we move our viewpoint to a position looking at the whole Earth; we can see that the red lines intersect with the Earth at locations around Central America. Furthermore, in Figure 8.12(d), we turn on the shadow region (penumbra region) and the trace of the Solar eclipse, which are outlined as grey lines and a light red curve respectively. We can see that the red curve matches precisely the prediction shown in Figure 8.11: the red curve going from the Pacific Ocean to Central America. Finally, if we set the time back to one and two hours earlier, we get the renderings shown in Figure 8.12(e) and (f). In addition, it is worth noting that because of the accuracy offered by the JPL ephemeris dataset, it is possible to simulate lunar eclipses as well.

2. Simulating Planetary Trajectories

Besides supporting eclipse simulations, the JPL ephemeris dataset provides planetary positions over time for us to simulate the Solar system orrery.

Figure 8.13 shows the simulated orrery on 14th August 2003. The first row shows trajectories of the inner planets of the Solar system including Mercury, Venus, Earth, and Mars in cyan, yellow, blue, and orange colors, respectively. We set 150 days as the length of the simulated trajectories. Note that Jupiter is much further away from Mars than Mars from Earth, so it is not shown in the simulated picture. The next row in the figure zooms out to the outer planets; the length of the simulated trajectories is extended to 3650 days. Note that the orbital periods of outer planets are much longer than those of the inner planets; we have to increase the length of simulated trajectories so that we can visualize their orbits.

3. Simulating Retrograde motion

We can also simulate retrograde motions of the planets using the JPL ephemeris dataset. Because the Earth is not the center of the Solar system, if we look at other planets against
Looking at inner planets with 150 days as the trajectory length

Looking at outer planets with 3650 days as the trajectory length

Figure 8.13: Orrery of the Solar System on 14th August 2003.
the background stars, instead of moving in a unique direction like the Sun, all planets exhibit occasional retrograde motions as seen from Earth.

Figure 8.14 shows the simulation of retrograde motions exhibited by the planets Mercury, Jupiter, and Pluto. The first four snapshots animate the retrograde motion of Mercury. We can see from the simulation that, after reaching a certain point between snapshots 1 and 2, Mercury reverses its moving direction. Later on, it reverses its direction again (between snapshots 3 and 4) and continues traveling to the left. For visualization purposes, at the lower-right corner of each snapshot, there is a small overview picture showing the Mercury-Sun-Earth system. The red bar in these overview pictures reveals the direction to Mercury as seen from Earth against the background. If we animate snapshots 1 to 4, it is possible to observe the red bar moving backward when the retrograde motion of Mercury happens.

Retrograde motion not only happens for inner planets between the Sun and the Earth; it happens for all other planets as well. The last row in the figure simulates retrograde motions exhibited by Jupiter and Pluto. Since Pluto is so far from the Earth and Sun, the retrograde cycles exhibited by Pluto are much smaller.

8.6 Modeling and Rendering the Heliosphere

As mentioned before, the Heliosphere is a structure where the out-flowing Solar wind meets the interstellar medium (ISM). The Heliosphere structure can be studied using the following three models provided by theoretical calculations of Timur Linde at the University of Chicago:

- Termination shock (pseudo-color in yellow)
  surface where the Solar wind drops below sound speed (75-90 AU).
Figure 8.14: Retrograde motions exhibited by Mercury, Jupiter, and Pluto as seen from Earth.
- **Heliopause (pseudo-color in green)**
  actual contact sheet between the Sun’s ions and the Galaxy’s ions (~ 110 AU).

- **Bow shock (pseudo-color in blue)**
  supersonic wavefront of the Solar wind meeting interstellar medium (~ 230 AU).

Figure 8.15 depicts the rendering of the Heliosphere. The left column shows the surface rendering, whereas the right column reveals the underlying wireframes. The first row looks at all three models: Termination shock, Heliopause, and Bow shock. The second row zooms in to the Heliopause behind the bow shock; and the third row further zooms in to the Termination shock inside the Heliopause. Notice that the orbits of Pluto, Neptune, and Uranus (in purple, blue, and cyan respectively) are located inside the Termination shock. In the rendering of the bow shock, an animated “flaming texture” is mapped onto its surface to make it look vivid in the animation (thanks to Eric Wernert for creating the framing texture).

In addition to the satellite rendering application, the rendering of the Heliosphere was selected for the NASA APOD of 24th June 2002. Figure 8.16 is the picture shown in the APOD webpage http://antwrp.gsfc.nasa.gov/apod/ap020624.html. The explanation of the APOD follows:

Explanation: Where does the Sun’s influence end? Nobody is sure. Out past the orbits of Neptune and Pluto extends a region named the Heliosphere where the Sun’s magnetic field and particles from the Solar wind continue to dominate. The surface where the Solar wind drops below sound speed is called the termination shock and is depicted as the inner oval in the above computer-generated illustration. It is thought that this surface occurs as close as 75-90 AU – so close that a Pioneer or Voyager spacecraft may soon glide through it as they exit the Solar system at about 3 AU/year. The actual contact sheet
Figure 8.15: Rendering the Heliosphere: the Bow Shock (blue), the Heliopause (green), and the Termination Shock (yellow).
Figure 8.16: The Sun's Heliosphere and Heliopause. Model by Timur Linde, Credit & Copyright: P. C. Frisch (University of Chicago) et al., Indiana University.
between the Sun's ions and the Galaxy's ions is called the Heliopause and is thought to occur at about 110 AU. It is depicted above as the middle surface. The Sun's Heliopause moves through the local interstellar medium much as a boat moves on water, pushing a bow shock out in front, thought to occur near 230 AU.

8.7 Conclusion

So far, we have discussed the modeling and rendering of various entities including our Earth, satellites, etc. In order to incorporate these three-dimensional models into large-scale environments, we had to consider the following:

1. All three-dimensional models, such as the Earth and satellites, should have their geometries defined inside a $2 \times 2 \times 2$ cube, i.e., the model vertices are bounded by $[-1, +1]$ in all dimensions.

2. Then, during the rendering, we can position these objects at their appropriate scales by making use of the enhanced modelview transformation presented in Part I of this thesis.

This rendering approach is suitable for most conventional three-dimensional objects. However, for entities like stars, galaxies, or the Milky Way, it is inefficient to use bare three-dimensional geometries to represent them. For example, it is impractical to use polygonal spheres to model all the stars. At present, no current rendering engine can process several thousand spheres in real-time. Special considerations have to be taken into account to model and render these entities; related methods will be presented in the next three chapters.

Nevertheless, there is still some additional work to do in the modeling of the Solar system. We can, for example, add in comets and asteroids based on the comet and asteroid
ephemeris datasets, model these objects three-dimensionally, simulate the Oort cloud and the Kuiper belt, add models of the surface activities of the Sun, incorporate textured spheres for other planets, add planetary moons, and utilize elevation maps for Mars, the Moon, and the Earth.
Modeling and Rendering Stars

Photorealistic rendering of stars is a major issue in astronomy visualization and virtual astronomy. The problem of interactive and real-time star rendering is especially challenging due to the increasingly large amount of three-dimensional stellar data available from the Hipparcos dataset [53, 137, 61], and, soon, the Sloan sky survey [166, 165]. Both dataset size and the astronomical range of position scales present unique problems to the visualization process, where interactive speed and realism are both major objectives.

The first two sections of this chapter begin with some background knowledge about star rendering: description of datasets, coordinate systems, basic physics and mathematics concerning stars, etc. In the next section, we propose a new approach, stellar database structures, to support quality interactive rendering based on scalable tree structures, the VP-tree and the M-tree. Extending the concepts of previous chapters on very large scales, we implement power homogeneous coordinates (PHC) as a powerful method for representing distances and vectors inside the tree structures; we are thus able to devise efficient pruning and rendering rules that deal with large scale differences as well as visibility selection issues.

To address the issues of realism and rendering performance, we discuss including
within our framework a number of effective star-rendering techniques such as star resizing, star coloring, star rescaling, and star bucketing. We conclude with additional results on the visualization of three-dimensional star fields, constellations, extra-solar planets, star trajectories, and related phenomena.

9.1 Background

Like our Sun, the stars shining in the night sky are gas balls held together by gravitational force. Because of the high temperature and pressure inside the core, stars can generate radiation on their own by means of nuclear fusion. Subsequently, heavy elements are generated as by-products. Stars of different temperatures produce starlight of different colors, while stars of different luminosities and at different distances from us have different apparent brightnesses.

Since stars are a principal source of the radiant energy that shapes our entire perception of the observable Universe [103], rendering them well is of great significance. The purpose of this chapter is to treat the depiction and rendering of stars in virtual astronomy. We will focus on these issues:

- First, we are doing 3D astronomy with arbitrary spatial navigation, not 2D conventional planetarium views with an Earth-fixed viewpoint.

- Second, we are working with scenarios that require high interactivity, so off-line rendering techniques used in pre-stored 3D virtual astronomy are not generally adequate.

- Third, we place no practical restrictions whatsoever on scales, permitting objects of literally astronomically different dimensions to coexist on the screen, and permitting
continuous scale changes of viewpoint while maintaining a technically correct star field at all times.

- Fourth, we are combining other types of data interactively, so that the rendering of the stars themselves is only part of the problem and must therefore be particularly efficient by itself.

**Issues: Interactive Star Rendering**

In this subsection, we discuss three important issues for interactive star rendering:

- **Large Dataset for Interactive Rendering**

  Over 100,000 stars are cataloged in the Hipparcos dataset [61] — These stars have precise positional information. Though this may not be an exceptionally large number for a database, it is quite significant if we need to do interactive rendering. It is difficult to render this order of magnitude of points in conventional graphics hardware at interactive speeds, and the bottlenecks become increasingly significant when we begin adding realistic details such as texture, size, and color. When we attempt to add supplementary astrophysical data such as gas clouds, real-time rendering can be even more difficult to achieve.

  Table 9.1 summarizes the number of stars in the Hipparcos Catalogue listing the apparent magnitude (magnitude in Johnson V units). Note that stars with apparently incorrect parallaxes are ignored and the Sun is excluded in these numbers.

- **Large in Scale**

  Besides being large in quantity, the dataset is large in scale for applications such as a planetarium fly-through. When we are working with virtual planetariums, stars are of course not the only objects shown in the virtual environment. We could have
the Earth, the Solar system, the Milky Way Galaxy, or some faraway galaxies. By putting altogether different celestial bodies at different scales, we end up with a huge scale variation at any given observation point in space. Rendering stars at different scales is unavoidable if we need to have continuous scale transitions in the virtual fly-through.

- **Interpretation of the Raw Dataset**

Finally, we must relate physical star characteristics to the actual rendering parameters. Briefly, what we have in the Hipparcos dataset are raw astronomical data such as the apparent magnitude, B-V color index, and parallax. However, what we have to provide to the computer graphics program are the colors, point sizes, and coordinates of the stars. A well-defined procedure for determining the rendering parameters from the raw astronomical data is essential.

### Datasets

In our implementation for modeling and rendering stars, we have explored the following datasets:

<table>
<thead>
<tr>
<th>$V_{\text{mag}} \leq$</th>
<th>Number of stars</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>49</td>
</tr>
<tr>
<td>3</td>
<td>177</td>
</tr>
<tr>
<td>4</td>
<td>516</td>
</tr>
<tr>
<td>5</td>
<td>1618</td>
</tr>
<tr>
<td>6</td>
<td>5013</td>
</tr>
<tr>
<td>7</td>
<td>15388</td>
</tr>
<tr>
<td>8</td>
<td>40829</td>
</tr>
<tr>
<td>9</td>
<td>81289</td>
</tr>
<tr>
<td>10</td>
<td>104193</td>
</tr>
</tbody>
</table>

Table 9.1: The Hipparcos Catalogue (apparent magnitude - in Johnson V units).
• The Bright Star Catalog (BSC) [90, 91]

This catalog records intrinsic and positional information of stars. However, since the data quality and quantity provided by the Hipparcos Catalogue is better, this catalog was eventually replaced by Hipparcos in our data environment.

• The Hipparcos (High Precision Parallax Collecting Satellite) Catalogue [53, 137]

Similar to the BSC, this catalogue contains a large quantity of very high quality astrometric and photometric data including proper motions, parallaxes, trigonometry, photometry, fundamental information, etc. There are 118,218 records (stars) in the main catalogue. Note that the precision of parallax available in this catalogue is the highest among all catalogs we currently have access to.

• Catalog of Constellation Boundary Data [3]

This dataset provides lists of points (in RA and DEC) outlining the boundary of the standard eighty-eight constellations.

• General Catalog of mean radial velocities [10]

This catalog provides the radial velocities of 36,145 stars so that, by incorporating radial velocity with the three-dimensional position and proper motion from the Hipparcos Catalogue, we can determine the space velocity of stars (see the next section for details).

• Extra-solar planets dataset [148, 112, 113]

This dataset provides information (mass, orbital period and radius, etc.) about extra-solar planets. Up to May 2003, about 102 extra-solar planets had been discovered, but this number was raised to 110 in July 2003; this is an active research area in astronomy and the extra-solar planet dataset keeps growing almost every month.

Details about these datasets can be found in Appendix A.3. Other related datasets include star labels in the Bayer system ($\alpha$, $\beta$, ...) and the Sloan sky survey.
Coordinate systems

When positioning stars in the sky, most datasets use the Equatorial coordinate system, which represents the direction to a star using right ascension (RA) and declination (DEC). Using the parallax information, astronomers can determine three-dimensional positions of stars relative to the Earth. To place stars in the coordinate frame used for the Milky Way Galaxy, we can transform from Equatorial to Galactic coordinates. Details about these coordinate systems are presented in Appendix A.4.

9.2 Basic Physics and Mathematics about stars

In this section, we detail several physics and mathematics formulae and knowledge useful for manipulating the star dataset, which is typically the Hipparcos Catalogue.

Parallax, the Parsec, and Distance

As mentioned in Chapter 1, Friedrich Bessel first measured the parallax of the star Cygnus 61° in the constellation Cygnus and determined its approximate distance. Parallax is the angular displacement of a star given the 2 A.U. observational baseline. A.U. is the "Astronomical Unit," mean Earth-Sun distance (see table A.1 in Appendix A for details about astronomical units). Then, distances to stars can be found based on simple trigonometry (see Figure 9.1).

Since measured angles (parallaxes) are usually very small, we can notice that

\[
\text{parallax} \propto \frac{1}{\text{distance to star}}.
\]

To simplify the computation, astronomers defined the parsec unit, which is related to distances measured from parallaxes as follows:
A star at a distance of one \( \text{parsec} \) from the Sun has a \( \text{parallax} \) of one \( \text{arcsecond} \).

So, we have the following formula:

\[
d(\text{parsecs}) = \frac{1}{p} (\text{arc-seconds}) .
\]

Note that in the Hipparcos Catalogue, parallax is precisely measured in milli-arc-seconds, say \( plx \), so,

\[
d(\text{parsecs}) = \frac{1000}{plx} (\text{milli-arc-seconds}) .
\]

Furthermore, the Hipparcos Catalogue provides the parallax error \( \Delta plx \). Letting \( \Delta d \) be the
error corresponding to a measurement of \( d \) (parsecs), we have

\[
d = 1000 \cdot \frac{1}{2} \cdot \left\{ \frac{1}{(plx + \Delta plx)} + \frac{1}{(plx - \Delta plx)} \right\}
\]

\[
= 1000 \cdot \left[ \frac{plx}{(plx^2 - \Delta plx^3)} \right] 
\]

\[
= 1000/plx \cdot \left[ 1 - \left( \frac{\Delta plx}{plx} \right)^2 \right]^{-1}
\]

\[
d \approx 1000 \left( \frac{1}{plx} + (\Delta plx)^2/plx^3 \right), \quad (9.1)
\]

and the error term,

\[
\Delta d = \| 1000/(plx + \Delta plx) - d \| 
\]

\[
= 1000 \| 1/(plx + \Delta plx) - 1/plx - (\Delta plx)^2/plx^3 \| 
\]

\[
= 1000 \left\| \left[ \left( -plx^2(\Delta plx) - plx(\Delta plx)^2 - (\Delta plx)^3 \right) \right] 
\right\| 
\]

\[
\Delta d \approx 1000 \Delta plx / plx^2. \quad \text{(note: put } (\Delta plx)^3 = 0 \text{)}
\]

Note that when using the Hipparcos Catalogue, computing distances \( d \) using the above formula is statistically more accurate because the error term \( \Delta plx \) could be significant when compared to \( plx \).

**Brightness of stars: luminosity, apparent magnitude, and absolute magnitude**

Definitions:

- **Luminosity**, \( \text{lum} \),
  - total amount of energy a star gives off per second.
- Apparent magnitude, \( m \),
  the brightness of a star as it appears to us.

- Absolute magnitude, \( M \),
  the brightness of a star as it would appear to us if it were exactly 10 parsecs away from us.

Rules:

1. The brightest stars appear to be 100 times brighter than the faintest stars as seen by the unaided eye.

2. A difference of 5 magnitudes corresponds to a factor of 100 in brightness. The smaller the magnitude the brighter the star is.

3. One magnitude difference is a factor of \( \sqrt[3]{100} \) in brightness; astronomers usually approximate \( \sqrt[3]{100} \) by 2.5 to simplify their calculations.

Let \( (m_1, m_2), (L_1, L_2), \) and \( (d_1, d_2) \) be the apparent magnitudes, luminosities, and distances (parsecs) of two stars, respectively. Noting that the capital letter \( M \) refers to absolute magnitude, based on the definitions and rules above (take \( L = 1 \) when \( M = 0 \)), the luminosity is

\[
L = (\sqrt[3]{100})^{-M}
\]  

(9.2)

and the magnitude difference is

\[
m_1 - m_2 = -2.5 \log_{10} \frac{L_1}{L_2}.
\]

By the inverse-square law, we can relate luminosity and distance,

\[
L \propto \frac{1}{d^2}.
\]
Therefore, we can rewrite the previous equation as,

\[ m_1 - m_2 = -5 \log_{10} \left( \frac{d_2}{d_1} \right), \]

so that by comparing apparent magnitudes of the two stars, we can determine the ratio of their distances to us.

Furthermore, based on the definition of absolute magnitude (apparent magnitude when the associated star is 10 parsecs from us), we can replace \( m_1 \) by \( M \), \( m_2 \) by \( m \), \( d_1 \) by 10, and \( d_2 \) by \( d \):

\[ M - m = 5 - 5 \log_{10} d. \]  \hfill (9.3)

This equation allows us to compute the actual distance to the star using just the apparent and absolute magnitudes.

Finally, using the distance \( d \) (in parsec) computed from the parallax equation, Eq. (9.1), in the previous subsection, and the apparent magnitude (magnitude in Johnson V), \( m \), available in the Hipparcos Catalogue, we can substitute Equation (9.3) into Equation (9.2):

\[
L = \left( \sqrt[3]{100} \right)^\left( -5 + 5 \log_{10} d - m \right)
= 0.01 \cdot d^2 \cdot \left( \sqrt[3]{100} \right)^{-m}
\approx 0.01 \cdot d^2 \cdot (2.5)^{-m}. \]  \hfill (9.4)

This allows us to compute relative luminosity of stars based on the data provided in the Hipparcos Catalogue. Note that these equations are useful for computing star sizes when navigating through a star field in 3D.

In addition, it is important to note that for Equation (9.3) to be valid, it is assumed that the space between the star and the Earth is completely empty. However, in reality, the intervening space could be filled with some interstellar medium such as nebulae, dust, molecular...
clouds, etc. Part of the radiation from the star could be absorbed or scattered away by such material. This loss in magnitude is called extinction, denoted by $A(r)$; Equation (9.3) can be updated to include extinction as follows:

$$ M - m = 5 - 5 \log_{10} d + A(r). $$

Note that another source of extinction may include the Earth's atmosphere if the measurement is done from Earth. Another stream of research work is being carried out with Prof. Priscilla Frisch at the University of Chicago to study the structure of gas clouds in interstellar space to help us account for $A(r)$.

**Proper motion, Radial velocity, and Space velocity**

![Diagram illustrating proper motion, radial velocity, and space velocity](image)

Figure 9.2: Transverse velocity (Proper motion) + Radial velocity = Space velocity.

**Definitions:**

- Radial velocity, $v_r$.
  
  velocity of stars directly toward or away from us, measured by the Doppler effect.
- Proper motion, \((\dot{\theta}, \dot{\phi})\),
  angular velocity of stars across the sky in the \((\theta, \phi)\) domain.

- Space velocity, \((\dot{x}, \dot{y}, \dot{z})\),
  actual velocity of stars relative to the Sun, or a chosen standard of rest.

Figure 9.2 depicts these velocities geometrically. Note that \((r, \theta, \phi)\) is the polar coordinate
of the star, corresponding to the Cartesian values \((x, y, z)\); \(\theta\) and \(\phi\) are also called RA and
DEC, respectively, whereas \(r\) is the distance to the star.

Based on the Equatorial coordinate system, which is illustrated in Appendix A.4, the
three-dimensional position, \((x, y, z)\), can be written as

\[
\begin{align*}
  x &= r \cos \phi \cos \theta \\
  y &= r \cos \phi \sin \theta \\
  z &= r \sin \phi,
\end{align*}
\]

with \(0 \leq \theta \leq 2\pi\) and \(-\frac{\pi}{2} \leq \phi \leq \frac{\pi}{2}\). So, by taking the derivative of the above system with
respect to time, we get the space velocity,

\[
\begin{align*}
  \dot{x} &= \dot{r} \cos \phi \cos \theta - r \dot{\phi} \sin \phi \cos \theta - r \dot{\theta} \cos \phi \sin \theta \\
  \dot{y} &= \dot{r} \cos \phi \sin \theta - r \dot{\phi} \sin \phi \sin \theta + r \dot{\theta} \cos \phi \cos \theta \\
  \dot{z} &= \dot{r} \sin \phi + r \dot{\phi} \cos \phi,
\end{align*}
\]

and its magnitude,

\[
\left[ (\dot{r})^2 + (r \dot{\phi})^2 + (r \dot{\theta} \cos \phi)^2 \right]^\frac{1}{2}.
\]

Denoting by \(\Delta \dot{\theta}, \Delta \dot{\phi}, \text{ and } \Delta \dot{r}\) the errors of \(\dot{\theta}, \dot{\phi}, \text{ and } \dot{r}\), respectively, we can obtain the
errors of \(\dot{x}, \dot{y}, \text{ and } \dot{z}\):

\[
\begin{align*}
  \Delta \dot{x} &= \Delta \dot{r} \cos \phi \cos \theta - r \Delta \dot{\phi} \sin \phi \cos \theta - r \Delta \dot{\theta} \cos \phi \sin \theta \\
  \Delta \dot{y} &= \Delta \dot{r} \cos \phi \sin \theta - r \Delta \dot{\phi} \sin \phi \sin \theta + r \Delta \dot{\theta} \cos \phi \cos \theta \\
  \Delta \dot{z} &= \Delta \dot{r} \sin \phi + r \Delta \dot{\phi} \cos \phi.
\end{align*}
\]
Note that, using this method, we do not need to compute the transverse velocity explicitly before computing the space velocity. We can compute the space velocity directly from the proper motion and the radial velocity.

**Local standard of Rest (LSR)**

Definitions:

- Apex,
  
  the point (or direction) towards which the Sun's motion among the stars seems to be directed.

- Antapex,
  
  the opposite point (or direction).

Because the Sun is in motion, stars around the apex appear to be approaching us, whereas stars around the antapex appear to be moving away. Note that we can determine space velocity of stars based on the method described previously. Now, assuming that the velocities of the stars are randomly distributed, we can define the local standard of rest (LSR) as follows,

**LSR definition:**

*The Sun's velocity is the opposite of the mean velocity of the set of stars being considered.*

Note that the LSR is a coordinate framework orbiting around the Milky Way Galaxy with the Sun. In other words, with respect to the LSR, the mean velocity of the set of stars being considered is always zero according to the assumption. As depicted in Figure 9.3, to cancel out the Sun's motion in stars' observed motion (space velocity), the peculiar motion...
of a star is defined as the velocity of an individual star with respect to the LSR; it can be computed by adding the Sun's velocity to its space velocity. The Sun and its neighborhood (neighboring stars) are similar to passengers randomly walking inside a big compartment of a fast-moving train. The compartment is the LSR, which is orbiting at a very high speed around the Milky Way Galaxy, whereas the random walk of the passengers in the compartment is analogous to the peculiar motions exhibited by the Sun and the members of its neighborhood.

**Temperature and Color of stars**

Starlight received on Earth is in the form of a spectrum of electromagnetic radiation. Such a spectrum is a plot of intensity against wavelength, normally including the visible region of the Electromagnetic (EM) spectrum.

- According to Wien's Displacement Law,

  \[ \lambda_{\text{max}} T = \text{constant} \]
where $\lambda_{\text{max}}$ is the wavelength at which the energy given off is at the maximum, and $T$ is the temperature of star.

Therefore, given a starlight spectrum, we can determine the temperature $T$ of the associated star.

- We can also determine star radiation according to the Boltzmann law,

$$E = \sigma T^4,$$

where $E$ is the radiation of a star per unit area and per unit time, $\sigma$ is the Stefan-Boltzmann constant, and $T$ is the temperature.

Since $E$ varies as $T^4$, a small increase in $T$ makes the star much brighter (it gives out more radiation).

- Spectral types:

  Hottest to Coolest: **OBAFGKM**

  - O: $30000K - 60000K$
  - B: $10000K - 30000K$ (bluish white)
  - A: $7500K - 10000K$
  - F: $6000K - 7500K$
  - G: $5000K - 6000K$ (yellowish, e.g., Sun)
  - K: $3500K - 5000K$
  - M: below $3500K$ (reddish)

  Missing from this standard hierarchy are R/N stars, which are very cool stars, e.g., carbon stars, and S stars, which are red stars of very low temperature (about 3000K). Furthermore, a number range [0, 10) usually follows the labels O, B, A, etc., to indicate finer detail in each type, for example, A10 is equivalent to B0 and A0 is equivalent to F10.
• Color Index:
The color of stars is usually measured by a set of filters, typically the standard UBV filter set. The UBV system measures apparent magnitudes for ultraviolet (U), blue (B), and visual (V) electromagnetic spectral ranges. This measurement process is called three-color photometry.

The difference $B-V$ is called the color index. Noting that magnitude is smaller in value for larger brightness, we note that a hot star, which emits bluer radiation, has a smaller magnitude value for $B$, and so a more negative $B-V$ color index value. In general, the $B-V$ color index ranges from about $-0.3$ (hottest stars) to $+2.0$ (coolest stars), e.g.,

<table>
<thead>
<tr>
<th>Class</th>
<th>B-V</th>
<th>appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>-0.29</td>
<td>Red</td>
</tr>
<tr>
<td>A0</td>
<td>0.00</td>
<td>Orange</td>
</tr>
<tr>
<td>F0</td>
<td>0.31</td>
<td>Yellow</td>
</tr>
<tr>
<td>G0</td>
<td>0.59</td>
<td>Yellowish</td>
</tr>
<tr>
<td>K0</td>
<td>0.82</td>
<td>White</td>
</tr>
<tr>
<td>M0</td>
<td>1.41</td>
<td>Blue</td>
</tr>
</tbody>
</table>

Note: Astronomers sometimes use color indices such as $U-B$ (ultraviolet minus blue).

• Hertzsprung-Russell Diagram (H-R Diagram):

This diagram is a plot of temperature (temperature, spectral type, etc.) versus brightness (visual brightness). We can see from these plots that there are five different classes of stars (I-V):
I: Bright supergiants / supergiants
II: Bright giants
III: Giants
IV: Subgiants
V: Main sequence (Dwarfs)

Other classes not belonging to these five include the white dwarfs, which are extremely dim stars.

9.3 Stellar Database Structures

Due to the fact that there are large numbers of data records (stars) in three-dimensions, instead of rendering all of them every time, we need a data organization structure for flexible spatial scaling, for rapid data access and pruning, as well as for improving realism in appearance.

Promising data-indexing approaches that have seen heavy use in large multimedia database problems include the KD-tree [16], the R-tree [77], the $R^+$-tree [150], the $R^*$-tree [11], the VP-tree [181, 31], the MVP-tree [23], and the M-tree [32]. We found the VP-tree and M-tree to be most natural because of their organization relative to spherical structures surrounding a moving astronomical viewpoint.

In this section, we first introduce the conventional Vantage-Point (VP) and Metric (M) tree structures from the database literature, and then customize the tree structure for star rendering by incorporating scaling information and physical star characteristics into the tree. Furthermore, two pruning rules making use of the tree structure are derived: large-scale difference pruning and visibility pruning.
Introducing the Tree structures

In this subsection, we present the organization structures of the VP-tree and M-tree and the method for building these trees given a set of three-dimensional data points (stars). One crucial advantage of this two tree structures is that both of them employ spheres as their building blocks, which makes it possible for us to devise various efficient pruning techniques. Furthermore, since the VP-tree and M-tree structures are well-known for fast indexing performance in multimedia database applications, we are able to carry out spatial queries on stars efficiently; for example, we can easily find the N-nearest stars to the current viewpoint as well as performing a range search for stars around a certain three-dimensional position. Efficient spatial query algorithms for the VP-tree and M-tree have been discussed by Yianilos and Chiueh [181, 31] and by Ciaccia et al. [32], respectively.

(A) The Vantage-Point Tree.

Building the Vantage-Point Tree

In general, the Vantage-Point tree partitions the data space using spheres. Given a set $S$ of $N$ stars, we first let the star chosen as the root node be $v$ (the vantage point). Then, we can compute the median of distances from all the other stars to $v$, say $\mu$, and make use of this $\mu$ as the partition radius to divide $S$ into two approximately equal subsets $S_1$ and $S_2$.

![Figure 9.4: Partitioning Mechanism of the Vantage-Point Tree.](image)
Figure 9.4 illustrates this partition mechanism:

\[
S_1 = \{ s \in S | \text{distance}(s, v) \geq \mu \}
\]
\[
S_2 = \{ s \in S | \text{distance}(s, v) < \mu \}
\]

Note that in our implementation of the VP-tree, the star at \( v \) goes into \( S_2 \), whereas some database literature may exclude the data point at \( v \) from both \( S_1 \) and \( S_2 \). Furthermore, we recursively apply this partitioning mechanism on \( S_1 \) and \( S_2 \) until the number of stars in the divided subset falls below a certain limit (\( \leq k_{\text{leaf}} \)). In this way, we can obtain a hierarchical structure such that each internal node is a sphere and each leaf node contains at most \( k_{\text{leaf}} \) stars. Notice that using the median as the partition radius helps to make the VP-tree more balanced.

Finding the Vantage Point

The conventional algorithm for finding the vantage point \( v \) in a set of stars \( S \) is presented in Table 9.2 below:

1. Randomly pick a set \( C \) of candidate vantage points in \( S \)
2. Randomly pick a sample subset \( S' \) in \( S \)
3. For each candidate vantage point \( c \in C \)
   Compute the distance values from \( c \) to each star in \( S' \)
   Compute the mean and the standard deviation of these distance values
   End
4. The vantage point \( v \) for partitioning \( S \) is the candidate vantage point with the largest standard deviation.

Table 9.2: Algorithm for Finding a Vantage Point.

It has been shown in the database literature that, in practice, it is not necessary to find the vantage point with the optimum standard deviation: a vantage point with near maximum
standard deviation is sufficient for the efficient construction of the VP-tree. Therefore, the number of candidate vantage points and the sample subset size are merely tradeoffs between the quality of the vantage point and the time needed to build the VP-tree.

**Branching Factor**

Furthermore, to make the VP-tree structure more feasible and efficient, we can allow the VP-tree to have multiple branches, say $k$, instead of just two. Thus, we compute $k - 1$ quantiles, $\mu_i$ for $i = 1$ to $k - 1$, and partition $S$ into $k$ approximately equal subsets instead. Such a VP-tree is called a $k$-ary VP-tree. Note that $k$ is basically the maximum branching factor and the number of branches can vary from 2 to $k$, in particular when the number of stars to be partitioned is less than $k$.

**(B) The Metric Tree.**

**The M-Tree Structure**

Similar to the VP-tree, the M-tree makes use of spheres as partitioning geometries in the hierarchical structure. However, unlike VP-trees, which partition data points both inside and outside spheres, the sphere geometry in an M-tree contains every descendental data point within the sphere spatially. For example, the sphere associated with the root node encloses all data points descended from the root.

Generally, there are two kinds of nodes (routing objects) in an M-tree: leaf nodes, which contain data points (maximum: $k_{leaf}$), and internal nodes, which contain leaf nodes or child internal nodes (maximum: $k_{root}$). In [32], the following are defined:
9. Modeling and Rendering Stars

<table>
<thead>
<tr>
<th>$O_r$</th>
<th>a routing object</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r(O_r)$</td>
<td>covering radius of $O_r$</td>
</tr>
<tr>
<td>$P(O_r)$</td>
<td>parent node of $O_r$ (NULL if $O_r$ is root)</td>
</tr>
<tr>
<td>$d(O_r, P(O_r))$</td>
<td>distance of $O_r$ from $P(O_r)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$O_j$</th>
<th>a data object (e.g. star) in leaf node</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(O_j)$</td>
<td>parent leaf node of $O_j$</td>
</tr>
<tr>
<td>$d(O_j, P(O_j))$</td>
<td>distance of $O_j$ from $P(O_j)$</td>
</tr>
</tbody>
</table>

Note that both leaf nodes and internal nodes are routing objects, but leaf nodes are routing objects that contain data objects whereas internal nodes are routing objects that contain child routing objects; note also that distances $d(O_r, P(O_r))$ and $d(O_j, P(O_j))$ can be pre-computed and stored at the parents $P(O_r)$ and $P(O_j)$, respectively, so that we can avoid computing some distance terms while performing data indexing or spatial queries.

**Building the M-Tree**

Like other dynamic balanced trees such as R-trees [77], M-trees grow in a bottom-up manner. That is, data objects are sequentially inserted into the tree at the leaf nodes; whenever the number of data objects in a leaf node exceeds the limit, $k_{\text{leaf}}$, we split the associated leaf node into two leaf nodes and enter them into the parent of the original leaf node. Furthermore, if the number of leaf nodes the parent node contains exceeds another limit, $k_{\text{route}}$, we split the parent node into two, and so on, until we arrive at the root. Therefore, the M-tree grows from the bottom.

Altogether, there are four essential algorithms included in [32] regarding the building of the M-tree structure:

1. **Insert**:
This algorithm recursively goes down the M-tree and picks up a leaf node for inserting the incoming data object.

2. **Split:**
This algorithm recursively splits routing nodes from leaf to root (if necessary) in case of overflow.

3. **Promote:**
Whenever a split happens, the algorithm chooses two data objects (or routing objects in case splitting happens at internal nodes) as representatives so that the two split regions (the two spheres) have “minimum volume” and “minimum overlap.”

4. **Partition:**
This algorithm distributes data objects (or routing objects) between the two chosen representatives.

Details for these algorithms such as pseudo-code and in-depth descriptions can be found in the original M-tree paper [32] and associated source code by the author can be downloaded from http://www-db.deis.unibo.it/Mtree/index.html.

**Parameters**

There are basically four parameters when building an M-tree:

1. $k_{leaf}$ — maximum number of data objects a leaf node can contain.

2. $k_{route}$ — maximum number of routing objects an internal node can contain.

3. “promote” criterion — $m_{RAD}$, $mM_{RAD}$, $MLB_{DIST}$, RANDOM, or SAMPLING (refer to Subsection 4.1 in [32]).

4. “distribution/partition” criterion — Generalized Hyperplane, or Balanced (refer to Subsection 4.2 in [32]).
Both VP-tree and M-tree structures support efficient spatial queries:

1. **Range Query** \((Q, R)\)
   - find all data objects (stars) within radius \(R\) from point \(Q\).

2. **Nearest-Neighbor Query** \((Q, N)\)
   - find the \(N\) nearest data objects (stars) from point \(Q\).

Methods for implementing these queries can be found in the corresponding papers [181, 31, 32]. In general, using these spatial queries, we can increase the richness of the virtual astronomy application, which allows users to identify nearby stars while navigating inside the enormous stellar space. Furthermore, we can make use of range queries to quickly determine the stars nearest to the current viewpoint and to create a three-dimensional map of stars around the current navigation region.

**Customizing the Trees for Star Rendering**

In this subsection, we discuss how we adapt the conventional VP-tree and M-tree for star rendering by incorporating the scaling and rendering information into the internal nodes of the tree structures. With this information incorporated, we are able to devise efficient pruning and large-scale handling methods during the rendering.

*Using Power Homogeneous Coordinates.* Due to the large variation in spatial scale, storing distances and vectors in a conventional manner may result in floating point number overflow or inaccuracy during calculations. We therefore utilize power homogeneous coordinates for all spatial computations (see Chapter 3).
Besides vectors and coordinates, distances have to be stored and computed in $\log_{10}$ scale. For example, in the VP-tree, radii (scalars) of partitions stored in internal nodes are in $\log_{10}$ scale and coordinates of vantage points are represented using power homogeneous coordinates as well. Similarly, in the M-tree, precomputed distances, $r(O_r)$, $d(O_r, P(O_r))$, and $d(O_j, P(O_j))$, are in $\log_{10}$ scale whereas coordinates of $O_r$ and $O_j$ are represented using power homogeneous coordinates. Coordinates of stars (data objects) as well as the viewing position are also represented using power homogeneous coordinates.

**Storing Star Information.** Besides incorporating power homogeneous coordinates into the trees, we incorporate star rendering information in the trees as well. This is realized in a bottom-up manner. Information of individual stars is stored under the leaf nodes; rendering information includes coordinates (in power homogeneous coordinates), B-V color index, absolute luminosity, etc. In addition to the VP-tree or M-tree related parameters responsible for the tree structure, since internal nodes are clusters of stars, rendering parameters of star clusters are stored in internal nodes as well: these rendering parameters can be computed based on the parameters of the child:

\[
lum_{\text{node}} = \sum_i \lum_{\text{child}_i}
\]
\[
weight_i = \frac{\lum_{\text{child}_i}}{\lum_{\text{node}}}
\]
\[
color_{\text{node}} = \sum_i \color_{\text{child}_i} \times weight_i.
\]

This aggregate rendering information can be used for rendering the star clusters when a certain pruning rule criterion holds (see the next section). Note that $\lum$ and $\color$ refer to luminosity and color of a star; methods for computing these values for individual stars and how to render a star based on these parameters will be discussed in the next section. Nevertheless, the main idea here is that based on a bottom-up strategy, we are able to compute the aggregate luminosity and color for star clusters from the leaves up to the root.
Pruning Rules for Rendering

After equipping VP-trees and M-trees with the above information, we are ready to render stars by traversing the tree from the root to the leaves. Given the nice spherical property, it is usually not necessary to render every data point (star) stored in the tree. The following pruning rules are devised to speed up the rendering process.

A. Large-Scale Difference Pruning. The first pruning rule is based on difference in large spatial scale. Since each internal node is spherical, we can easily compute its visual angle ($\alpha_{\text{cluster}}$) subtended at the viewpoint by simple trigonometric calculation.

Let $r$ and $\hat{d} = (\hat{d}, d_s)$ be the $\log_{10}$ radius of the star cluster and the vector (in normalized power homogeneous coordinates) from the viewpoint to the star cluster center (internal node), respectively. Thus, based on simple trigonometry depicted in Figure 9.5, we obtain

$$\alpha_{\text{cluster}} = 2 \sin^{-1}(10^{r-d_s}).$$

In this way, if $\alpha_{\text{cluster}}$ is smaller than $\theta_{\text{pixel}}$, which is the angular size of a pixel or radians per pixel of the screen (see Subsection 6.2), the whole star cluster is so far away from the viewpoint that it is smaller than a pixel. Thus, there is no reason for us to go into this subtree; and instead, we can render the whole star cluster as if it is a single star based...
on the aggregate rendering parameters associated with the star cluster. Note that we can simplify the inequality as follows:

\[
\theta_{\text{pixel}} > \alpha_{\text{cluster}} \\
\log_{10}(\sin \frac{\theta_{\text{pixel}}}{2}) > r - d_s \\
d_s > r + \log_{10}(\sin \frac{\theta_{\text{pixel}}}{2}).
\]

Here, \(\log_{10}(\sin(\theta_{\text{pixel}}/2))\) can be pre-computed given a certain field of view and screen resolution; furthermore, if \(d_s\) is greater than this value plus \(r\), the whole subtree can be pruned. We can generalize this criterion by allowing subtree pruning when the whole star cluster is less than 1.5 or 2 pixels wide. Essentially, this method is the same as the object disappearance criterion in Chapter 7, but effectively, we are applying the criterion recursively inside the tree structure, not just to a single object. Furthermore, even when the criterion holds, instead of ignoring the whole star cluster, we may render it as a single star based on the aggregate luminosity parameter, unless the combined luminosity is still too low to make the cluster bright enough to be visible.

**B. Visibility Pruning.** Secondly, we can prune star clusters that are not visible at the current viewing position. Again, this pruning rule is derived from the spherical nature of the VP-tree and M-tree. Consider the viewing configuration shown in Figure 9.6, where the maximum visual angle subtended at the viewpoint is related to the diagonal of the image plane. We define such a visual angle as \(\alpha_{\text{view}}\). Mathematically, we can compute \(\alpha_{\text{view}}\) from the field of view in \(x\) (\(\text{fovx}\)) and in \(y\) (\(\text{fovy}\)) using the following equation:

\[
\alpha_{\text{view}} = 2 \tan^{-1} \left[ \left( \tan^2 \left( \frac{\text{fovy}}{2} \right) + \tan^2 \left( \frac{\text{fovx}}{2} \right) \right)^{\frac{1}{2}} \right].
\]
Figure 9.6: Note that $\alpha_{\text{view}}$ is the angle subtended by the diagonal of the image plane at the viewing position.

Figure 9.7: We can prune the star cluster when the sum of the two visual angles, $\alpha_{\text{cluster}}$ and $\alpha_{\text{view}}$, is smaller than $2 \theta$. 
Consequently, we can determine the angle, say \( \theta \), between the viewing direction and the vector \( \vec{d} \) to the star as depicted in Figure 9.7. In this way, if

\[
\alpha_{\text{cluster}} + \alpha_{\text{view}} < 2 \theta,
\]

the visual cone of the star cluster is guaranteed not to intersect with the viewing frustum. As a result, the star cluster is ensured not to be visible and can be pruned. One important precaution when using this rule (and the previous rule) is that the viewing position is assumed to be outside the star cluster (associated sphere), that is \( d_s > r \), or else the mathematics may fail as \( \alpha_{\text{cluster}} \) becomes meaningless.

C. Pruning k-ary VP-Trees. When dealing with a k-ary VP-Tree, as a speedup, we can perform the two pruning rules from outer branches (outer shell) to inner branches (inner shell). As demonstrated in Figure 9.8, if an outer shell is prunable, all the enclosed inner shells are also prunable; thus, the pruning procedure can be accelerated.

9.4 Photorealistic Rendering of stars

In the previous section, we described our database structure for handling star datasets, or collections of stars; in this section, we focus on the rendering procedure for a single star, including consideration of the relationship between star size and apparent magnitude, star resizing, star modeling, star coloring, etc. We present the rendering approach step by step in the following subsections.
9. Modeling and Rendering Stars

Figure 9.8: Perform pruning from outer to inner branches of a k-ary VP-tree: Both pruning rules are applicable.
Measuring Star Size against Apparent Magnitude ($V_{mag} \rightarrow pixel – unit$)

To render stars realistically in the night sky, the very first step is to find an appropriate and well-defined method to resize stars. Stars of different magnitudes should be rendered with different on-screen sizes so that the night sky is more realistic and attractive, and so people can distinguish various constellations in the sky naturally. Some pioneers in astronomy attempted to characterize the appearance of stars on film plates, and to classify star sizes accordingly [105]; we will need similar information for our star models. For photorealistic rendering, as in image-based rendering, our approach begins with real photographs showing a starry sky in such a way that we can extract star size information from these real photographs. There are four steps in our method:

1) Selecting Photographs
We first collect photographs showing different parts of the night sky. To make the following steps successful, these photographs should have similar exposure, color balance, and preferably be from the same imaging device, so that stars of similar magnitude appear to be of more or less the same size in the photographs. Typically, the Akira Fujii/DMI collection (by David Malin, dfm@aoepp.aao.gov.au) were used in our experiments: this collection can be reached at http://davidmalin.com/fujii/fujii_index.html.

2) Star Registration.
The next step is to carry out registration between the real photograph and the star database in order to establish correspondences between image locations on the photographs and the three-dimensional directions to stars in the catalogue. To make this procedure more efficient, we developed an image alignment tool to position images against the night sky. Figure 9.9 shows the program interface with a working example: aligning a photograph
showing the constellation Orion.

![Star registration interface](image)

Figure 9.9: Star registration interface.

In the program, we can pinpoint locations on the real photograph by clicking on the picture at the lower-left corner, and then we can pick up corresponding stars in the simulated star field. The yellow border in the star field shows the corresponding position of the photograph after the registration. Mathematically, we define \((u, v, w)\) and \((x, y, z)\) to be the image coordinates (in homogeneous coordinates) and the star coordinates, respectively, and so, by division, we can find the actual two-dimensional image coordinate: \(U = u/w\) and \(V = v/w\). Now, we can define \(M\), a three-by-three matrix, to be the transformation between the image and star coordinates:

\[
\begin{pmatrix}
  u \\
  v \\
  w
\end{pmatrix} = M \begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix} \quad \text{and} \quad \begin{cases}
  U = u/w \\
  V = v/w
\end{cases}
\]
Figure 9.10: Locating stars on the Orion photograph.
Note that we use homogeneous coordinates in the image space so that \((x, y, z)\) maps to the same image coordinate \((U, V)\) regardless of any scaling of \((x, y, z)\). In a word, this formulation can be understood as a perspective projection into the image space. After locating three or more correspondences by hand using the interface, we can solve for \(M\) in the equation above. In this way, given the three-dimensional location of any star known to be in the photograph, we can determine its location on the photograph easily. Figure 9.10 marks locations of some bright stars on the photograph.

3) Extracting Stars from Photographs.

After the previous step, we can automatically locate a number of stars in the Orion photograph. Now, by jittering variable-sized Gaussian filters around these image locations, usually within one or two pixel units, we can extract sub-images of these stars from the photograph.

![Figure 9.11: Plotting extracted star sub-images: apparent magnitude against B-V.](image)

Figure 9.11 plots sub-images of stars extracted from three different real photographs: the horizontal axis is the apparent magnitude (visual brightness), which increases to the left, whereas the vertical axis is the B-V color index, which increases downward. The
brightest star visible in the middle plot is Sirius in the constellation Canis Major; it is one of the brightest stars in the night sky.

4) Formulating the Equation: Star Size versus Apparent Magnitude.
The last step is easy; we only need to create a table of pixel size against apparent magnitude to be used in the rendering process. Nevertheless, when constructing such a table, we have to pay attention to the fact that different photographs could be captured with different fields-of-view. Thus, stars of the same magnitude could appear with different sizes on different photographs.

In this way, we need to normalize pixel sizes with respect to the corresponding field-of-view. Interestingly, as careful readers may notice, at the time we register real photographs against the star field, it is possible to determine an approximate field-of-view associated with the real photograph if the associated field-of-view is unknown beforehand. Furthermore, due to the fact that the intensity decreases gradually around the stars, there is no clear boundary for stars in the images. Thus, when we fit Gaussian filters to measure star sizes, these filters have to enclose at least 90% of the pixel intensity in the star sub-images so that we know how much to scale the star model during the rendering.

Star Resizing Method

For rendering, we need to acquire two basic star properties from the Hipparcos Catalogue: luminosity and spectral type. Spectral type can be obtained directly from the catalogue, whereas luminosity can be computed using Equation (9.2). Note that star magnitudes cannot be summed up right away because it is non-linear with respect to radiance. Instead, since luminosity is a measure of radiation energy, this quantity is additive; thus, we can compute the aggregate luminosity of star clusters, and store the sum in internal nodes. When our viewpoint moves to a location in space other than the Earth, with star position
\((d, d_s, ((d, d_s)\) is a normalized power homogeneous vector from the viewpoint to the star), we can re-arrange Equation (9.2) to compute the apparent magnitude of the star (or a star cluster) with respect to the current viewpoint:

\[
\log_{10} L = -2 + 2 \log_{10} \text{(distance in parsec)} - 2.5 \, m
\]

\[\therefore m = \frac{2}{5} \left( 2 d_s - 2 - \log_{10} L \right).\]

This \(m\) is the apparent magnitude of the star observed at the current viewing position. It can be used to look up the star model information in the table we established in the previous subsection. As a result, when we navigate within the three-dimensional star field, we are able to precisely adjust star sizes as we move toward or away from them. In the implementation, for efficient computation, we store \(\frac{2}{5} \log_{10} L\) instead of just \(L\) for each star as well as for each star cluster. Figure 9.17 simulates changes in star size and appearance while we are moving towards the Pleiades.

**Star Modeling Method**

Being able to adjust star sizes in the sky, our next step is to model stars geometrically in the virtual environment. In practice, there are three modeling approaches:

1. **Billboard textured square**

   Similar to the rendering approach used for the Sun in the Solar system (see Section 8.4), we can model each star as a billboard square, always facing the viewing position. Then, a chopped-off Gaussian function can be used as the image texture mapped onto the square. In addition to the Gaussian distribution, other image textures such as a Fourier-transformed point source, single star photographs, etc. can be employed.
2. **Anti-aliased point**

The second modeling approach is simply to render stars as anti-aliased points. In practice, OpenGL offers a range of discrete point sizes depending on the hardware implementation. To simulate a continuous size range, we use alpha blending, and adjust the alpha value for each star according to the ratio of the desired star screen coverage to that offered by OpenGL. Lookup tables make this computationally efficient.

3. **Mixed Representation**

Finally, we can define a certain limit in pixel units such that when the lookup table of a star pixel size is below this limit, the star is rendered as an anti-aliased point. Otherwise, the star is rendered as a textured billboard with size compatible with the original anti-aliased point. Typically, the threshold star diameter is chosen to be 3 to 4 pixels, depending on the screen resolution. In practice, to smooth the transition between the two representations, we define a tuning parameter to scale billboard polygons in such a way that the star sizes from the two renderings appear to be more or less the same. The reason for this tuning process is that the Gaussian image mapped onto the square occupies only the central region of the square. Thus, we need to scale the square to fit the corresponding anti-aliased point perceptually.

### Star Coloring Method

We have two modes for coloring stars: using spectral type or $\log_{10}(distance)$. Based on precise measurements, Mitchell N. Charity derived the color map available at the website: [http://www.vendian.org/mncharity/dir3/starcolor](http://www.vendian.org/mncharity/dir3/starcolor), where the coloring model can be regarded as a mapping function from spectral type to RGB color values.
Figure 9.12: Colormap: from $\log_{10}(\text{meter})$ distance to star color (the further away from the Earth, the larger the red shift and thus, stars (or galaxies) become redder in color).

Besides using spectral type to make the coloring more realistic, we can use the $\log_{10}(\text{distance})$ as the coloring parameter to provide visual cues. As shown in Figure 9.12, we can color stars (or galaxies when appropriate) redder when they are farther away and bluer when they are closer. This method was used in the Cosmic Clock animation [80]; and a description on this animation can be found in Chapter 16.

Note that after we determine colors of individual stars based on spectral type or distance, we can also determine colors of star clusters (see Subsection 9.3). On the other hand, it is possible that we can study the color of stars in the same way as star sizes. However, since the provided color table was found to be of high quality compared to real photographs, we omit this analysis.

**Star Rescaling Method**

Due to the depth precision problem introduced by the near and far planes, as discussed in Chapter 2, we have to keep stars between the near and far planes so that we can have appropriate depth buffering. Therefore, in order to render stars at different scales, stars beyond the far plane have to be projected in front of the far plane so that their scales are within the renderable region. This is implemented using the depth rescaling method proposed in Chapter 4.

Technically, we have to fix our viewpoint in *OpenGL* at the world origin, that is the
same as undoing any translation in the current Modelview matrix before we render stars. Then, when we are ready to render stars (or star clusters), we can make use of the distance vector \( d \), which is computed during the pruning phase, to check if any star is located beyond the far plane. For example, if \( d_s \) is greater than the \( \log_{10} \) of the far plane distance, we can project the related star into the far safety region and rescale it according to the depth rescaling model. On the other hand, we can directly render the star at the rescaled value \( \tilde{d} \times 10^{d_s} \) in the linear region as might be expected. By using the depth rescaling method, we can ensure correct depth buffering even when stars are rendered at widely varying scales.

**Brightness Adjustment**

Within the rendering framework for stars, we can see that the star brightness, i.e., size and intensity, should be physically fixed and not controllable. However, under different display environments such as a single screen, the CAVE, a dome-based planetarium, etc., it is likely that we will need some way to adjust star brightness to fit the specific display conditions. Therefore, within the proposed framework for star rendering, we provide three ways to control brightness.

1. **Direct Star Size Adjustment**

   This parameter adjusts star size by linearly scaling the pixel sizes obtained from the lookup table. Note that this table maps apparent magnitudes to pixel units (see Subsection 9.4).

2. **Luminosity Adjustment**

   Another way to adjust star brightness is by scaling the luminosity term. We can have a tuning parameter, say \( \alpha_{\text{lum}} \), such that,

   \[
   \text{lum} = \text{lum}_0 \cdot 10^{\alpha_{\text{lum}} \cdot \text{mag}} \quad \text{OR} \quad \text{lum}_0^{\alpha_{\text{lum}}}.
   \]
where $l_{um0}$ is the intrinsic luminosity of a star.

3. Gamma Correction

This is the standard gamma correction that fine-tunes the color response curve when showing stars on different display hardware. Note that different display hardware will have different gamma values, and that each color must be separately adjusted.

**Performance Issues**

**Bucketing Stars:**

When walking through the tree structure, we encounter stars to be rendered. Since star size is independent of the traversal sequence, we typically need to change graphics states such as point size for each star. However, in *OpenGL*, such graphics state changes are expensive. To avoid unnecessary point-size switches during the traversal, we sort stars into size buckets, one for each of a range of discrete anti-aliased point sizes offered by *OpenGL*, and invoke the graphics system to draw them whenever a bucket fills. In our implementation, the bucket size is adjusted according to the size of the dataset (method courtesy of Stuart Levy).

**Rendering Faint Stars:**

As the dataset size increases, the number of stars that are too dim to be barely renderable on the screen (one or half a pixel in size) also increases. Instead of throwing all these faint stars away, we should render some of them to make the night sky richer and more realistic. To accomplish this, we define a faintness value, $c\%$, so that we can randomly pick $c\%$ of these faint stars to be rendered with the minimum point size and minimum alpha value. In the implementation, a hashing function is pre-computed to select a percentage of faint stars with a specific faintness value (method courtesy of Stuart Levy).
9.5 Rendering Results

In order to experiment with star renderings and navigation from the Earth up to stellar scale, we developed a star rendering program called "starview," using OpenGL as the rendering environment. Thus, it can be run on various platforms supporting OpenGL. In this subsection, we illustrate the use of this program to produce the following figures.

- Rendering stars as anti-aliased points or textured polygons (Figure 9.13).
- Adjustment of star brightness: the constellation Leo (Figure 9.14).
- Increasing the apparent magnitude limit: progressively adding more stars to the simulated night sky (Figure 9.15).
- Various Constellations: name (yellow), boundary (green), and connecting lines (yellow) (Figure 9.16).
- Navigating at the stellar scale – moving towards Pleiades: notice the changes in star size as we move closer to the stars (Figure 9.17).
- Powers-of-10 transition from the Solar system scale to the stellar scale (Figure 9.18).
- Visualizing positional errors, $\Delta d$ (see Subsection 9.2): Red bars denote the error, which increases for stars further away from Earth. The cubes mark the spatial scale (Figure 9.19).
- Visualizing space velocity of stars (trail length is set to 20,000 years) (Figure 9.20).
- Displaying orbits of extra-solar planets as rings (Figure 9.21).
- Rendering stars in the presence of other data sets at various scales (Figure 9.22).
Photorealistic rendering – comparing real photographs and our simulations. Note that the Hipparcos catalogue does not have all the stars we can see in the photographs ((c) AAO/ROE, photographs courtesy David Malin) (Figure 9.23).

Figure 9.13: Rendering stars as textured polygons (left) or anti-aliased points (right).
Figure 9.14: Adjustment of star brightness: the constellation Leo.
Figure 9.15: Increasing apparent magnitude limit: progressively adding more stars to the simulated night sky, looking at Taurus (left) and Pleiades (right).
Figure 9.16: Various Constellations: name (yellow), boundary (green), and connecting lines (yellow).
Figure 9.17: Navigating at the stellar scale – moving towards Pleiades: notice the changes in star size as we move closer to the stars.
Figure 9.18: Powers-of-10 transition from the Solar system scale to the stellar scale.
Figure 9.19: Visualizing positional errors, $\Delta d$ (see Subsection 9.2): Red bars denote the error, which increases for stars further away from Earth. The cubes mark the spatial scale.
Figure 9.20: Visualizing space velocity of stars (trail length is set to 20,000 years).
Figure 9.21: Displaying orbits of extra-solar planets as rings. Note that we have to scale the orbit length by 50,000 to make them visible. We zoom in to two stars with extra-solar planets in the middle row. Also, the sphere shown in the last row has size about 50 light years, which is about the time since we first sent out radio waves from the Earth.
Figure 9.22: Rendering stars in the presence of other data sets at various scales. Note that the Heliopause is the shock wave resulting from the Solar wind and Loop I is a gas cloud produced by a supernova explosion in the Scorpius association.
Figure 9.23: Photorealistic rendering – comparing photographs and our simulations: Orion (top), Big Dipper (middle), and Canis Major (bottom). Note that the Hipparcos Catalogue does not have all the stars we can see in the photographs ((c) AAO/ROE, photographs courtesy David Malin).
9.6 Conclusion and Future Work

In this chapter, we explored interactive rendering of large numbers of high-quality stars for virtual astronomy applications by using a scalable VP-tree or M-tree to store the star dataset in a hierarchical structure. Since we make use of the power homogeneous coordinates to carry out computations in log scale, we can safely avoid floating point errors resulted from large spatial scale ranges. Furthermore, exploiting the spherical partitioning mechanism using the tree structures, we are able to derive various pruning rules to speed up rendering performance by pruning too small or invisible star clusters. To achieve realism in the rendering results, a photorealistic approach for rendering stars is proposed; it is based on star sizes and colors determined by studying real photographs. At the same time, to augment the rendering performance, we adopt a bucketing method for rendering anti-aliased points and a faintness measure to avoid the rendering of stars that are too dim.

In our future work, we would like to look at the MVP-tree structure, which is a VP-tree with multiple vantage points (or a mixed M-tree and VP-tree), as well as to investigate whether introducing multiple vantage points at the same tree level (using the MVP-tree) can speed up the rendering. We also plan to study the effect of various rendering parameters such as the faintness measure, and to investigate how the depth of a tree affects the rendering performance. In addition, we would like to study how we can provide a frame rate guarantee even if the dataset is too large in size. Furthermore, since the VP-tree and M-tree are distinguished by their fast indexing capability, using an $N$-nearest-stars search and range query is possible during real-time rendering, which would be a great geometrical asset to the visualization process.
Modeling and Rendering the Milky Way Galaxy

10.1 Background

On clear, moonless nights during the summer, if we look toward the Scorpius and Sagittarius regions in the night sky, we can see a faint band of light extending across this area. It is known as the Milky Way Galaxy, or the Silvery river in Chinese. For thousands of years, no one knew exactly what the Milky Way was; however, when Galileo pointed the newly invented telescope towards this band of light in the early 17th century, he discovered numerous previously unseeable stars in that region, and eventually people realized that our Sun is merely one of the billions stars in the huge structure.

The Milky Way Galaxy is a flat disk, consisting of billions of stars, along with molecular clouds, dust clouds, and nebulae, held together by gravity; at the center, a giant black hole is suspected to exist. Closely orbiting around the disk are some ancient star clusters, known as globular clusters, in the form of spherical balls surrounding our Milky Way Galaxy. There are about 150 - 200 globular clusters, each has about $10^5$ stars. By studying the spatial distribution of these globular clusters in 1920, Harlow Shapley estimated the size of the
Milky Way Galaxy and the Sun's distance from the Milky Way center, which are about 15 kpc and 8.5 kpc, respectively (from [102]). We note also that the direction to the galactic center is at RA=17h45.7m and DEC=−29°00' (year=2000) in the Sagittarius region.

On the other hand, nothing is at rest in the Milky Way Galaxy; everything, including stars and gas clouds, is orbiting around the Milky Way center so that the whole physical structure is dynamic. Typically, the Sun takes about 240 million years to finish one round-trip around the Galaxy at a speed about 220 km/s. Statistics on the Milky Way can be found in Table A.2 of the Appendices. Note that as in common practice, we use the term “Galaxy” with a capital “G” to refer to the Milky Way Galaxy, and “galaxy” with a lowercase “g” to refer to any stellar system, for example, galaxies outside our Milky Way Galaxy. We will look at galaxies in the next chapter.

In this chapter, there are five sections: The first section introduces basic knowledge about the Milky Way Galaxy including datasets, coordinate systems, and related physics and mathematical models. The next section presents panoramic views of the multi-wavelength Milky Way Galaxy as seen from the Earth at Solar system scales. The third section takes us to the stellar scale (scale of the Hipparcos Catalogue) and looks at the panoramic view of the Milky Way with Hipparcos stars removed. The fourth section presents a three-dimensional model of the Milky Way Galaxy based on our particle system model. Finally, we draw conclusions and discuss possible followup work in the last section.

Datasets

To model and render the Milky Way Galaxy, we have explored the following datasets:

- The Optical Night sky: Mellinger image [119]
  This is a two-dimensional all-sky panoramic image showing the visible (optical) appearance of the entire night sky. The image is oriented along Galactic l and b. The...
highest available resolution at present is $14K \times 7K$ (in pixel units), along $l$ and $b$.

- Multi-wavelength all-sky images [119, 73, 85, 45, 157, 147, 40, 86, 44, 88, 64, 111, 87, 89]

In addition to the Mellinger image, which corresponds to the optical wavelength spectrum, we gathered panoramic images for various other ranges in the electromagnetic (EM) spectrum: Gamma, hard X-ray, soft X-ray, ultra-violet, infra-red (dust), radio-wave, HI-data, etc.

- The Wainscoat/Dwek G2 Galactic bulge model (or the Goddard Milky Way model) (from NASA) [172, 49]

This is a point-based representation of the three-dimensional Milky Way Galaxy produced by Monte Carlo sampling, in which there are two versions: one with 40,000 data points and one with 4,000 data points. The model itself is stored in VRML (Virtual Reality Modeling Language) format.

Details about these datasets can be found in Appendix A.3. Other related datasets to be explored in the future include globular cluster catalogs, nebula catalogs, etc.

**Coordinate System**

In the Milky Way Galaxy, position in the sky is usually expressed using the Galactic coordinate system. We assume the system's origin to be located at the Earth. Since the Earth-Sun distance is negligible at the Galactic scale, it does not make any difference in terms of data precision if we put the origin at the Sun or the Earth. The direction to Galactic center is the Galactic x-axis and the normal to the galactic plane is the Galactic z-axis, namely the North Galactic Pole (NGP). Details and transformations related to this coordinate system can be found in Appendix A.4. Note that galactic longitude ($l$) is $0^\circ$ at the galactic center and increases anti-clockwise around the NGP, viewed from the Earth.
Physics and Mathematics

**Coordinate Conversion** \((u, v) \rightarrow (l, b) \rightarrow (x, y, z)\)

(1) **Image coordinates** \((u, v)\) to **Galactic coordinates** \((l, b)\)

Most panoramic images we have obtained from the astronomy community have their image coordinates aligned with Galactic \(l\) and \(b\) so that the galactic center \((l = 0, b = 0)\) is at the center of the image. Figure 10.1 shows the relationship between image coordinates in panoramic images and galactic coordinates in detail. Here, we can define the following quantities:

- \((w, h)\) is the resolution of a panoramic image,
- \((u, v)\) is the image coordinate range \([0, w - 1]\) and \([0, h - 1]\),

and \((l, b)\) is the galactic coordinate range \((-180^\circ, +180^\circ)\) and \([-90^\circ, +90^\circ]\).

**Figure 10.1:** Image Coordinates \((u, v)\) and Galactic Coordinates \((l, b)\).
Then, we can compute \((l, b)\) based on \((u, v)\):

\[
\begin{align*}
I &= \frac{u + 0.5}{w} \cdot 360^\circ + 180^\circ \\
b &= \frac{v + 0.5}{h} \cdot 180^\circ - 90^\circ,
\end{align*}
\]

and vice versa:

\[
\begin{align*}
u &= (0.5 - \frac{l}{360^\circ}) w - 0.5 \\
v &= (0.5 + \frac{b}{180^\circ}) h - 0.5,
\end{align*}
\]

where the "+0.5" in \((u + 0.5)\) and \((v + 0.5)\) accounts for the shift in the pixel center along \(l\) and \(b\).

(2) Galactic coordinates \((l, b)\) to Three-dimensional direction \((x, y, z)\)

We can obtain three-dimensional directions in the Galactic coordinate system (see the Galactic coordinate system presented in Appendix A.4): from

\[
\begin{align*}
x &= \cos b \cos l \\
y &= \cos b \sin l \\
z &= \sin b,
\end{align*}
\]

and vice versa:

\[
\begin{align*}
l &= \text{atan2}(y, x) \\
b &= \arcsin z,
\end{align*}
\]

where the \text{atan2} function computes the principal value of the arc tangent of \(y/x\), using the signs of both arguments to determine the quadrant of the return value (from manpage).
The Aitoff Transformation

Sometimes all-sky panoramic images are depicted or stored in another way, through the Aitoff equal-area transformation: The all-sky image is squeezed into an elliptical shape. Figure 10.2 shows a rectangular \((l, b)\) grid and the corresponding grid after the Aitoff transformation.

Given \((l, b)\) as the galactic coordinate on the rectangular grid domain and \((x, y)\) as the image coordinate on the Aitoff domain,

where \(l \in [-180^\circ, +180^\circ]\) and \(b \in [-90^\circ, +90^\circ]\), and \(x \in [-180(\frac{2\sqrt{2}}{\pi}), +180(\frac{2\sqrt{2}}{\pi})]\) and \(y \in [-90(\frac{2\sqrt{2}}{\pi}), +90(\frac{2\sqrt{2}}{\pi})]\),

the forward Aitoff transformation from \((l, b)\) to \((x, y)\) is

\[
\begin{align*}
\text{RHO} & = \arccos \left( \cos(b) \times \cos(l/2) \right) \\
\text{THETA} & = \arcsin \left( \frac{\cos(b) \times \sin(l/2)}{\sin(RHO)} \right)
\end{align*}
\]
10. Modeling and Rendering the Milky Way Galaxy

\[
x = -4 \times \frac{180}{\pi} \times \sin(\rho/2) \times \sin(\theta)
\]
\[
y = 2 \times \frac{180}{\pi} \times \sin(\rho/2) \times \cos(\theta)
\]
If \( b > 0 \) then
\[
y = -y
\]

and the backward Aitoff transformation from \((x, y)\) to \((l, b)\) is

\[
Y = -y \div (2 \times \frac{180}{\pi})
\]
\[
X = -x \div (2 \times \frac{180}{\pi})
\]
\[
A = (4 - X^2 - 4 \times Y^2)^{0.5}
\]
\[
b = \frac{180}{\pi} \times \arcsin (A \times Y)
\]
\[
l = 2 \times \frac{180}{\pi} \times \arcsin \left(\frac{A \times X}{(2 \times \cos(b))}\right)
\]

Note that we intentionally set the signs so that \(x\) and \(y\) increase in the same directions as \(u\) and \(v\), respectively, though they may be negative due to the centering of the origin (from http://lambda.gsfc.nasa.gov/product/iras/coordproj.cfm).

**Differential Rotation of the Milky Way Galaxy**

By analyzing the space velocity of stars obtained from proper motion and radial velocity, astronomers already know that the Milky Way Galaxy does not rotate like a rigid body. The angular velocity varies with distance from the center: This is called the differential rotation of the Milky Way Galaxy. As depicted in Figure 10.3, stars near the center orbit faster (larger angular velocity) than those outside, so a spiral pattern results.

In 1920s, Dutch astronomer Jan H-Oort analyzed the mathematical nature of this differential rotation. Defining \(V\) as the tangential velocity and \(R\) as the corresponding distance
at the beginning million of years

Figure 10.3: The differential rotation of galaxies: rotation curve.

(radius) from Galactic center, he discovered two constants, namely the first and second Oort constants (or coefficients), say $A$ and $B$, respectively:

\[
\begin{align*}
A &= \frac{1}{2} \left( \frac{V_0}{R_o} - \frac{dV}{dR} \big|_{r=R_o} \right) \\
B &= -\frac{1}{2} \left( \frac{V_0}{R_o} + \frac{dV}{dR} \big|_{r=R_o} \right),
\end{align*}
\]

where $V_o$ and $R_o$ are $V$ and $R$ for our Sun, respectively.

Note that by re-arranging the above equations, we have

\[
\begin{align*}
A - B &= \frac{V_o}{R_o} = \omega_o \\
A + B &= -\frac{dV}{dR} \big|_{r=R_o},
\end{align*}
\]

which are the angular velocity $\omega_o$ and radial change rate of the tangential velocity $-\frac{dV}{dR}$ at our Sun, respectively. Modern values of $A$ and $B$ recommended by the International Astronomical Union (IAU) are $15 \text{ km s}^{-1} \text{ kpc}^{-1}$ and $-10 \text{ km s}^{-1} \text{ kpc}^{-1}$, respectively (from [102]).

Nevertheless, due to the non-uniform distribution of matter in the Milky Way Galaxy, the rotation curve, $V(R)$ (some papers use $\theta(R)$), is not strictly increasing with decrease in $R$ according to Keplerian Laws. Based on measurements in radio astronomy, astronomers obtained the rotation curve in Figure 10.4. Note that because matter at the Galaxy center is highly concentrated, the rotation within $4 \text{ kpc}$ behaves like a rigid body rotation.
10. Modeling and Rendering the Milky Way Galaxy

233

Figure 10.4: The rotation curve of our Milky Way Galaxy.

10.2 The Milky Way at Solar system scale: Panoramic images

We now return to our computer graphics concerns: modeling and rendering the Milky Way Galaxy. In this section, we present the Milky Way appearance corresponding to the Earth or Solar system scale.

We first show a set of all-sky panoramic images exposing the Milky Way Galaxy at different wavelengths in the electromagnetic spectrum. Note that since these images are either acquired on Earth or by Earth-orbiting satellites, renderings are guaranteed to be correct if the viewpoints from which we are looking at these images are located at the scale of the Solar system, which is small in scale compared to the Milky Way or stellar...
Figure 10.5: All-sky multi-wavelength panoramic images.
scale (refer to the environment caching analysis in Chapter 6). Figure 10.5 shows the all-sky panoramic images used in this section; descriptions of these images can be found in Appendix A.3.

In practice, these panoramic images can be rendered as environment maps (in the form of sphere maps or cube maps) enclosing the whole 360° × 180° sky at the viewpoint or as Aitoff-transformed images preserving equal-area image pixels. Each representation has its own advantages: An environment map can make use of the whole dome in the planetarium environment so that the audience can have a better sense of immersion inside the virtual astronomical space, whereas an Aitoff representation squeezes the whole sky into a single image so that the audience can perceive the whole sky into a single view. In particular, both representations are used in our Solar journey animation. Figure 10.6 shows spherical environment maps corresponding to the Visible and Soft X-ray all-sky images from Figure 10.5.

![Visible](Visible.png) ![Soft X-ray](Soft_X-ray.png)

Figure 10.6: A spherical environment map encloses the whole sky around the viewpoint.

Noting that each panoramic image corresponds to a specific wavelength range, we can sort these images in order of increasing wavelength and then animate the multi-wavelength sky by adjusting a wavelength parameter along a spectral slider. Figure 10.7 shows the
spectral slider: Corresponding to a tab along the slider, the system chooses a related panoramic image(s) to be displayed: If the selected wavelength parameter matches exactly the wavelength associated with a certain panoramic image, that image will be selected for display. Otherwise, the two panoramic images having wavelengths closest to the specific wavelength parameter are chosen and blended for display. Their proximities to the wavelength parameter determine their corresponding blending factors so that we can fade between pairs of images smoothly. Furthermore, this visualization method can be applied in either a full dome or Aitoff mode. Strictly speaking, since most of these panoramic images are encoded with pseudo-color, the blended panoramic image in-between pairs is certainly not physically correct, but, it does give audiences a sense of transition and comparison.

10.3 The Milky Way at the stellar scale: No Hipparcos stars

In the previous section, we modeled and rendered the Milky Way Galaxy as a panoramic image, which could be rendered as an environment map enclosing the viewing position. If we navigate only within the Solar system, the parallaxes exhibited by nearby stars are still so small that they do not move against the background, i.e., the panoramic image. This is guaranteed according to the environment caching criteria we derived in Chapter 6.

However, if we journey further out to the stellar scale (the scale of the Hipparcos Catalogue), stars will gradually move out of the panoramic image and appear three-dimensionally
Figure 10.8: Doubled stars occur when we move to the stellar scale if we do not remove 2D Hipparcos stars in the Mellinger photograph.
in the navigation space. If we still use the same panoramic image to represent the Milky Way Galaxy, we will end up rendering some nearby stars in the Hipparcos Catalogue twice and double images of these stars will appear in the rendering. Figure 10.8 show two typical renderings of the sky for which double stars occur: one originally captured on the panoramic image and one rendered interactively as a three-dimensional star.

To avoid this unpleasant artifact, we have to remove all stars with 3D models from the panoramic image. In essence, these are the stars in the Hipparcos Catalogue, or Hipparcos stars in short. To fix this problem in the implementation, we make use of the star identification technique in Subsection 9.4(iii) and the texture synthesis algorithm [154, 50, 174] in the computer graphics literature. The following subsections detail our star removal method.

**Labeling star pixels**

The very first step to remove Hipparcos stars from the panoramic image is to locate these stars on the image. This is fairly an easy step given our previous work in Subsection 9.4. All we need to do is 1) determine the image location of Hipparcos stars on the panoramic image, 2) jitter the Gaussian filter to determine a better image location for the star center, and 3) label the pixels belonging to the stars.

The first row in Figure 10.9 zooms in to two different regions in the Mellinger image, and the second row shows labeled pixels in red. In this example, red pixels belong to Hipparcos stars with apparent magnitude less than 7. Since the panoramic image covers the whole sky, we do not need the image registration step in Subsection 9.4. Instead, we can directly determine image coordinates given the Galactic coordinate of stars based on the mathematics in Subsection 10.1 (under the Physics and Mathematics heading). In addition, to avoid the star halo remnants in the image, we enlarge the effective star size during the labeling.
Figure 10.9: Removing Hipparcos stars (apparent magnitude < 7) from the Mellinger Image [119].
Filling star pixels using Texture Synthesis

After labeling the star pixels, our next step is to synthesize pixel values in the red spots as if they were statistical background pixels. To do this, we make use of the texture synthesis algorithm in [50]: It is basically a pixel-by-pixel filling algorithm that puts in the most probable RGB values into the labeled areas. The assumption of the algorithm is that the two-dimensional image has to possess certain statistical distribution features. Some example textures having statistical patterns include knitwear, marble, wood, etc., as shown in Figure 10.10.

![Figure 10.10: Textures having statistical patterns: knitwear, marble, and wood.](image)

We assume that if some pixel values (say at pixel \( p \)) in an image like Figure 10.10 are missing, we can look up a sample image that possesses the same kind of statistical pattern, and synthesize the pixel value at \( p \) (the missing pixel) by comparing pixels around \( p \) against the sample image. Since the sample image has the same statistical pattern as the original input image, the statistical pattern is theoretically preserved. Details and pseudo-code of the texture synthesis algorithm in [50] can be found in http://www.cs.berkeley.edu/~efros/research/NPS/alg.html.

Note that when applying texture synthesis for the red pixels in Figure 10.9 (second row), we regard the input image that contains to-be-filled pixels as the sample image at the same time. However, we have to constrain the texture synthesis process so that it picks up only sub-regions in the input image (the sample image) that do not overlap with the to-be-filled
pixels. After the texture synthesis process, we obtain the stars-removed images in the last row of Figure 10.9. By comparing the first and third rows in the figure, we can see that the troublesome Hipparcos stars have been removed seamlessly and the labeled red pixels have been filled with other colors reproducing the statistical variation of the background, which makes the filled areas look more natural. Furthermore, we can fix the double star problem occurring in Figure 10.8 using the star-removed panoramic image: Corrected renderings are shown in Figure 10.11.

![Image showing comparison of star-removed panoramic images](image)

**viewpoint at stellar scale (same as second row in Figure 10.8)**

**same viewpoint position without double star images**

Figure 10.11: Doubled stars are eliminated when we move to the stellar scale after removing Hipparcos stars in the Mellinger photograph.
The Controversy

Nevertheless, there is still one controversial issue in the resultant panoramic image after Hipparcos star removal. The situation here is that since we are applying the texture synthesis method naively, the algorithm may synthesize anything that is statistically reasonable in the labeled areas. In this way, small stars may be generated in the labeled areas.

The controversy here is that whether these small stars are acceptable. Strictly speaking, in terms of scientific measurement, what we have detected or observed in the labeled areas are only the bright Hipparcos stars; no small stars have been detected in these areas and so generating these small stars could be an error. However, in terms of the statistical distribution of stars over the sky, what the algorithm generated is statistically correct; there should be a distribution of small stars in the background if they were not overshadowed by the bright stars. Ignoring these small stars denies the validity of the original statistical distribution.

Concerning this dilemma, my point of view is to let the developer/animator decide which way they prefer in their own applications. If they choose to avoid these generated stars, they can constrain the texture synthesis process by forbidding star colors from being generated during the synthesis step. In the next subsection, we extend our star removal process with a color analysis step, which helps to avoid the generation of new stars during the texture synthesis process.

Color Analysis

Essentially, this color analysis step is an amendment step, which avoids the generation of new stars during the texture synthesis process. The general approach of this method is to find out the range of background (non-star) pixel values surrounding each to-be-removed Hipparcos star, and constrain the texture synthesis process so that it generates pixel values...
probabilistically in the range of background pixel values into the labeled areas. Altogether, there are four steps in this color analysis method:

[1] Obtain Surrounding Pixels
First, pick up pixels surrounding each Hipparcos star to be removed (but not those labeled pixels belonging to the star). Typically, around 200 or at least 100 surrounding pixels have to be included, depending on the size of the Hipparcos star in the image.

[2] Pixel value Analysis
Regard RGB values of each picked pixel as positions in the RGB domain \([0, 1]^3\) and determine the major axis (principle projection) of these data points using the PCA or Eigenvector analysis.

[3] Pixel Distribution
Next, project all pixel values (R,G,B) onto this principle axis and formulate the distribution of pixels along this axis as a histogram. In practice, this histogram usually exhibits two maxima: one for the background pixels and one for the star pixels.

[4] Background Analysis
Finally, since background pixels always contribute to the lower intensity peak, we can fit a Gaussian distribution (or other distribution) around the peak and make use of this function to constrain the texture synthesis process probabilistically. In a word, the texture synthesis process thus has a higher chance to synthesize pixels with values around the Gaussian peak, and almost no chance to generate pixels possessing star colors.

Note that these four steps have to be performed for each to-be-removed star, because the nature of background pixels around each star could be different.
10.4 Representing the Milky Way in Three-dimensions

At the stellar scale, it is fairly correct to show the Milky Way Galaxy as an all-sky panoramic image with Hipparcos stars removed because the Milky Way Galaxy is still much further away from us than the Hipparcos stars. Note that the Hipparcos Catalogue ranges up to about $10^{18}$ meters from Earth whereas the Milky Way Galaxy has size $10^{21}$ meters. However, when we journey beyond the Hipparcos stars, representing the Milky Way Galaxy with the same panoramic image is no longer valid. Now, we need a fully three-dimensional model for the Milky Way Galaxy.

Figure 10.12: The Milky Way VRML Model from the Goddard Space Flight Center at NASA.

The Goddard Model

We have based our statistical Milky Way model on a simulation from the Goddard Space Flight Center at NASA; as described in the Appendices (Section A.3), this model is a three-dimensional point-based VRML (Virtual Reality Modeling Language) model, consisting of about $40K$ points. Figure 10.12 shows the model as it is in VRML (using the VRML browser in Netscape).
The Particle Model

To increase the realism and visual quality of the Goddard model, our approach is to turn it into a particle model based on the standard particle system technique [142, 143, 136, 155] in computer graphics.

The Particle System. Basically, particle system [142, 143, 136, 155] is a conventional computer graphics technique to model spray, rain, snow, fireworks, gas, etc. It has become so common and useful for graphics modeling that most commercial animation packages like 3D Studio Max and Maya support this rendering method.

Figure 10.13: Rendering examples of Particle systems from www.cs.unc.edu/~daveme/Particle.

The general idea of a particle system is that models created by particle systems are made up of a (usually large) number of small particles. Each particle has its own set of attributes such as position, size, and color. Depending on individual applications, we can have additional attributes such as mass, velocity, lifetime, associated geometrical models (for individual particles), etc. Sometimes, these attributes are animated (change with time) to make the model more realistic and lively.

Making the Milky Way Model. There are two basic issues for making a three-dimensional particle model for the Milky Way Galaxy: modeling and rendering.
To make our own particle model for the Milky Way Galaxy, we make use of the NASA Goddard model as the base model and then, we

1. Randomly pick a subset of points from the Goddard model as particles, and
2. Assign color and size to these particles roughly based on the distribution of matter in the Milky Way Galaxy, as well as galaxy photographs (see Figure 10.14) with shapes similar to our Milky Way.

- Rendering

Then, during the rendering, we can draw these particles in a way similar to the way we drew stars: textured billboards or anti-aliased points. In particular, if we render with the textured-billboard mode, we need to depth-sort all particles from back to front with respect to the viewpoint because the texture we use is semi-transparent. In the implementation, the Quicksort algorithm is employed for the sorting, which takes $O(n \log n)$, where $n$ is the number of particles.

**Performance Issues.** To achieve higher realism and visual quality for the Milky Way Galaxy, instead of using a particle model to imitate its volumetric appearance, we should
model the whole Milky Way Galaxy truly as a volumetric model and apply volume rendering. However, a high-quality volume-rendering of a volume as large as the Milky Way usually takes very long processing time (hours or days), so that real-time rendering of the Milky Way becomes impossible with current graphics technology. Since we are targeting real-time rendering, the particle model is chosen as the modeling tool with appropriate quality/performance tradeoffs.

Since we do not just render the Milky Way particle model, but also other data, we may unfortunately need to lower the particle count in the Milky Way model. In particular, we found that about 8,000 to 10,000 particles are enough to accurately suggest the general shape of the Galaxy.

To further augment the rendering performance, we can explore different level-of-detail strategies: various particle models with different particle counts could be used for viewpoints at different ranges from the Milky Way center, or a hierarchical rendering approach like the VP-tree or M-tree approach for star rendering, could also be attempted.

**Rendering Results.** Figure 10.15 shows the simulated Milky Way Galaxy implemented using our particle system. The background of the renderings is filled with galaxies, indicating that we render much more than just the Milky Way Galaxy. In addition, the tiny satellite galaxy next to our Milky Way, namely the Large Magellanic Cloud (LMC), is present in the renderings. More information about modeling and rendering of galaxies can be found in the next chapter. We can compare these renderings with the actual photographs of similar galaxies in Figure 10.14.
Figure 10.15: Renderings of the 3D Milky Way Galaxy using our particle system.
10.5 Conclusion and Future work

In conclusion, we propose in this chapter various modeling and rendering approaches for the Milky Way Galaxy at three different spatial scales: at Solar system scale, where the Galaxy is represented as an all-sky panoramic image in the form of an environment map; at the stellar (Hipparcos) scale, where it is still represented as an all-sky image but having 2D Hipparcos stars removed for correctness; and beyond the stellar scale, where it is represented as a fully three-dimensional particle model.

In the future, we would like to investigate the following possibilities to extend our existing work on the Milky Way virtual model.

- The first problem is that we need a neat way to do the transition between the 2D panoramic image and the 3D particle model. Now, we are only blending between the two representations while the viewpoint travels across the two scales.

- Second, we need a scientific way to produce size and color for particles in the particle model so that the 3D Galaxy appears more like the observed data, e.g., the Mellinger image, and other barred-spiral galaxies.

- Thirdly, to improve the realism of the particle model, we can incorporate the rotation curve, $V(R)$, as the local standard of rest (LSR) along $R$ for each particle, and simulate the slow differential rotation of the Milky Way Galaxy.

- Finally, when describing the Milky Way in 3D, astronomers divide the Milky Way plane into several regions such as the Orion arm, the Sagittarius arm, the Pegasus arm, etc. We can make the three-dimensional model more informative by plotting a map of these sectors along with the three-dimensional model.
Modeling and Rendering Galaxies

11.1 About Galaxies

The night sky contains more than just stars, nebulae, and the Milky Way Galaxy; indeed, if we look deeper into the sky, we find fuzzy structures called galaxies. However, since most galaxies are so far away from us compared to stars, most of them were unknown to mankind until the 17th century. Only the Large and Small Magellanic Clouds (LMC and SMC) had actually been observed and recorded up to that time. With the help of the telescope, Charles Messier in 1780 produced the first catalog of fuzzy objects in the night sky; this is known as the Messier catalog. Information and pictures of objects in this catalog can be found in http://www.seds.org/messier/indexes.html. Note that the common astronomy notation that starts with an M (standing for “Messier Object”) was introduced in this catalog. For example, M31 and M33 refer to the Andromeda galaxy and the Pinwheel galaxy, respectively. Table A.14 lists all objects in the Messier catalog.

Due to the fact that telescopes in Messier’s time were still of fairly low resolution, astronomers at that time were not able to measure heavenly distances. As a result, galaxies were not differentiable from nebulae and globular clusters, and all these objects were documented together in Messier’s catalog. Other related catalogs include the New General
Catalog (NGC) and the Index Catalog (IC), which were initially compiled by Dreyer et al [46, 47]; identifiers from these two catalogs start with NGC and IC, respectively.

In ancient times, the conceptual extent of the Universe was limited to our Solar system, with stars assumed to be fixed on a big celestial sphere. Then, after the discovery of star distances and the nature of the Milky Way Galaxy, our conceptual Universe expanded to include the whole Milky Way Galaxy, which was then thought to encompass the whole Universe. Starting around 1920, with the development of Cepheid variables, the bright star measurement method, and the Hubble’s distance-redshift relationship, astronomers could finally estimate distances to galaxies and verify the existence of other Milky-Way-like galaxies. In this way, our concept of the Universe expanded again: Like our Sun in the Milky Way, our Milky Way Galaxy is merely one example among billions of galaxies in the Universe. In reality, there exist huge galaxies, whose radii approach 250,000 light years, about 10 times bigger than our Milky Way Galaxy, as well as dwarf galaxies, whose sizes can be as small as 10,000 light years.

This chapter is devoted to galaxies and includes four sections altogether: The first section presents some background knowledge about galaxies, including dataset sources, coordinate systems, and some basic astronomy knowledge. The second section presents various representations used to model and render galaxies, whereas the third section presents the rendering results. Finally, we draw our conclusions and discuss possible followup work in the last section.

Datasets

We have explored the following data sources for the modeling of galaxies and our known Universe beyond the Milky Way:

- Brent Tully’s Galaxy Catalog [169, 168]
This dataset contains about 35K galaxies in our known Universe; each has a unique galaxy ID/name, coordinates, galaxy diameter, absolute magnitude, distance in velocity units, etc. This catalog was assembled by Brent Tully at University of Hawaii (tully@ifa.hawaii.edu).

- Photographs of galaxies
  In addition, photographs of galaxies belonging to different morphological types were gathered from the web and used for galaxy modeling.

- Cosmic Background Radiation (CMBR) [13, 15] and Wilkinson Microwave Anisotropy Probe (WMAP) [14]
  These are all-sky panoramic images of the cosmic background radiation (CMBR), measured by the COBE team and the Wilkinson Microwave Anisotropy Probe (WMAP). Detailed information about the COBE project can be found in http://lambda.gsfc.nasa.gov/product/cobe, while WMAP data reside at http://map.gsfc.nasa.gov.

  Details about these datasets can be found in Appendix A.3. Other to-be-explored datasets at the Universe scale include quasars, radio galaxies, etc.

**Coordinate Systems**

At the scale of the Universe beyond the Milky Way Galaxy, astronomers use another coordinate system, the Supergalactic coordinate system. Details and transformations relating to this coordinate system can be found in Appendix A.4. Further clarification on the exact meaning of this coordinate system can be found in the Third Reference Catalogue of Bright Galaxies [41] by de Vaucouleurs, et al, 1991.
Basic Knowledge about Galaxies

In this subsection, we discuss some basic astronomy knowledge concerning the understanding and visualization of galaxy catalogs.

Figure 11.1: The Hubble relation: $V = Hd$ (by the Hubble Space Telescope Key Project).

(A) Hubble relation: Red shift and Distance. The Hubble relation states that the velocity of recession, as determined from the spectral redshift, is directly proportional to the distance to the galaxies; this statement can be expressed as

$$V = H \times d,$$

where $V$ is the velocity of recession (along the radial direction), $d$ is the distance to galaxy, and $H$ is the constant of proportionality, known as the Hubble constant.
Efforts made by the Hubble Space Telescope Key Project showed in 2000 that the Hubble constant is $70 - 75$ km/s/Mpc with an uncertainty of 10%. Figure 11.1 shows the result of this project. However, some other methods based on Cepheid data from HST suggest $H$ could be as low as 58 km/s/Mpc. As a compromise, we choose an intermediate well-accepted range for $H$, namely 50 - 77 km/s/Mpc.

(B) Positioning Galaxies. There are two steps to positioning galaxies: 1) position the center of the galaxy in three-dimensional space, and 2) orient the galaxy plane.

Step 1 -- Galaxy Location
As mentioned previously in the coordinate system subsection, galaxy positions are expressed in the Supergalactic coordinate system; we denote the coordinate as $(sgx, sgy, sgz)$, which can be obtained directly from the galaxy catalog.

![Figure 11.2: Positioning the galaxy plane: major and minor axes.](image)

Step 2 -- Major and Minor Axes
In addition, it is worth noting that galaxies are not points or spheres like stars; they have...
different shapes as observed from Earth. Concerning this, astronomers define the major and minor axes of galaxies to represent the principal axis and the co-principal axis in the viewing plane, respectively (see Figure 11.2(a)).

Then, based on the vector to the galaxy, $\vec{p}_{\text{galaxy}} = (sgx, sgy, sgz)$, we can set up the initial position of these axes in the sky (see Figure 11.2(b)). Here, the major axis is parallel to the $xy$-plane and perpendicular to the SGZ-axis in the Supergalactic system, whereas the minor axis is perpendicular to both the major axis and to $\vec{p}_{\text{galaxy}}$ (vector to the galaxy). Next, astronomers define another term, $\theta_{pa}$, the hold position angle, which rotates the two axes counter-clockwise about $\vec{p}_{\text{galaxy}}$ (line of sight to the galaxy center). Thus, we can align the galaxy axes (and galaxy photographs) from the Earth's perspective.

**Step 3 — Tilting the Galaxy Plane**

So far, we have learned how to fix galaxy photographs on the sky according to the documented data, but all of them are having their planes facing the Earth after steps 1 and 2. This is completely unlikely to be the real situation. Astronomers attempt to predict the orientation plane by making the following assumption: First, we denote $a$ and $b$ as the length of the major and minor axes as projected on the galaxy photograph, respectively.

![Tilting the galaxy plane](image)

Figure 11.3 illustrates the conventional estimation: galaxies are assumed to be circular...
in size, thus $b$ is lengthened along the line of sight. In a word, the plane of a circular galaxy is tilted about the major axis in such a way that the length of the minor axis equals the length of the major axis. Given $\theta$ as the tilting angle, we have

$$\theta = \cos^{-1} \left( \frac{b}{a} \right).$$

Nevertheless, since galaxies are so far away from us, it is not yet possible to determine the tilting direction experimentally. In other words, the galaxy plane may be tilted by $+\theta$ or $-\theta$. In practice, we have to pick the tilting direction at random.

**Step 4 — Transformations**

Finally, we may need to transform galaxies from the Supergalactic to the Galactic or Equatorial coordinate system if we are working in a different coordinate system.

**Note:**

Fortunately, the following quantities are available for each galaxy record in the catalog we obtained from Brent Tully:

- $(sgx, sgy, sgz)$ — Cartesian position in Supergalactic coordinates (in 10kpc units)
- $hpa$ — hold position angle (in degrees)
- $b/overa$ — axial ratio
- $adiam$ — metric diameter (in kpc units)

(C) **Galaxy type.** Unlike stars, which can be modeled as points or perfect spheres geometrically, galaxies vary in shape, from elliptical, lenticular, and spiral, to irregulars.
1. **Hubble's classification (from [102])**

Edwin Hubble first classified galaxies and proposed the "tuning fork" diagram for galaxy type (see Figure 11.4) in 1936.

In Hubble's classification scheme, there are four types of galaxies:

i) **Elliptical galaxies** ($E_n$)

- elliptical concentration of stars without signs of interstellar medium
- classified as $E_0, E_1, \ldots, E_7$ such that $E_n$ has

$$n = 10 \left( 1 - \frac{b}{a} \right),$$

where $a$ and $b$ are length of the major and minor axes, respectively. For example, an $E_0$ galaxy is a perfect circular disk in the sky.

- cD - giant elliptical galaxies.

ii) **Lenticular galaxies** ($S_0$)

- between the elliptical and the spiral type
- has very little interstellar medium and shows no sign of spiral structure

iii) Spiral galaxies ($S_a$-$S_b$-$S_c$ or $SB_a$-$SB_b$-$SB_c$)

- has a well-defined spiral pattern in the disc, a central bulge like the $E$ type, and a stellar disc like the $S_a$ type

- Capital B indicates the presence of a central bar

iv) Irregular galaxies ($Irr$ I or $Irr$ II)

- $Irr$ I: has spiral-arm-like structure, but patchy and disjoint (beyond $S_c$)

- $Irr$ II: more dusty and distorted

Other types like dwarf galaxies ($dE$) were introduced to the scheme later.

2. De Vaucouleurs classification (from [173])

Extending Hubble's work, De Vaucouleurs added two more identifiers to describe the shape of galaxies. Figure 11.5 illustrates his scheme.

- Bar strength index ($A$–$B$)

\[
\begin{align*}
A & \quad \text{negligible} \\
B & \quad \text{prominent}
\end{align*}
\]

- Inner structure index ($r$–$s$)

\[
\begin{align*}
r & \quad \text{ring-like} \\
s & \quad \text{spiral-like}
\end{align*}
\]

Other types introduced by De Vaucouleurs include the $S_d$ class (beyond $S_c$), the $S_m$ or $I_m$ class (irregular class like Large and Small Magellanic Clouds), etc. The Milky Way is now believed to be an SABb(rs) barred spiral. Knowing these codes helps us to understand the galaxy catalog better.
3. Morphological type

In addition to the above alphanumeric types for galaxy shape, astronomers use also a coarser classification, the morphological type, denoted by a number ranging from $-5$ to $10$. The negative values correspond to elliptical galaxies, whereas the positive values correspond to spiral galaxies. This can be understood as a flattening of Hubble's tuning fork in Figure 11.4.

Figure 11.5: De Vaucouleurs classification scheme (the central $SABb(rs)$ type is thought to be very close to the shape of our Milky Way) (from [173]).
Zone of Avoidance. Due to the fact that we are sitting inside the Milky Way Galaxy when we detect distant galaxies, galactic light from outer space has to penetrate the Milky Way to reach us. However, galactic light can be blocked by the interstellar medium in the Milky Way. Since the Milky Way is a disc in structure, galactic light coming in parallel to the Milky Way disc is more likely to be blocked. As a result, if we plot galaxies against the sky, we will find that fewer galaxies are detected in the plane of the Milky Way. Figure 11.6 shows a plot of 34,729 galaxies cataloged by N.A. Sharp in 1986; we find very few galaxies in the horizontal area, which corresponds to the galactic plane. This 5° to 10° band in the sky is called the zone of avoidance. It can be easily seen in a 3D view of the Tully catalog.
11.2 Modeling and Rendering at the Scale of Galaxies

Modeling and Rendering Galaxies

Representation  Unlike stars, which are represented only as points or textured billboards, we represent galaxies in a variety of different ways.

1. Anti-aliased points

We can represent galaxies as points of variable sizes according to their apparent magnitude as seen from the viewing position. The resizing method used here is exactly the same as that for stars (see Subsection 9.4).

2. Axial frames

We can plot the major and minor axes of a galaxy to visualize its orientation. This is achieved by the positioning method described in the previous section.

3. Textured polygons

The available galaxy catalogs document also the morphological types of galaxies. Thus, we can collect sample photographs of galaxies belonging to each type and texture-map these photographs onto the axial planes of those galaxies of each type for which we have no image in our collection.

4. Particle model

Currently, a particle model is supported only for the Milky Way Galaxy, but it is extensible to other galaxies for which detailed geometric information is available.

Example renderings showing these representations can be found in the next section.

Rendering Issues
1. **Need for Depth Rescaling.**

Due to hardware limitations for large-scale visualization, we need to apply the depth rescaling method for rendering galaxies. This is exactly the same method used for handling stars.

2. **Size exaggeration for visualization.**

The other issue for galaxy rendering is that the size of galaxies is usually small compared to the intergalactic distances between them. Therefore, when we navigate among them at the intergalactic scale ($10^{22}$ to $10^{24}$ meters), galaxies rendered at their true sizes (as textured polygons) according to the $adiam$ parameter will generally be too small to be viewable on the screen. To compensate for this, we may apply a scaling parameter to magnify their sizes during the visualization. Note that astronomers use a very narrow field-of-view to look at galaxies, whereas we have to use a natural field-of-view while navigating in three-dimensional space.

**Cosmic Background Radiation (CMBR and WMAP)**

Figure 11.7(top) shows the all-sky image corresponding to the three-degree Kelvin Cosmic Microwave Background Radiation (CMBR) measured by the COBE team (The word COBE stands for “Cosmic Background Explorer”), while Figure 11.7(bottom) shows the more recent results of the Wilkinson Microwave Anisotropy Probe (WMAP).

To put a CMBR map into the virtual environment, we apply it as a texture on a large sphere. In cosmological terms, since distance is actually related to time, we can trace time back to the moment when the CMBR was released during the “Great Flash”. We thus can position the CMBR sphere at that scale, which is about $1.296 \times 10^{26}$ meters from Earth in comoving coordinates.
Figure 11.7: The original COBE Cosmic Microwave Background Radiation Anisotropy (CMBR) [15, 13] (top) and the more recent Wilkinson Microwave Anisotropy Probe (WMAP) [14] (bottom).
11.3 Rendering Results

To experiment with the modeling and rendering of galaxies, we developed an interactive program called "galview," which stands for "galaxy viewer." This program was written in OpenGL using GLUI and GLUT for interactivity. Taking advantage of OpenGL, galview can run on different platforms including Win32, SGI, and Solaris. Figure 11.8 shows the interface of "galview." The left hand side is the control panel for various parameters like display control, viewing control, galaxy display mode, etc. The central area is the rendering screen showing the galactic space; the default initial view focuses on our Milky Way galaxy.

![Figure 11.8: The interface of "galview."](image)

1. Representing Galaxies

Figure 11.9 shows the rendering of galaxies using different representations. The left column shows galaxies as variable-sized points according to their apparent magnitude from...
the viewing position. The central column shows the axes (major and minor axes) of galaxies for the visualization of galaxy orientation. The last column shows galaxies as textured polygons using real galaxy photographs. We first focus our attention on the Milky Way Galaxy in the first row of the figure. In the next row, we move away and see the cluster formed by the Milky Way and the two Magellanic Clouds. In the bottom row, we move still further outward to expose the galaxy clusters in the Local Group. Note that, except for the Milky Way, we have magnified galaxy sizes (the geometry) by a factor of 10 to make them visible on the screen.

2. Powers-of-ten transition

In this example, we look at a powers-of-ten transition from the Milky Way scale to a very large scale enclosing all documented galaxies in the Tully catalog. Figure 11.10 demonstrates the walk-through: The image sequence in the figure is left-to-right and then downwards. To facilitate the perception of spatial magnitudes, users can enable the power-cubes (yellow wire-frame boxes), to show relative sizes of corresponding spatial volumes. This walk-through starts at the Milky Way scale (at about $10^{22}$ meters), and slowly moves out to the Virgo cluster, whose distance scale is about $10^{23}$ to $10^{24}$ meters. Finally, we zoom out to see all the documented galaxies of the catalog in the final picture. At this scale, white spots reveal dense concentrations of galaxies, i.e., galaxy clusters. In addition, notice that there is a purple/blue background image in the renderings; this is the COBE Cosmic Microwave Background Radiation (CMBR) fluctuation shown as a background panorama marking the scale of the earliest visible radiation.

3. Zone of Avoidance

We can look also at the zone of avoidance. Figure 11.11 visualizes this region at different scales. In the first row of the figure, we are at the scale of our Milky Way. If we put our viewpoint in the plane of the Milky Way disc, we can see that almost no galaxies are found
Figure 11.9: Representing galaxies as points (left column), as axial frames (central column), and as textured polygons (right column), with $10X$ exaggeration for visibility.
Figure 11.10: Powers-of-ten transitions starting from the Milky Way Galaxy, left-right, top-bottom order with increasing scale.
along the horizon matching the Milky Way plane. The situation is more obvious if we render galaxies as axes.

![Figure 11.11: Visualizing the zone of avoidance. Galaxy size exaggerated by 10X (top) and 100X (bottom) for visibility.](image)

If we move out to see the structure of the entire set of galaxies, we again can find the zone of avoidance if we match our viewpoint to the plane of the Milky Way. From the second row of the figure, we see a missing gap in the distribution of galaxies that corresponds to the zone of avoidance.
11.4 Conclusion and Future work

This chapter addressed some basic issues concerning the modeling of galaxies using computer graphics: 1) positioning galaxies and 2) galaxy type and shape. We presented our approach for modeling and rendering galaxies with different representations: anti-aliased points, textured polygons (not billboards), and axial frames. Various renderings were made to demonstrate different views of galaxies, the zone of avoidance, and the overall appearance of the whole galaxy catalog.

With the development of galaxy surveys and catalogs, we now know that galaxies tend to cluster together to form groups. One example is the Local Group of our Milky Way Galaxy, which is about 1.5 Mpc in size, and includes Andromeda, SMC, LMC, etc. Groups like this congregate together to form clusters and then even bigger clusters, called superclusters. Our local group belongs to the Virgo supercluster, which is roughly 20 Mpc in size. Beyond this scale, there are even larger structures like filaments, sheets, and bubbles, which are believed to be the left-over structures from the formation of the Universe. Our current approach renders individual galaxies, which does not allow us to visualize larger structures easily. The next exciting visualization task would be to create seamless transitions from galaxy-level rendering to global-structure rendering so that we can understand the hierarchical structure of galaxies more intuitively.
Part III

Visualizing Cosmological Time
Time is a critical aspect of visualization systems that implement dynamic simulations and animation. Dealing with time at the scales required to conceptualize astrophysical and cosmological data, however, introduces specialized problems that require unique approaches. In this part, we have extended our previous investigations on interactive visualization across extremely large scale ranges of space to incorporate dynamical processes with very large scale ranges of time. Chapter 12 shows how to visualize the whole body of cosmological data through a volumetric cone, namely the Lightcone, which can be understood as a clock representing the volume of the Universe that we can see with a camera pointed at the night sky. Using comoving coordinates, which treat the Universe as though its present size was always its size, the volume of visible space increases continuously until we reach the origin of the Cosmic Background Radiation, about 13 (13.7, says WMAP) billion years ago, currently estimated to have occurred when the Universe was about 400,000 (380,000, says WMAP) years old.

In Chapter 13, we focus on several issues: time scales that are too short or too long to animate in real time, those needing complex adjustment relative to the scale of space, time simulations that involve the constant finite velocity of light (special relativity) in an essential way, and those that depend upon the dynamics of the coordinate system of the Universe itself (general relativity and cosmology). We conclude that a basic strategy for time scaling should ordinarily track the scaling of space chosen for a particular problem; e.g., if we are adjusting the interactive space scale, we scale time in a similar way. At cosmological scales, this has the interesting consequence that the time scale adjusts to the size of each era of the Universe. If we make a single tick of the viewer’s clock correspond to an enormous time when viewing an enormous space, we see motions in viewer’s time of increasingly larger, and usually appropriate, scales.
The Cosmic Clock – Lightcone

12.1 Time in Cosmology

Background

The nature of the physical Universe has been increasingly better understood in recent years, and cosmological concepts have undergone a rapid evolution (see, e.g., [106, 17, 27]). It is now generally accepted that the large-scale relationships and homogeneities that we see can only be explained by having the Universe expand suddenly in an “inflationary” jump to the true vacuum state in its earliest moments. We also have resolved Olber’s paradox — the fact that the night sky appears black when, in a uniform Universe, it would be brilliantly lit — by invoking the Hubble expansion, so that all the galaxies are flying away from each other, leaving enough of a gap to make the night sky dark. Furthermore, we can even attribute different rates of this expansion to the domination of different cosmological processes, with the very recent observation that we may be currently entering a new period of exponential growth dominated by the Cosmological Constant — the vacuum state of the Universe — rather than radiation or matter [26].
The physical processes of the very young Universe are believed to be accurately described by the "standard model" of elementary particle physics, with plasmas of quarks, gluons, leptons, and photons gradually cooling down to form hydrogen, helium, and lithium nuclei in precise ratios. The only step in this process we can measure directly is the final stage, where the temperature finally decreases to the point where enough protons and electrons combine to form stable neutral atoms, thus allowing light to escape and travel freely into the future. In this era, about 380,000 years after the Big Bang, the mean path length of radiation increased significantly, and, in a relatively short time, the Universe became transparent to radiation. The result was the "flash" of electromagnetic radiation at 3000° K that we see today, after a thousand-fold Hubble expansion, as the 2.7° K cosmic background radiation.

In this work, we focus on the question of visualizing the Universe while relying as much as possible on observational data. We could in principle treat the eras prior to the onset of transparency to radiation, as well as microscopic scales down to the Planck length of $10^{-35}$ meters and the Planck time (the travel time for a light ray across the Planck length) based on theoretical models of currently unobservable phenomena; however, the great variety of problems involved in building intuition about the spacetime phenomena we can actually observe experimentally provides sufficient challenges: We shall see this shortly as we use our spacetime model, the Lightcone, to depict the entire Universe as a single iconic model visualizable at a single glance.

The Universe is Expanding

When we design visualizations at the scale of the visible Universe, space and time intermingling in a manner described by Einstein's theory of general relativity. In this research work, we focus solely on the visible Universe and employ, as much as possible, actual experimental observations. Astrophysicists and cosmologists commonly employ a
radially symmetric form for Einstein's equations, the Robertson-Walker metric; following Carroll [27], we can write this equation in the form:

\[ ds^2 = a(t)^2 \left[ \frac{dr^2}{1 - kr^2} + r^2(d\theta^2 + \sin^2 \theta d\phi^2) \right] - dt^2 \]

to describe the qualitative features of the evolution of the Universe. When we choose appropriate energy-momentum tensor assignments for the radiation, matter, and vacuum-dominated eras such as those proposed by Carroll [27], we find that the Universe has been expanding, as Hubble observed.

Indeed, since Hubble's original discovery that the Universe is expanding, both the observational data and the corresponding theories have been significantly refined. According to our current best understanding, the time evolution of the size of the Universe is believed to follow the cosmological expansion curve in Figure 12.1, which plots the effective radius \( a(t) \) of the Universe as a function of time \( t \) (in units of the age of the Universe).

Another variable, the redshift \( z \), where

\[ a(t) = \frac{1}{1 + z} \]

is traditionally used to describe observed distant objects, their light, and their estimated ages. We see that \( z = 0 \) corresponds to the present era, the time of the most recent observable light. At the other end of the Universe, around \( z = 1100 \), when the Universe was three orders of magnitude smaller in terms of linear distance than it is today, the first visible light, sometimes called figuratively "The Great Flash," was set free to reach our present-day instruments. As the redshift decreased from \( z = 1100 \) to \( z = 0 \), the temperature characterizing this radiation dropped from a brilliantly visible cauldron of light at over 3000\( K \) to the present-day 2.7\( K \). The latter is the cosmic background radiation, the earliest measurable remnant of the birth of the Universe.
Figure 12.1: The expansion of the Universe given in terms of its effective radius $a(t)$ as a function of time $t$, in units of the age of the Universe.
12.2 The Making of the Lightcone

To support a more flexible and informative visualization of the whole body of cosmological data, we introduce an iconic representation of the Universe, the Lightcone.

The Basic form

Conceptually, when we look at the night sky, our view cuts through a visible volume of the entire spatial model space that we can represent as a solid view frustum or cone. Inside this solid cone, there are all kinds of datasets, ranging from the Moon or Mars, which are inside our Solar system, to Hipparcos stars, which are neighboring stars of our Sun, up to galaxy clusters far beyond our Milky Way.

Essentially, each ray of light that enters our eyes through this solid cone has a history. Due to the limiting speed of light, a light ray coming from the Moon or Mars takes a shorter traveling time through the solid cone, compared to light coming from faraway objects like galaxies that could have taken billions years of traveling time before reaching us. Using the fact that we have rough 3D position data of these objects, we can color code the data based on their distances from the Earth.

Here is the color coding scheme:

- The color red is assigned to the most distant objects.
- Intermediate distances interpolate from red to yellow to green to blue, and finally white.
- The color white is assigned to the closest objects.

Note that we used the same coloring scheme for star rendering: Figure 9.12 shows the same...
color map we used in Chapter 9. In practice, since astronomical objects span a vast spatial range, we use their logarithmic distances \( \log_{10}(\text{distance}) \) to look up their color values.

![Figure 12.2: Constructing the Lightcone (Earth at the tip of the cone).](image)

In order to provide an effective summary viewpoint of the datasets in the solid cone, we have adopted a logarithmic scale. The canonical viewing origin on the Earth is at the bottom tip of the solid cone, and the final upper surface is the event horizon for the first visible radiation, the Cosmic Microwave Background (CMB). Here, the whole solid cone corresponding to our view of the night sky is squeezed logarithmically so that objects located \( 10^d \) meters from us are placed at height \( d \) in the cone (see Figure 12.2). Since the CMBR dataset is located at around \( 10^{26} \) meters, the squeezed cone is about 26 units tall. Using this approach, we can visualize the entire viewing frustum across gigantic spatial scales at a single glance. We call this model the Lightcone, or the Cosmic Clock, because it records where historical events happened in the Cosmos from the present day back to the times of the Great Flash.
The Warp form

We can make one more transformation to see the entire volume of data, the whole Universe, at a single glance: We can take each spherical surface at constant radius from the observer on Earth and warp its $4\pi$ solid angle (a spherical shell) into a much smaller disk at the same radius in the solid cone. Mathematically, as shown in Figure 12.3, we define $\hat{v}$ and $\theta$ as the viewing direction and the half angle of the Lightcone, respectively. Then, given a data point located at true distance $10^8$, in the direction $\hat{d}$, or scaled distance $s$, from the observer, we can rotate the data point, $\tilde{d} = 10^s \hat{d}$, towards $\hat{v}$ using these formulas:

\[
\begin{align*}
\text{axis of rotation} & = \hat{d} \times \hat{v}, \\
\text{angle of rotation} & = \left( 1 - \frac{\theta}{\pi} \right) \cos^{-1} \left( \hat{d} \cdot \hat{v} \right).
\end{align*}
\]

Figure 12.3: Warping data points into the Lightcone.
In a very real technical sense, this is the visible light cone of four-dimensional spacetime reduced to a stack of disks, each corresponding to one particular age band of light-rays. The correspondence between the time when the currently-observed light was emitted from each disk and the emitter's distance from us in comoving coordinates motivates us describing this visualization as a clock.

According to the astronomical literature, if we ignore the effect of the Hubble expansion during navigation and assume the objects are located as though the Universe has always had the size it has today, we are visualizing the cosmos using a coordinate system referred to as Comoving Coordinates. Conversely, the coordinates using the actual size of the Universe at a particular moment are the rescaled Physical Coordinates.

Figure 12.4 gives an animation sequence corresponding to the view of the Universe from the tip of the Lightcone. This is accomplished by doing the angular warp described above. Note that the pale brown band correspond to the Milky Way Galaxy, which does indeed appear as a loop against the sky when scaled this way, whereas the red data points correspond to galaxies. Note that in this chapter, we render datasets as anti-aliased points for simplicity in the visualization.

The $a(t)$-rescaled form

Finally, we can take the Hubble expansion into account by incorporating the effective radius $a(t)$ into the Lightcone model. Since light travels at a finite speed, different datasets with different scales relative to the Earth actually correspond to snapshots taken at significantly different times. Therefore, we can use the distances to the Earth to deduce the corresponding times $t$ in the units of the age of the Universe. Using this $t$, we are able to look up the effective radius of the Universe and factor in a scaling by the $a(t)$ term. In effect, we modify the size of the slices in the Lightcone by this $a(t)$.
Figure 12.4: Animation sequence: warping the Universe's entire data volume into the Lightcone.
12. The Cosmic Clock – Lightcone

Figure 12.5: Animation sequence (from left to right): rescaling the Lightcone according to Physical Coordinates using the effective radius, $a(t)$ from Figure 12.1.

Figure 12.5 demonstrates the rescaling process to Physical Coordinates as an animation sequence: The image of the Cosmic Background Radiation is put at the far end of the cone and is later squeezed into a single point, illustrating the fact that the size of the Universe is in fact very small during the Great Flash.

12.3 Visualization Results with Lightcone

Slicing the Lightcone model
In the Lightcone model, each layer of objects corresponds to a different time or a different distance from Earth. In this sense, we can regard the Lightcone as being a spatial slider so that we can select spatial portions of the night sky in our viewing frustum, similar to the near and far clipping planes. Figure 12.6 demonstrates the slicing mechanism with a pair of dashed lines (yellow) to depict the slicing region: We can see from the figure that our
view of the night sky starts from the Solar system scale, then slides up to the stellar scale, the Milky Way scale, and finally up to the scale of intergalactic space.

Figure 12.6: The Lightcone as a spatial slider – we can slice through the cone and look at light in the night sky originating in different time ranges.

**External View in Comoving Coordinates**

In Figure 12.7, we show symbolic, but physically impossible, views of the Universe, as though we could watch a spherical shell of light traveling back in time from our viewpoint on the Earth to the earliest visible objects. As we reach each landmark – the earth, the Solar system, the Milky Way, and the most distant galaxies, we mark the scale in meters as each group of objects enters the visible spherical shell. The Cosmic Clock cone on the side has been modified: As in Figure 12.4, it now includes the entire Universe warped from a set of spherical shells to disk-shaped shells layered to make up the cone. We are still in comoving
coordinates, and all distances are scaled to the current size of the Universe.

Figure 12.7: Navigation with comoving coordinates.

**External View in Physical Coordinates**

In Figure 12.8, we now show the expanding sphere of light using the Robertson-Walker physical-coordinate rescaling. To the left is a warping of the cone in comoving coordinates to the true physical coordinates, where the upper tip is at $z \approx 1100$, the origin of the cosmic background radiation, and each $z$-region is about $1/(1+z)$ times smaller than the Universe in our present era. The bright light at the upper tip of the warped cone represents the $3000K$ temperature at this early time. The right hand images match those in Figure 12.7, but are rescaled to physical coordinates by $a(t) = 1/(1 + z)$ in Figure 12.1.
The Solar system  The stellar scale

The Milky Way  Galaxies

The Cosmic Background  The Great Flash

Figure 12.8: Navigation with physical coordinates, using the comoving coordinate scales in Figure 12.7 for absolute reference.
12.4 Conclusion

In this chapter, we presented the Lightcone visualization method, which allows us to look at the entire data volume (the visible Universe) at a single glance. Essentially, it is a clock recording the history of the Cosmos from the Great Flash period up to our present day. By rescaling the Lightcone with the effective Universe radius, \( a(t) = \frac{1}{1+z} \), we can visualize the effect of Hubble expansion and trace the spacetime volume back to the time when the first radiation was released; that is the compressed tip at the top of the Lightcone. Furthermore, we can regard the Lightcone as a spatial slider, and look at different spatial (or equivalently, temporal) portions of the night sky. These visualization strategies were assembled to make our animation, “Cosmic Clock” [80].
13

Time Management in the Universe

13.1 Introduction

Exploring large scale datasets that have a time component requires special attention to temporal scaling and representation issues. In the first part of this dissertation, where we treated large spatial scales, we described a power-homogeneous-coordinate-based approach to static datasets that spanned large orders of spatial magnitude; implicit in the treatment was the problem of handling large spatial scale; here, we will focus on the handling of large scales in the temporal domain.

In this chapter, we attempt to work out various design issues encountered in building a dynamic astrophysics visualization system, and present a family of techniques for dealing with animation and time visualization at huge scales, from daily life time-scale up to millions of years, corresponding to the rotational speed of the Milky Way. Table A.18 in the Appendices illustrates typical events and motions using units of powers-of-ten in time.
Challenges of Large-scale Dynamic Astrophysics Visualization

To navigate through large-scale dynamic astrophysical data sets, in addition to the difficulties of large spatial scale, there are difficulties such as the following:

- **Temporal magnitude.** Besides huge orders of magnitude in space, we encounter huge orders of magnitude in time. It takes one day for the Earth to rotate once on its axis, whereas it takes one year to orbit around the Sun, and Pluto takes almost 248 years. In contrast, our Galaxy takes 240 million years to complete one rotation. Thus, handling motion and animation timing properly in astronomical visualization requires special attention.

- **Reference Frames.** Second, there is no absolute reference center in the Universe. We cannot have the same reference frame fixed to the Earth as it moves around the Sun and to the Sun as it moves through the Milky Way. In this dynamic astronomical environment, nothing is at rest. Since we can place the observer anywhere, neither the Ptolemaic Model nor the Copernican model is adequate to simulate the dynamic interaction among datasets.

In the next three sections, we discuss techniques for dealing with the above problems inherent in large time scales. In the following chapter of this dissertation, we will extend the management of large space-time scales to navigation methods for large-scale environments.

### 13.2 Managing Time in the Virtual Environment

We deal in this chapter mainly with real-time interactive environments, either on the desktop or in a virtual reality simulation. The human viewer has his or her own specific
perceived time scale, typically quantized in units of screen refresh time; we must on occasion also account for the human input time scale and human perceptual limitations. The relation between the simulated time and the viewer’s “screen time” is then the critical issue for animations. The Universe is billions of years old; major steps in biological evolution happen on scales of millions of years; Pluto takes 249 years to orbit the Sun; light takes 9 minutes to reach us from the Sun; molecules undergo chemical reactions in microseconds; currents flow through computer chips in less than a nanosecond (see Table A.18 for typical cases). However, human interactive scales go no smaller than units of about 1/30 of a second, with durations from a few seconds to a few hours at most. We must match these time constraints when designing and managing a virtual environment.

Representation

To supply intuition that assists the visualization of dynamic processes over a large range of time scales, we must rescale the intervals between time-dependent changes in the simulation so they are appropriate to the viewer’s expectations and perceptual abilities. To handle this, we developed a log-scale technique based on the same general concepts as our spatial scaling.

To specify rotational periods, orbital periods, or other periodic motions of astronomical datasets, we enter the triple: \( (\text{magnitude scale unit}) \) in the dataset description file, which stands for \( \text{magnitude} \times 10^{\text{scale unit}} \). For example, \( (3.65 \text{ day}) \) expresses the same information about the Earth’s orbital period as \( (1 \text{ year}) \). The system automatically converts the input to seconds, \( (3.1536 \times 10^7 \text{ seconds}) \) and stores the period of motion as \( \log_{10}(3.1536 \times 10^7) = 7.4988 \). This \( \log_{10} \) value is denoted as \( t_{\text{motion}} \) for every defined motion.

In parallel to the spatial scale \( s_{\text{spatial}} \) we defined in Chapter 4, we define an animation scale, \( t_{\text{animation}} \) with this property:
Quantitatively, this time variable specifies the equivalence between one second of observer's screen time (the wall clock) and $10^{t_{\text{nav}}}$ seconds in the simulated Universe.

That is, if $t_{\text{nav}}$ is 7.4988, we could observe the Earth in the virtual Universe going around the Sun once every second. Using this rule, we can determine the observable period of motion, $t'$ (in seconds), for a specified $t_{\text{nav}}$ using the following formula:

$$t' = 10^{t_{\text{motion}} - t_{\text{nav}}}.$$

Consequently, if we want to visualize the rotation of the Earth on its axis, we can reduce $t_{\text{nav}}$ from 7.4988 to 4.9365, which corresponds to a time scale of a day. By controlling $t_{\text{nav}}$, we can manipulate relative simulation time scales to optimize the visualization properties of the simulated motion.

**Step sizes in Time**

Note that when computing the above equation, we should avoid computing $t'$ immediately. Instead, we have to determine the difference between $t_{\text{motion}}$ and $t_{\text{nav}}$ and make sure that their difference is not large enough to overflow $t'$. Indeed, during our initial development, due to numerical errors, we found that the Earth vanished whenever $t_{\text{nav}}$ became too large. To correct this, we need both to check the ranges of $t_{\text{motion}}$ and $t_{\text{nav}}$, and to support time-scale-dependent representations of the objects. Only then can we construct intuitively correct visualizations of object motions when $t_{\text{motion}}$ and $t_{\text{nav}}$ are substantially different.
13.3 Motion Visualization Methods

Based on the $t'$ consideration in the previous subsection, we then end up with three basic animation categories: those with time scales close to the scale appropriate to the screen time, those that are too slow for us to deduce the properties of their motions, and those that move so fast that ordinary animation would produce drastic sampling-error effects. Special attention has to be paid to scales of motions that are far out of step with the current screen time.

Quantitatively, we can once again make use of $t_{\text{nav}}$ and $t_{\text{motion}}$ to select different model representations for different situations:

$$t_{\text{motion}} - t_{\text{nav}} \begin{cases} > t_{\delta_1} & \text{implies too-slow motion} \\ \in [t_{\delta_2}, t_{\delta_1}] & \text{implies screen-time motion} \\ < t_{\delta_2} & \text{implies too-fast motion} \end{cases}$$

where $t_{\delta_1}$ and $t_{\delta_1}$ are pre-defined thresholds specific to different kinds of motion. Note that we have $t_{\delta_1} > 0$ and $t_{\delta_2} < 0$. In essence, this is a kind of level-of-detail (LOD) strategy in time: we call this strategy temporal-LOD, which is parallel to the spatial-LOD strategy.

Screen-Time Objects

Figure 13.1 shows a simple example of the depictions of astronomical objects whose motions span a range of scales that are still animatable at screen scales.

Our design assigns to the Moon a finite length "trail" that serves as a visualizable velocity field; thus, even in a still image, we can tell that the Moon is moving at a certain velocity relative to the overall scale. Similarly, we blur the trails of very fast objects: in the figure, the space shuttle is the object represented by the already circular blurred orbit...
13. Time Management in the Universe

Figure 13.1: The Earth with time-streaked satellite animations and static stars and galaxies.

... trail; it is starting to exhibit sampling errors, and will have the icon turned off and completely replaced by the orbit icon if we increase the rate of time evolution in this scene. The Earth itself is making the transition to a motion-blur-anti-aliased texture since the sampling rate per refresh is getting too close to the rotation period. The main point of this multiple-component visualization is to show our design emphasizing that even objects that are moving slowly enough to animate directly in screen-time intervals may have diverse properties; differing velocities in particular need to be distinguished so that the user can intelligently focus on interesting properties and exercise appropriate time scale adjustment.

Slow Objects

Just because an object is moving so slowly that it cannot be perceived as moving on the real-time graphics screen does not mean that we do not need to know it is moving. In fact, such objects may have extremely significant motions in the context of the entire visualization, and therefore we may need somehow to annotate their past and future positions, however slowly they may move in screen time.
Figure 13.2 shows schematically how a moving object in the scene is depicted if it has a significant past and future, but is static at the current time scale. Fundamentally, we present a trace of the object’s predicted trajectory coupled with scale marks showing how many powers of 10 are needed to reach a scale of visible motion starting with the current screen-time scale. The interactive interface will typically have one additional control that allows the user to read just the screen time. That is, the scene model itself together with its navigation fields will have a “default” screen time attached to the scene corresponding to the viewpoint and spatial scale; since there may be many moving objects in the scene that are of interest, but with vastly different time dependence, the user needs a way to adjust the time scale up and down to check the intuitive behavior of each object. As this is done, of course, objects migrate among the possible representation categories -- a static object may start to move in screen time, and vice versa.

**Fast Objects**

Conversely, for a given choice of screen time, some contextually significant objects may nevertheless be moving far too fast to display meaningfully in the visualization. They
may not appear on the screen at all, or perhaps may appear in a single time-sampled frame and disappear, and yet it may be important to know that they passed through the field of view and are on their way elsewhere.

Figure 13.3: Motion-blurred trails of objects moving, orbiting, or rotating too fast for animation in current screen time.

Figure 13.3 sketches a representation of an object that is moving too fast to perceive continuity at the screen pixel level, one that is moving in a fast periodic orbit, and one that is rotating too fast to see coherent surface texture warping. Non-periodic paths are denoted by lines or curves with time scale notation to indicate how to slow down the motion to render it simulatable at screen scale. Periodic motions are represented as closed paths, again with a time scale; periodically moving objects centered on a moving body may be shown pedagogically as a single elliptical curve rather than the more correct but less informative spiral. Interactive time scale adjustments can be made as usual to match the screen time to a particular object.

Figure 13.4(a–c) illustrates the level-of-detail strategy for circular orbit trails in the Solar system. As depicted in the figure, we can use the lengths of neighboring comet trails to quantitatively visualize relative motions of each planetary or satellite orbit in the Solar system. For "too slow" motion, we display variable-sized arrows (or dotted lines as Figure 13.2) indicating the direction of motion.
13. Time Management in the Universe

Figure 13.4: "Too fast" representations at (a) roughly 300 years per screen second and (b) roughly 30 years per screen second. (c) "Too slow" representations for the motion of planets in our Solar system at a scale of approximately 10 days per screen second.

Figure 13.5: Motion Blur Representation: (a) Normal texture. (b) Texture blur for one rotation in two screen seconds. (c) Texture blur for one rotation in less than one screen second.
On the other hand, it is worth noting that certain common astronomical models such as planets and moons not only move in their orbits, but spin on their axes. When the object's surface is represented by a complex texture, this rotation creates a texture representation problem exactly analogous to the too-slow/too-fast orbit problem. Whenever the textured object reaches a time scale where the texture is moving by a large amount in a unit of screen time, the motion loses smoothness and undesirable effects such as stroboscopic aliasing (time aliasing) become apparent. We handle this problem by adopting a texture-based motion blur method (see Figure 13.5) that gives the object a smoother appearance as it speeds up as well as eliminating stroboscopic effects. For very fast rotation speeds, the texture turns into blurred bands exactly analogous to the satellite trails.

Indeed, the problem of reconstructing the path or appearance of a fast object and representing it meaningfully is closely related to the problem of digitally sampling analog data; the Nyquist theorem says that the sampling rate of a signal must be at least twice the frequency in order to accurately capture the periodic content of the signal. The legendary problem of motion blur in computer graphics, along with the classic “wagon wheel” stroboscopic freezing of rotating wheels in films, are essential problems that challenge the requirements of accurate sampling; blending together many finely spaced samples to construct a blurred image or a comet trail is a classic approach to producing an intuitively satisfying visual representation.

13.4 The Interaction Graph

On the other hand, as mentioned in the first section, there is no static reference frame in our dynamic Universe; all objects are constantly moving if we are animating the Universe. We thus need some way to keep track of such a dynamic world. Essentially, there are two basic kinds of interactions among entities in our Universe:
• **Orbiting.**

Periodic orbits of one dataset around another. Examples are the Earth and the Sun, the Moon and the Earth, the Asteroids and the Sun, Saturn’s rings and Saturn, etc.

• **Locking.**

One dataset maintains a fixed position or orientation with respect to another dataset. An example is the use of different observational coordinate systems such as polar coordinates relative to the Earth (Equatorial Coordinates), relative to the Sun (Ecliptic Coordinates), relative to the Galaxy (Galactic Coordinates), or relative to the whole set of galaxies (Super Galactic Coordinates). Since a dataset of nearby stars is typically expressed in Equatorial coordinates, we can lock the dataset to the Earth to maintain the observing position relative to the Earth.

In this way, when we animate the orbital motion of the Solar system around the Galaxy, the coordinate locking mechanism forces the nearby star dataset to orbit accordingly around the Galaxy. Similarly, we can lock local galaxies, distant galaxies, and the Cosmic Background Radiation dataset to the Galaxy.

Figure 13.6 depicts interaction relationships as a graph, called the *Interaction Graph*. In the graph, each dataset is represented as a node and each node-to-node interaction is represented as an edge. Although this interaction graph structure looks similar to the scene-graph management structure (the tree structure storing geometry for the rendering process) in software like Open Inventor, VRML, or OpenGL Performer, the interaction graph differs substantially from the scene-graph: No particular dataset is singled out as the root node and no geometry is necessary to associate with the nodes. In a practical sense, the interaction graph serves mainly as a data structure that 1) organizes the datasets by their interactions, 2) facilitates scaling, and 3) keeps track of the animation process. This is especially useful for navigation control and planning, given that the environment is dynamic.
Figure 13.6: The Interaction Graph for the Universe dataset collection.
13.5 Conclusion

In conclusion, we proposed in this chapter a collection of techniques suitable for animation management in large spatial scale as well as in large temporal scale. In terms of time management, we established a log-scale framework to treat temporal scale in parallel to the spatial scale approach in Part I of this dissertation, so that we could select various temporal-LODs based on the specified screen time. Furthermore, we introduced the interaction graph structure to govern the animation process so that we could keep track of relative motions or positions in the dynamic environment.
Part IV

Large-scale Navigation
In the previous parts of this dissertation, we presented various topics concerning large-scale visualization, including our framework for handling large spatial scale in Part I, various methods for modeling and rendering an assortment of astronomical bodies in Part II, and some of our techniques for handling large scales in time and visualizing cosmological time in Part III.

In this part, we focus on navigation: 1) problems and issues concerning navigation in large-scale environments, 2) different models for large-scale navigation, including particularly

- Unaided Navigation
- Spatial-scaling Navigation
- Path-based Navigation
- Constrained Navigation (Manifold-based) [78, 179, 83]

as well as 3) some rendering considerations and visual cues for large-scale navigation.
14

Astronomical Navigation

14.1 Introduction

Navigating in virtual environments is difficult in computer graphics; users are generally given limited depth perception and a screen-sized field-of-view when making their navigation decisions. This has motivated the development of stereoscopic rendering and immersive environments such as the CAVE and ImmersaDesk. Astronomical environments present still harder navigation challenges.

In practice, when navigating in astronomical environments, we have to face the following problems: First is the fact that the astronomical environments are huge in spatial scale; we have to take care of numerical precision for camera positioning as well as various navigation parameters. We use power homogeneous coordinates for spatial representations in order to take care of this problem as described in Chapter 3. Second, again because of the large-spatial-scale problem, we have to navigate around different objects at the right speed since different astronomical bodies differ greatly in size. Third, due to the fact that astronomical bodies are separated by large scales, we may easily get lost in the empty space between them.
This chapter focuses on handling navigation in large-scale environments, realized in the context of astronomy. In Section 14.2, we will point out in detail various characteristics of astronomical navigation, and illustrate how it is different from ordinary navigation in daily-life. Then, in Section 14.3, which attempts to deal with these issues, we will present various navigation models and show how they can be employed for large-scale navigation. After that, Section 14.4 will discuss rendering as related to navigation: how we can accelerate rendering performance by making use of the navigation information, and how we can assist navigation using visualization cues. We conclude in Section 14.5 with our conclusions and a discussion of possible follow-up work.

14.2 Characteristics of Astronomical Navigation

Before we present the collection of models for astronomical navigation in Section 14.3, we would like to discuss some characteristics of astronomical navigation. By this means, we hope that the material presented in this section can help readers to justify the design of future navigation models and to better understand why various constraints are needed in the navigation context.

Our navigation issues are mostly related to the fact that our Universe is huge in spatial scale. Though we have been talking about “large spatial scale” again and again in this dissertation, when dealing with navigation, the consequence of “large spatial scale” is a different story.

1. Navigation Speed or Step size

The first characteristic related to “large spatial scale” is the navigation speed, or the step size in the movement. In daily-life navigation, we can have $0.5 \, \text{m/s}$ for walking, $223.159 \, \text{km/h}$ for driving a car.
MPH for car racing, and 760 MPH for traveling on a Mach 1 plane. But for astronomical navigation, we need to adapt the speed for different situations.

For example, to travel among planets inside the Solar system, we may have chosen a certain range (scale) of navigation speed. But if we travel with that same speed range inside the interstellar space or intergalactic space, we will see every star or galaxy being static since they are just too far away from us compared to the traveling distance generated by the relatively low speed. On the other hand, if we travel at interstellar or intergalactic speed inside the Solar system, we could quickly get lost in deep space and never come close to any planet since such speeds do not provide sufficiently fine control.

2. Exceeding the Speed of Light

The other characteristic of astronomical navigation is that we have to violate the laws of Physics to some extent. Sunlight takes about 8.5 minutes to reach us on Earth, whereas starlight from our nearest stars, the Alpha Centaurus system, takes about 4.3 years to reach us. These numbers mark the limiting travel time for visits to these bodies from Earth. If we insist on following this same physical rule in the virtual environment, we will end up wasting our time in the simulation; no one working on space simulation insists on the speed of light limit for investigating interstellar structure. What we normally do is to allow speeds beyond light speed during the navigation. In a word, we have to ignore the effects of relativity during the navigation out of sheer practical need.

3. Getting Lost in Empty Space and Spatial Perception

As discussed in Chapter 12, the Universe has gone through the cosmological eras of radiation and matter, and reached the state of a vacuum-dominated Universe. Indeed, in this vacuum-dominated era, astronomical bodies are largely separated from one another. In terms of navigation, this spatial emptiness among objects poses two hard problems:
First, when we move from one object (planet, star, or galaxy) to another, we may easily get lost on the way between them. It is also hard to find our way back during the navigation. Second, unlike daily-life experience, in which we can normally identify some landmarks, buildings, or signs relative to the origin or destination of travel, the space separating astronomical objects is essentially empty. It is hard for navigators (users) to perceive the spatial context around their location until they finally reach the destination. In other words, the size of space is hard for users to perceive during the navigation. As a consequence, users may feel that distances among stars and planets are more or less the same because traveling at the two scales takes approximately the same amount of time in the computer simulation.

4. Datasets are Earth-centered in nature

It is interesting to note that because most astronomical datasets, except those that are Milky-Way-related, are either captured on Earth or by Earth-orbiting satellites; thus, the captured datasets are generally Earth-centered, or symmetrical around the Earth at the origin. This data feature implies that we actually navigate in an Earth-centered data space. Typical datasets demonstrating this feature are the Hipparcos Catalogue for stars and the Galaxy catalog from Prof. Brent Tully (see Appendix A.3 for details about these datasets). Figure 14.1 illustrates these two datasets, for which the Earth defines the center. Note that the yellow scale-marking cubes help to reveal the spatial scales of these datasets.

5. Field-of-view and Viewing Direction

Finally, we have to take care of the field-of-view (zoom factor) in the navigation. In human vision, our eyes can span up to a 200 degree field-of-view when looking at the environment; using binoculars to look at the moon, we have a field-of-view from 0.5 to 1 degrees; but when we look at stars or galaxies in the deep space, we need high power telescopes with a very narrow field-of-view, as small as 0.1 arc-seconds.
Figure 14.1: Most datasets are Earth-centered in nature.

Figure 14.2: Controlling Field-of-view linearly and logarithmically in different ranges.

---

The Hipparcos stars

Galaxies

PROCEDURE UPDATE FOV

INPUT:
- fov /* Current Field-of-view */
- dmouse /* Change in Mouse Coordinate in X */

BEGIN
/* Convert to X */
IF fov > k THEN
    x = (fov - k) * S
ELSE
    x = log (fov / k) * Sn
END

x = x + dmouse

/* Convert back to fov */
IF x > fovMax THEN
    x = fovMax
ELSE
    x = k * EXP (x / Sn)
END

RETURN x
END

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
In practice, we can see from the above numbers that if we use a mouse drag or 2D slider to control the field-of-view proportionally, we cannot have fine control to get a very narrow field-of-view value. Figure 14.2 shows our proposed response curve to solve this problem by providing a non-linear field-of-view control relative to mouse interaction: The response curve is linear when \( fov > k \), while it is logarithmic when \( fov \leq k \), where \( fov \) is the current field-of-view; \( k \) is the threshold in the field-of-view domain, and \( x \) is the parameter in the mouse-motion domain. Mathematically, we have

\[
fov = \begin{cases} 
  k + x/S_l & \text{if } x \geq 0 \quad \text{(linear)} \\
  k \cdot e^{x/S_n} & \text{if } x < 0 \quad \text{(logarithmic)} 
\end{cases}
\]

Note that \( S_l \) and \( S_n \) are sensitivity factors such that the larger their values the more flatten the curve is for corresponding part.

Corresponding pseudo-code is presented on the right-hand side of the figure: \( fov_{\text{Min}} \) and \( fov_{\text{Max}} \) define the limiting fields-of-view in the viewing. Note that since the CAVE or the dome (planetarium) environment has a fixed field-of-view, this interactive technique is not applicable in environments like this, but in practice, this field-of-view control feature is built into most of our developed tools (see Chapter 15).

Another essential concern during navigation is that we should avoid motion sickness. The precautionary measure with field-of-view is that when using a very narrow (small) field-of-view, the rate of change in the viewing direction has to be very small so that our view on the screen is not changing too rapidly. Another situation is the dome environment in a planetarium: Due to the fact that the dome fills most of our 200 degree field-of-view (in human vision) as compared to that of a computer monitor, small changes in viewing direction are much more obvious in the dome. We have to keep the rate of change in viewing direction in the dome to be much smaller than in other situations to avoid motion sickness.
14.3 Navigation Models

Overview: Navigation Models

Based on the discussion in the previous section, we hope to have helped readers to understand the fact that unaided or unconstrained navigation model is inappropriate for astronomical navigation. We conclude that we need certain constraints imposed on the navigation system to avoid the problems just described, and to assist users to readily locate objects of interest during the navigation process.

In this section, we describe four different navigation models:

1. Unaided Navigation Model
2. Spatial-scaling Navigation Model
3. Path-Based Navigation Model
4. Constrained Navigation Model (Manifold-based) [78, 179, 83].

Each model will take up one subsection in the following. To give users some hints when going through these models, we note that the chosen order corresponds to the fact that the complexity of these models increases from model (1) to model (4).

The Unaided Navigation Model

The first navigation model in our list is the "unaided navigation model." Note that we avoid calling it a "free" or "unconstrained" navigation model because in reality, there exist certain very primitive constraints given by the user interface, e.g., 2D mouse control, center of rotation in the virtual world, etc. On the other hand, the word "unaided" does not
mean that there is no navigation guidance in the model. Rather, it refers to the fact that no specific aid is provided for large-scale navigation. To facilitate the discussion, we will talk about the VRML [12] browser, typically the Cortona browser by Parallel Graphics, as an example. Figure 14.3 shows the interface of the VRML browser embedded inside Internet Explorer. The VRML model shown in the figure is a small prototype environment we constructed for gas cloud study at a scale about $10^{17}$ meters. The Heliopause model, which is texture-mapped with a flaming image, is shown around the middle of the figure (see Section 8.6 for information about the Heliopause). Note that we have to avoid putting objects with huge scale differences in the VRML world because VRML browsers do not take care of large-scale rendering issues.

![Figure 14.3: The Cortona VRML browser showing the Heliopause in the stellar space.](image_url)
In general, there are three basic navigation controls in the browser interface: Walk, Fly, and Examine (or Study) (see the help menu in the Cortona browser for detail). In “Walk” mode, movement in the virtual world is relative to the user’s current location: Users can move forward or backward along the screen-center direction or turn (or move) left or right. Similar to the “Walk” mode, the “Fly” mode is relative to a user’s current location, but instead of moving forward or backward, it allows users to move upward or downward. The “Examine” (or “Study”) mode works differently: It exhibits rotation or tilting relative to the world center, which is defined in VRML as the center of bounding box of the geometry in the 3D scene. Thus, navigation with this mode is non-local compared to the others. Nevertheless, the “Examine” mode can help users to readily examine the whole 3D scene or a single 3D object. Note also that the “plan,” “pan,” “turn,” and “roll” in the browser interface are sub-categories under the three basic modes.

These navigation controls are common in many 3D browsers, and using these controls is fairly effective when we navigate in daily-life 3D environments or examine 3D objects. However, in terms of astronomical navigation, since the spatial scale is huge, this navigation approach is inadequate to take users where they want to go due to navigation speed problems as well as the lost-in-empty-space problem (see the previous section).

Due to their generality and simplicity, these navigation controls are included in most astronomical navigation applications. However, in order to deal with the “large spatial scale” problem, most applications provide a “goto” command in the interface so that after the user types in an object name or mouse-clicks on a faraway object (e.g., a star), the system will fly toward that object in a few seconds. For example, Celestia [54] implements this feature like this: turn the viewing direction toward the selected object and then move straight toward the object. This is a common feature for assisting large-scale navigation in the unaided navigation model.
The Spatial-scaling Navigation Model

The second navigation model we describe is one step more advanced than the unaided navigation model; we call it the *spatial-scaling model*. Though the basic user controls provided by this navigation model are exactly the same as for the unaided model, we add to it one crucial feature to make it usable for large-scale navigation – spatial scaling.

In essence, the idea of spatial scaling is simple and is already embedded in our modelview transformation mechanism in Chapter 4. As a quick review, recall that we introduced the variable $s_{mv}$ in Section 4.3 to denote the spatial scale associated with the modelview matrix. In that context, we generalized the translation component of the modelview matrix to a power homogeneous coordinate (or vector) $(T_x, T_y, T_z, s_{mv})$:

$$
\text{modelview translation, } T = (T_x, T_y, T_z)^t \times k^{s_{mv}},
$$

where $k$ is the exponent base in PHC, typically chosen as 10. In this way, 1 unit in the virtual environment corresponds to $10^{s_{mv}}$ meters in our physical space; we now apply the same convention to the navigation. In other words, when we adjust the $s_{mv}$ parameter, we are either expanding or shrinking the whole Universe relative to the virtual space using a logarithmic scale. When dealing with large-scale modeling and rendering, this rescaling method helps present models at their most appropriate scales around the viewer in 3D; this method is basic to our large-scale framework, along with the depth rescaling method proposed in the same chapter. For details about $s_{mv}$, readers are referred to Section 4.3 for the in-depth mathematical derivation of the modelview transformation.

When dealing with large-scale navigation, this method permits us to locate datasets across different scales instantaneously: We only need to adjust $s_{mv}$ to the scale corresponding to the dataset we want. Furthermore, since the whole space is rescaled (either reduced or expanded) relative to the virtual environment that is at the ordinary scale, we never have
to worry about the navigation speed or the step size in the movement; we are always navigating at the same speed or step size because we are rescaling all models against our fixed scale. Nevertheless, on the other side of the coin, we can also view this as the situation where our motion step size varies as $10^{s_{inv}}$. As an illustration, we refer the reader back to Section 4.3, where we discussed the effect of adjusting $s_{inv}$, and presented a corresponding animation sequence in Figure 4.2.

*The Spatial Center.* So far our discussion has neglected the fact that every "scaling" needs an origin to refer to. Indeed, we also need a reference origin for the spatial-scaling method. Based on the Earth-centered nature of our data as discussed in the previous section, the Earth is chosen to be the spatial center in most cases, which means that the global scaling is done relative to the center of Earth. At the same time, we have implemented the "Examine" and the "Study" mode relative to the Earth as well.

![Figure 14.4: Global centers along the Interaction Graph.](image-url)
This is generally right when the world is static; however, if we are animating the astronomical environment as in the case we demonstrated in Chapter 13, putting the Earth at the center would be odd when we are at the scale of the Solar system. We will find the Sun orbiting around the Earth and the Earth serving as the center of the Solar system; thus, shifting the center to the Sun would be a better choice. To deal with this, we make use of the Interaction Graph (Figure 13.6), which was introduced in Section 13.4 for astronomical animation, to take care of this centering problem. Now, if we pick up a sequence of objects in the interaction graph as spatial centers at different scales, the navigation model can automatically shift the global center as appropriate.

Figure 14.4 depicts a typical example based on the interaction graph in Figure 13.6: The center is specified to be the Earth when \( s_{mv} \leq 10 \), the Sun when \( 10.5 \geq s_{mv} \leq 19.5 \), and the galaxy when \( 20 \leq s_{mv} \). In effect, the central object of the current \( s_{mv} \) is the origin of the virtual environment. For in-between scales such as \( 10 \leq s_{mv} \leq 10.5 \), this navigation model interpolates between central objects (the Earth and the Sun) to produce a smooth transition. In this way, we can naturally examine astronomical environments at different scales.

The Path-Based Navigation Model

We present in this subsection the path-based navigation model, which advances one more step over the previous model by providing a navigation path in the virtual environment. Compared to an ordinary camera path in a conventional animation system, the path used in this navigation model spans large spatial scale changes so that it can guide users across the gigantic astronomical space. This path can also be useful for generating animations.

Constructing a Path (Sequence of Keyframes). Like an ordinary camera path, the path in this model consists of a sequence of keyframes. Each keyframe consists of a camera position, a camera (or viewing) orientation, a field-of-view, and an aspect ratio. As usual, we represent all camera positions as power homogeneous coordinates and make use of the computation methods defined in Chapter 3. Figure 14.5(a) depicts the basic components of a keyframe,

- **position**: camera position in power homogeneous coordinates
- **(viewdir,viewup)**: viewing (look-at) and view-up directions
- **fovy**: vertical field-of-view (in degrees)
- **aspect-ratio**: ratio between vertical and horizontal field-of-view,

and Figure 14.5(b) shows a sequence of keyframes constituting a navigation path. To determine intermediate view configurations between keyframes, we make use of the Bezier spline interpolation method: Position, camera orientation, field-of-view, and aspect ratio...
are independently interpolated. However, it is important to note that to interpolate camera orientation, we need to convert each pair of orientation vectors \((\text{viewdir}, \text{viewup})\) to quaternions; then, we can correctly and effectively interpolate camera orientations between keyframes in the quaternion domain rather than interpolating each of the two orientation vectors. Rules for conversion between quaternions and orientation vectors \((\text{viewdir}, \text{viewup})\) can be found in [79].

Figure 14.6 shows the interface of our program, the *moviePET* (PET stands for Path Editing Tool), for constructing camera paths. Two typical camera paths created for the *Solar Journey* animation are shown in the figure. In the interface, the top-left, top-right, and bottom-left subwindows display the front view, the side view, and the planar view of the path, respectively, while the bottom-right subwindow shows the camera path together with the three-dimensional rendering. The camera configuration at each keyframe is depicted as a wireframe along the path, with red dots indicating locations of interpolated frames between keyframes.

*Using the Navigating Path.* The advantage of using a navigation path for astronomical exploration is obvious: We avoid the lost-in-empty-space problem because our camera is attached to the navigation path at each frame. In fact, during the path-based navigation, users can choose to stay on the path for the whole journey and let the system fly them along their way. This is the animation mode for the navigation. Or, if the user finds something interesting along the way, he/she can move out of the path and navigate around that object as in the spatial-scaling model.

**The Constrained Navigation Model (Manifold-based)**

*The idea of Constrained Navigation.* The final navigation model that we present in this chapter is the Constrained Navigation model [78, 179, 83] worked out together with Dr.
Path in the Solar System

Path in the interstellar space (Loop I is shown in the middle)

Figure 14.6: Constructing the Navigation/Animation Path using the "moviePET."
Eric Wernert [178]: It can be regarded as a generalization of the path-based model because it generalizes the concept of navigation path to a navigation manifold. Customarily, we can define constrained navigation to be the assignment of a mapping between a controller space and a general field of viewing control parameters as depicted in Figure 14.7.

![Diagram of constrained navigation model](image)

**Figure 14.7: The Constrained Navigation Model.**

Note that the viewing control parameters form a grid structure and the set of viewing control parameters at each grid point is like a keyframe defined in the path-based model. Our typical implementation involves the design of one or more 2D “sidewalks” that define the 3D spatial motion of the user and the view parameters in response to inputs in a limited controller space such as that of a mouse or the CAVE thumb joystick. Since the response of the user displacement to a unit of controller motion can be completely controlled and customized by the fields stored with the navigation manifold, we can easily adapt the motion...
response to meet the user requirements of large-scale navigation. In particular, it seems obvious that a “Powers of [Base k/10],” i.e., logarithmic, scaling of the response is the natural one to use: the farther we get from the Earth’s surface, the lower the density of detailed observational data, and the larger the scale of the visible structures that are interesting to depict.

**Using Constrained Navigation.** In addition to scaling motion control with the constrained navigation framework, we can have a number of constraint manifold designs that are well-suited to this work: among these, we describe below the “wedge” and the “twist.”

**Wedge Exponential Journey.** In Figure 14.8, we show a symbolic journey on a wedge-shaped navigation manifold; the width of the wedge as well as the actual distance traveled per unit controller motion expand exponentially with distance from the origin. This allows the viewer to pursue a “Powers of [Base k/10]” interactive exploration of the space, and to travel between viewpoints in logarithmic time instead of the much less appropriate constant velocity (linear time).

**Twisting Reorientation.** In Figure 14.9, we show the advantage of the constrained navigation approach for huge scales with smooth evolution between different orientation frames natural to each scale. Beginning with a “wedge” manifold, we twist the frames to get custom orientations. Beginning with an orientation suitable for viewing the Earth, we move out seven orders of magnitude, while twisting so the Solar system appears horizontal instead of aligned to the Earth’s equator; moving out seven more orders of magnitude to the galactic scale, we twist again to orient ourselves to the galactic plane.
Figure 14.8: Images corresponding to three viewpoints on the logarithmically scaled "wedge" path from the Earth to the Moon.

Figure 14.9: Images corresponding to three viewpoints on the adaptive twisted-orientation constraint manifold encompassing the Earth, the Solar system, and the Milky Way galaxy.
14.4 Rendering Issues

In the previous section, a series of four navigation models were introduced to take care of large-scale navigation. Alongside that, we need some rendering strategies to maximize our navigation power both in terms of performance and visualization cues. In this section, we first present our cycling mechanism based on the two criteria developed in Part I of this dissertation, and then explain visualization methods for assisting large-scale navigation.

Cycling Mechanism: Environment Caching and Object Disappearance

During astronomical navigation, users traverse the huge variation of scales in the Universe. Astronomical bodies located at extreme scales could be too far away from the user's current navigation region to be visible by the user in three dimensions. This situation has been addressed by the Environment Caching Criterion and the Object Caching Criterion presented in Chapter 6 and Chapter 7, respectively. In short, the two criteria tell us the following:

**Environment Caching Criterion**

Given a safety region of radius $10^S$ units, we can pre-render any object beyond $10^{S+\delta(k)}$ units from the region. During the program runtime, if the camera moves within the safety region, instead of rendering faraway objects in three-space dimensions, we can substitute a pre-rendered image at a $k$ pixel-unit accuracy.

**Object Caching Criterion**

For an object of radius $10^R$ units, if the camera moves beyond $10^{R+\delta(k)}$ units from the object center, the object will be smaller than $k$ pixel units on the camera screen.
Note that the function $\delta(k)$ depends on the resolution and field-of-view of the camera; details of its derivation can be found in the related chapters in Part I.

These two criteria help us to determine the representations of astronomical bodies when we navigate at different scales. One nice example is the Milky Way modeling and rendering in Chapter 10: When we navigate at the scale of the Solar system, the Milky Way Galaxy is so far away from us that it appears like a two-dimensional image on a celestial ball; it is thus rendered as a panoramic image as if it were infinitely away from us. This is precisely realized by the environment caching criterion. Then, when we move out to interstellar space, in which stars begin to appear in three dimensions, the environment caching criterion fails for the Hipparcos stars and we can no longer show these stars on the 2D Milky Way panorama. We thus have to filter out these star images (pixels) from the panoramic image. Nevertheless, at the same time we reach interstellar space, the object disappearance criterion comes into play and tells us that the Solar system is now too small to be viewable, so we can ignore its rendering. Finally, if we continue our journey beyond the local stars, the Milky Way itself breaks the environment caching criterion and needs to be truly represented in three dimensions.

In principle, this example demonstrates how navigation information can be effectively used to speed up rendering. It helps us to pick up an appropriate representation for the Milky Way in different navigation situations and to avoid redundant rendering during the program runtime. As a result, when we navigate from the Solar system up to intergalactic space, the system can cycle through different representations of various astronomical bodies so that we never need to render all of them truly in 3D at the same moment.
Powercubes Visualization

In the previous subsection, we discussed the cycling mechanism, in which navigation information is used to assist rendering. In this subsection as well as the following subsection, we will present two visualization methods for assisting navigation.

The first visualization technique is called the powercubes method, which provides landmarks to help users identify the scale of space during the journey. In essence, this method helps to alleviate the lost-in-spatial-context and lost-in-empty-space problems. This method is inspired by the spatial scale squares shown in the famous Eames Powers-of-10 film [29]. In our case, when going through the huge astronomical space in 3D, instead of showing 2D spatial squares, we render a nested set of 3D cubes concentric at the object of interest to reveal the spatial sizes in a similar way. We call these 3D cubes the powercubes; each powercube takes a powers-of-10 size in the exponential scale similar to the spatial squares shown in the Powers-of-10 film. Note also that the corresponding spatial size of the edge of each cube is labeled at the bottom front of each cube for visualization purposes.

As an illustration, Figure 14.10 demonstrates the use of powercubes when we travel from the interstellar scale toward to the Solar system: The sequence of images goes from captions (a) to (l); our camera moves from a distance of $10^{17}$ to $10^{11.5}$ meters from the Sun. In the first two frames, which correspond to scales of $10^{17}$ and $10^{16.5}$ meters, stars appear in three dimensions around our viewer. However, starting from $10^{16.0}$ meters, we can see from the images that most stars stay in the same location on the background without any motion at all. Although nothing seems to be moving from subimages (c) to (e), the images are correct because there is a large spatial gap between interstellar scale and the Solar system scale. During this journey, nothing significant changes in the renderings until we reach $10^{14.5}$ meters; here the Solar system appears as a bright spot in the middle. During this spatial transition, the powercubes play an important role, and demonstrate their usefulness for revealing the spatial scale to the user. In this example, these powercubes serve to
Figure 14.10: Visualizing spatial context using Powercubes.
annotate the spatial gap between the stellar scale and the Solar system scale.

**Minimum Spanning Tree Visualization**

Another visualization strategy we developed for assisting navigation is the *Minimum Spanning Tree visualization* technique. The situation when we navigate among stars is that there are several thousand stars floating in the space around us; though the rendering size of stars can be adjusted dynamically, it is too hard to visualize the spatial context among them. The situation is similar to the problems encountered in information visualization when there is too much information to visualize in the virtual space.

We propose using the minimum spanning tree [139] to expose the spatial context among stars. As a quick review, the minimum spanning tree is defined as follows: In an undirected connected graph, $G = (V, E)$, where $V$ is a set of vertices and $E$ is a set of edges connecting vertices in $V$, a cost value (or length) is associated with each edge in $E$. The minimum spanning tree is defined as a connected acyclic subgraph that covers all vertices in $V$ with the smallest total cost, which is measured as the sum of costs in all its edges.

In our case, each star is regarded as a node in $G$ and is connected by an edge to every other star, and the cost of each edge is defined as the distance between each associated pair of stars. Then, Prim's algorithm [139] is applied to determine the minimum spanning tree in this graph structure. In this way, by drawing the edges belonging to the minimum spanning tree among stars, we can better visualize the spatial context among them.

Figure 14.11 shows some star rendering examples alongside edges from the minimum spanning tree. In Figure 14.11(a), we are still at a scale of $10^{14}$ meters from the Sun, but by showing the minimum spanning tree edges connected to the Sun, we can visualize some neighboring stars in our view. When we move out to a scale of $10^{10}$ meters, we can use the edges to detect the spatial proximity among stars during the navigation.
Figure 14.11: Visualizing spatial context among stars using a Minimum Spanning Tree.
there are too many edges in the minimum spanning tree, we draw only edges that appear in three dimensions around the viewer.

### 14.5 Conclusion and Future work

In conclusion, navigation is a hard problem, but crucial for achieving better visualizations. Although it is often overlooked by computer graphics research, it is indeed a useful, interesting, and rewarding research area. In this chapter, by exploring various models for large-scale navigation, we show that large-scale navigation is even harder than ordinary navigation. Various constraints are introduced into the navigation system in order to help users to keep track of the spatial context during their manipulations; two different visualization strategies are adopted to reveal the spatial context during the navigation.

Navigation is a potential research area of high interest, and there are many possible subjects for future work based on the techniques presented in the chapter:

- First, we observe that in the path-based navigation or the constrained navigation model, the navigation path or the navigation manifold is fixed before application runtime. This is of course inadequate if users intend to go somewhere not accessible from the path or manifold. Having the ability to generate the navigation path or manifold on-the-fly would be valuable.

- Another similar situation is that if the Universe is dynamic, using a static path or manifold to fly around the space would be insufficient. On-the-fly navigation path generation or manifold planning is needed.

- On the other hand, if we are using the path-based navigation or the constrained navigation model, we indeed could be able to predict the navigation boundary, or the
safety region as described in Chapters 6 and 7, beforehand or during application runtime. In this way, we can develop a data caching algorithm to pre-fetch datasets in a more efficient way based on navigation prediction.

It would be interesting to correctly incorporate appropriate aspects of relativity, the effects and visualization of the finite speed of light, and the transformation of optical images in both shape and frequency due to relativity [94, 93, 176, 175]. Yet more ambitious would be the inclusion of general relativistic gravitational effects, e.g., when traveling near a black hole at the center of a galaxy.
Part V

Results, Conclusion, and Future work
The last part of this dissertation consists of two chapters. Chapter 15 presents our development results: an assortment of visualization tools and the two animations, "Cosmic Clock" and "Solar Journey," produced in this research work. Finally, Chapter 16 presents the conclusions and highlights the ideas and techniques developed in this research work.
15 Results

15.1 Introduction

This chapter presents the system development results of this research work. Based on the modeling and rendering techniques described in previous chapters, we created an assortment of visualization tools and two animations for educational purposes. In this chapter, Section 15.2 goes through the major visualization tools, whereas Section 15.3 reviews the two animations.

15.2 Visualization Tools

Overview

In this research, an assortment of astronomy-related tools has been developed for visualization and experimental purposes. Out of the wide-ranging set of tools we developed in the course of our research, here we list what we consider to be the seven major tools:
15. Results

15. Results

earthday – Visualizing the Earth environment
starview – Visualizing the interstellar environment, i.e., stars
galview – Visualizing the intergalactic environment, i.e., galaxies
specview – Multi-wavelength view of the sky
skyalign – Alignment tool for accurately placing images and 3D models in the night sky
distEdTool – Editing tool for data-driven gas cloud construction
power10time – CAVE-based visualization of the powers-of-ten scales of the Cosmos

Except power10time, which is written mostly in Performer [145], all the above listed tools use solely OpenGL for rendering. Therefore, we can execute these programs on a variety of platforms. Available platforms include IRIX, SunOS, Linux, and Windows. In addition to these seven major tools, we have meteorsim (meteor shower simulator), prtview (particle model viewer), objviewer (obj model viewer), and moviePET (movie path editing tool).

Tool: Earthday

The first tool presented in this section is called earthday. Basically, it presents the spatial environment surrounding the Earth, including about a thousand Earth-orbiting satellites and planetary trajectories in the sky. Snapshots taken from this tool have been shown extensively in Chapter 8, which presents our methods for modeling and rendering the Solar system.

The earthday program includes the following features:

- The Earth with sunlight shading and atmospheric effects
- Earth-orbiting satellites
Figure 15.1: Screenshots of `earthday`.
15. Results

- Accurate simulation of Solar Eclipses, including penumbra and umbra outlines
- Landmarks on Earth: major cities, time zones, the international date line, latitude and longitude grid, etc.
- The Sun, the Moon, the Zodiacs, and the Ecliptic
- Planetary trajectories and simulation of retrograde motions, with pedagogical inset to visualize cause of apparent retrograde paths
- Background with stars, Milky Way, constellation lines and boundaries
- Time control

Screenshots from earthday can be found in Figure 15.1 and throughout Chapter 8.

**Tool: starview**

The next tool, starview, brings us from the Solar system to interstellar space; we can experiment with star modeling and rendering with this tool, and can visualize numerous stellar characteristics. The current version of starview has the following features:

- Rendering of stars, typically from the Hipparcos catalogue
- Selection of individual stars and display of star properties
- Equatorial and Galactic grids
- Constellation lines, labels, and boundaries
- Velocity trails (motion vectors) of stars
- Visualization of position and velocity errors of stars
15. Results

- Orrery of the Solar system at $10^{18}$ meters
- Orbits of known extra-solar planets
- Visual cues: powercubes (see Figure 14.10) and minimum spanning tree (see Figure 14.11)
- Time control

Screenshots from starview can be found in Figure 15.2 and throughout Chapter 9.

**Tool: galview**

The program galview stands for galaxy view, which depicts the space of galaxies. This program starts at the Milky Way Galaxy, and brings us from the Milky Way to the whole catalog of galaxies. The current version of galview has the following features:

- Rendering of the Milky Way Galaxy using a particle model
- Rendering of galaxies including using images or iconic models
- Selection of individual galaxies and display of galaxy properties
- Equatorial, Galactic, and SuperGalactic grids
- CMBR (Cosmic Microwave Background Radiation) as the background image
- Visual cue for spatial scale: powercubes

Screenshots from galview can be found in Figure 15.3. Most rendering results shown in Chapters 10 and 9 are generated by galview.
Figure 15.2: Screenshots of starview.
Figure 15.3: Screenshots of \textit{galview}.
15. Results

**Tool: specview**

Unlike the previous three programs, in which users can fly through three-dimensional astronomical environments, *specview* (spectral viewer) focuses on the visualization of the multi-wavelength sky based on the Earth’s perspective, or employing various cartographic topologies like the Aitoff or rectangular projections. Related images and descriptions of the multi-wavelength sky can be found in Figure 10.5 of Chapter 10 and Section A.3 of the Appendices, respectively. The *specview* program has the following features:

- Rendering of the multi-wavelength sky (the electromagnetic spectrum of light)
- On-screen spectral slider for interactive selection of wavelength
- Different visualization modes of the sky: the spherical mode, the Aitoff mode, and the planar mode (rectangular), and smooth transitions between modes
- Galactic grid with labels

Screenshots from *specview* can be found in Figure 15.4.

**Tool: skyalign**

Similar to the full-sky view (spherical mode) in *specview*, the *skyalign* program fixes the viewpoint at the world center and renders the sky around it. The motivation for *skyalign* is that most images and photographs we have obtained from the astronomy community do not come labeled with accurate sky coordinates; thus, whenever we want to put an image like this into an animation relative to other objects in the sky, we need to know roughly where the image is in the sky. A related problem is positioning 3D gas cloud models in the sky. The *skyalign* tool provides an interface for accomplishing accurate alignment including the following features:
Figure 15.4: Screenshots of *specview*.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
15. Results

- Common 3D model format (obj format) as input and output
- Interactive adjustment against the sky
- Display of stars, constellation lines and boundaries in the background for cross-reference
- Display of multi-wavelength Milky Way background for multi-wavelength alignment

Screenshots from skyalign can be found in Figure 15.5: The first two sub-columns depict the alignment results of images, Barnard’s Loop and Vela, whereas the last sub-column depicts the alignment result of the 3D gas cloud model Ophiuchus-b.

**Tool: distEdTool**

Reliable modeling of three-dimensional (3D) gas cloud structures in the interstellar environment is a challenging task in the analysis of astronomy, and is required for understanding the relationships among multispectral observations of the interstellar matter (ISM) in the local galactic neighborhood of the Sun. The idea behind this tool is that after we obtain distances to stars based on the parallax method (see Chapter 9), we can compare the magnitudes of stars based on the observations and the theoretical values from their distances. The difference, or the drop in magnitude, suggests the presence of an interstellar cloud or dust between the Earth and those nearby stars. Astronomers can make use of this information plus their experience with the interstellar environment to construct a three-dimensional model for interstellar gas clouds, or the empty spaces between clouds such as the “Local Bubble.”

The *distEdTool* system software (the Distance Editing Tool, or DET for short) provides a program interface for the construction and verification of models representing spectrally
Figure 15.5: Screenshots of *skyalign*. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
absorbing and emitting gas clouds lying between the Earth and nearby stars; an expert scientist can interactively tune distances and local variables such as cloud volume densities for each line of sight and produce simulations and virtual explorations of the astronomical environments. Details regarding this tool can be found in our book chapter [82]. Figure 15.6 shows some screenshots of the tool.

**Tool: power10time**

Unlike all the above desktop-based tools, *power10time* is a CAVE-based application, which was written using Performer in addition to OpenGL. It is a multi-process application exploiting the multi-projector immersive rendering environment in the CAVE. In our development history, this program was one of the first tools to experiment with large-scale visualization before the development of all the other tools. The current version thus lacks the realistic rendering methods developed later on for stars, galaxies, and so on. Despite this, it includes all necessary functionality for large-scale rendering and navigation, including the large-scale constrained navigation, the environment caching method, and the cycling mechanism, as well as the large-scale animation management. Screenshots from *power10time* can be found in Figure 15.7; most scenes shown in our animation *Cosmic Clock* [80] were generated using this program. Hence, *power10time* can be regarded as a live demonstration of the animation. Merging the methods of the more recent and more sophisticated tools into a new and improved *power10time* CAVE application is a possible goal for continuing research effort.
Figure 15.6: Screenshots of DistEdTool (Distance Editing Tool).
Figure 15.7: Screenshots of power10time.
15.3 Animations

Overview

Two computer animations based our modeling and rendering techniques and tools were created in the course this research work:

- **Cosmic Clock** — Visualizing cosmological time using the Lightcone
- **Solar Journey** — Visualizing the interstellar environment around the Sun

**Animation: Cosmic Clock**

*Cosmic Clock* [80] is the first computer animation developed in this research work. This 3:35 minute animation contains a visualization of the entire Universe from three different points of view: the time spectrum of observable photon radiation arriving at the Earth, the constant-time shells of light sources represented in “comoving coordinates” (as though the Universe had always been the size it is today), and in “physical coordinates” (which incorporate the Hubble expansion since the “Great Flash”, when the Universe was about 380,000 years old).

Detailed explanations of these three different points of view can be found in Chapter 12, in which the Lightcone was introduced as a visualization method. This film was one of a select few chosen for showing at the Siggraph 2000 Electronic Theater (see Figure 15.8) in July 2000 [80]; snapshots taken from this animation can be found in Figure 15.9 and throughout Chapter 12.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
15. Results

On Earth, we see the night sky as a single image filled with points of light. Yet each beam of light that reaches our eyes has a different history. This animation depicts the fantastic story of these traveling light rays as they reveal snapshots of the cosmos reaching back toward the beginning of time itself.

Andrew J. Hanson
Indiana University
hansso@indiana.edu

Director
Andrew J. Hanson

Producer
Andrew J. Hanson

Animator
Phillip C.W. Hu

Post production
Daniel Rust

Annotation of Cosmic Clock

Cover of the Electronic Theater Proceedings

Figure 15.8: Annotation of the Cosmic Clock animation in the Electronic Theater Proceedings (top) and cover of the Proceedings (bottom) (Siggraph 2000).
Figure 15.9: Snapshots taken from *Cosmic Clock*.
15. Results

Animation: Solar Journey

Our Solar system is like a spacecraft traveling across the Milky Way Galaxy at a speed of about 220 km/s. The Solar Journey animation depicts the local view of interstellar objects around our Solar system and reveals to us the interstellar space through which our Sun travels.

The current version of this animation starts at the Solar system scale, looking at the Sun, and then moves out to see the whole Milky Way Galaxy. After reviewing the 1987a-supernova explosion in the LMC, we return to our local interstellar environment and fly around the supernova remnant “Loop I,” through the edge of which our Sun is traveling (at a viewing scale of about 10^18 meters). Then, we fly closer to the rim of the Solar system and review the Heliopause structure (at a viewing scale of about 10^16 meters), which is generated by the force of the Solar wind against the interstellar medium. Indeed, this Heliopause structure acts like a shield pushing away the interstellar medium originating from Loop I or other dust clouds so that we can move across interstellar space safely. An extended version of this animation was selected for exhibition at the planned 2004 opening of the Beijing Digital Planetarium; the extended version will include a visit to the black hole at the Milky Way center as well as additional multi-spectral visualizations. Snapshots taken from the current animation are shown in Figure 15.10.

15.4 Conclusion

In conclusion, we have developed new and advanced methods and techniques for large-scale visualization. These are realized via the implementation of a large assortment of scientific and pedagogical tools as well as two animations. These artifacts are being used by astronomers and digital planetarium developers to explore large-scale astrophysical environments as well as to help the general public gain knowledge about the latest advances.
Figure 15.10: Snapshots taken from *Solar Journey* (Note: since this computer animation is to be shown in Beijing, we have used Chinese characters for the annotations).
in astronomy.

The current development system is packaged in a well-defined directory structure:

- We maintain each individual library API (application programming interface) independently: stars, galaxies, satellites, 3D "obj" models, particle system, ephemeris, etc. The source code of each API is well-packaged (cross-platform) and altogether, these APIs form the development kernel for the various tools and animations we described in this chapter.

- We also maintain a "tools" directory, which includes all the tools developed in this research project. Each tool utilizes the corresponding library APIs for data retrieval, rendering, etc.

- Furthermore, we have another set of directories for animations. Similar to "tools," the animation source programs also make use of the library APIs for common data retrieval and rendering.

By organizing the development environment in this way, we can enforce better code sharing in the library APIs among the animations and the tools as well as a cleaner source code hierarchy, helping to avoid unnecessary debugging.

In the future, we would like to see more tools and animations developed based on this platform. One idea is to create a tool to bridge the gap between galview and starview; it is proposed to be called milkywayview for navigation inside the Milky Way Galaxy and for locating globular clusters, nebulae, and star associations in the Milky Way. Another future task is to polish our preliminary work for the CAVE, the power10time; based on the existing development environment, we could incorporate our advanced modeling and rendering techniques in the power10time application as well. Furthermore, most of the listed tools (e.g., earthday, starview, galview, etc.) can be used not only on the desktop,
but also can be extended to kiosk applications for demonstration and educational purposes in museums and planetariums.
16

Conclusion and Future work

16.1 Conclusion

This dissertation contains technical advances of value to large spatiotemporal simulation and visualization environments, of which virtual astronomy is the most natural testing ground. It summarizes our four-year program of research on large-scale astronomical visualization, including adapting the graphics rendering pipeline for large-scale rendering, manipulating a variety of raw astronomical data into pictures visualizable on the computer screen, and making visualizations, navigations, and animations for a gigantic virtual space-time arena.

Considered altogether, we addressed four major areas in this research work. Part I of this dissertation made up the general framework for large-scale visualization based on a number of strategies designated for speeding up and adapting the capabilities of the graphics rendering pipeline: power homogeneous coordinates (PHC) (Chapter 3), generalized modelview and projection transformations (Chapters 4 and 5), the depth rescaling method (Chapters 4 and 5), the environment caching criterion (Chapter 6), and the object disappearance criterion (Chapter 7). Figure 16.1 portrays the visualization framework.
Part II proposed a collection of modeling and rendering methodologies for a diversity of astronomical bodies, ranging from the Solar system (the Earth, satellites, the Sun, and the planets) (Chapter 8), the stars (Chapter 9), the Milky Way Galaxy (Chapter 10), up to the outer space galaxies (Chapter 11), billions of light years away from us. This indeed stretched the graphics rendering requirements far beyond the daily-life scale and demonstrated the capability of the large-scale visualization strategies in Part I by successfully simulating the entire scale-range of the Universe using interactive computer graphics technology.

Furthermore, Part III attempted to visualize cosmological time and to deal with time-related problems in the dynamic Universe. The Lightcone representation (Chapter 12) was introduced to reveal the whole space-time Universe at a single glance, so that we can better visualize the effect of the Hubble expansion by scaling corresponding time slices in the Universe. The visualization was later made into the “Cosmic Clock” animation. In addition, various time-related visualization issues concerning different motion speeds and

Figure 16.1: Large-scale visualization framework with “power homogeneous coordinates” as the basis.
dynamic interactions were analyzed in this part (Chapter 13).

Finally, Part IV of this dissertation presented the four different navigation models to address the problems of navigation in large-scale space; the space-scaling model, the path-based model, and the constrained navigation model (Chapter 14) were proposed in this research work to deal with the problem. Alongside this, some visualization strategies were introduced to provide visual cues for the navigation process.

Assembling all these components, this research work synthesized a unified framework for large-scale visualization. Typically, this was put into practice in the context of astronomy, in which we have huge scale in both spatial and temporal domains. In practice, these research achievements were realized as a collection of visualization tools as well as two computer animations (Chapter 15): A number of the listed tools have been put into practice by several astronomers; some of the programs will soon be distributed to the astronomy community for further exploration. The animation “Cosmic clock” has been shown not only at the Siggraph 2000 conference, but also in a Japanese TV program. In addition, the animation “Solar Journey” will be shown in the Beijing planetarium under construction at the time of this writing.

16.2 Development Highlights

This section highlights the ideas and techniques developed in this work (see the next subsection for the related publications). “ARTT” denotes developments that have been filed with the Indiana University Advanced Research and Technology Institute as intellectual property disclosures.
Part I
- Power homogeneous coordinates published [83, 81]
- Generalizing the Modelview transformation by publishable & ARTI
  attaching a spatial scale, s_{mv}
- Generalizing the Projection transformation by publishable & ARTI
  depth rescaling method
- Environment caching criterion published [83]
- Object disappearance criterion published [83]

Part II
- Visualizing satellites around the Earth published [68]
- Visualizing the Heliopause structure published [67]
- Photo-realistic rendering of stars publishable
- VP-tree and M-tree hierarchy for star rendering publishable
- Complete star rendering method exported to SGI
- Constrained texture synthesis for star removal publishable &
  in Milky Way panoramic image exported to SGI
- Complete galaxy rendering method exported to SGI

Part III
- Lightcone visualization and time slicing method in “Cosmic Clock” published [80, 84]
- Time-related visualization strategies published [81]

Part IV
- Extended constrained navigation for large-scale environments published [83]
16. Conclusion and Future work

Part V

○ Cosmic Clock published [80]
○ Solar Journey Planetarium Show
○ Interface design of DET published [82]
○ Dome / CAVE adaptation exported to SGI
○ Tools like “starview” and “earthday” planned booth demonstrations
○ Tools like “skyalign” planned open source

Related References


[67] Priscilla Frisch, Andrew J. Hanson, Chi-Wing Fu (animator), and Eric A. Wernert (animator). The Sun’s Heliosphere & Heliopause. Astronomy Picture of the Day, 2002 June 24.

[84] Andrew J. Hanson, Chi-Wing Fu, and Eric A. Wernert. Visualizing cosmological time. In Frits H. Post, Gregory M. Nielson, and Georges-Pierre Bonneau, editors,
16. Conclusion and Future work


16.3 Future work

We see two main directions for future work. The first is educational outreach. Using our existing work, we can communicate to the public and the astronomy community through media like booth demonstrations, planetarium shows, and visualization tools. This not only facilitates scientific visualization, but also serves an educational purpose. Another direction is to apply the large-scale visualization framework to other scientific domains. As mentioned earlier in this dissertation, the large-scale visualization framework introduced in Part I is independent of the data context. In principle, we can employ this visualization framework for other visualization applications such as physical data from the Planck-length scale up to molecular and human scales.
Astronomy-Related

A.1 Units

This section includes units for distances, wavelength conversion, and angles.

Table A.1: Units for Distances (www.unc.edu/~rowlett/units).

<table>
<thead>
<tr>
<th>Unit</th>
<th>Convert to meters</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>meters (m)</td>
<td>1 m</td>
<td></td>
</tr>
<tr>
<td>kilometers (km)</td>
<td>1,000 m</td>
<td>0.621371 miles</td>
</tr>
<tr>
<td>miles</td>
<td>1,609.344 m</td>
<td></td>
</tr>
<tr>
<td>astronomical unit (A.U.)</td>
<td>$1.495979 \times 10^{11}$ m</td>
<td></td>
</tr>
<tr>
<td>light-year (ly)</td>
<td>$9.460528 \times 10^{15}$ m</td>
<td>0.306595 parsec</td>
</tr>
<tr>
<td>parsec</td>
<td>$3.085678 \times 10^{16}$ m</td>
<td>3.261631 ly</td>
</tr>
</tbody>
</table>

Table A.2: Units for Wavelength, Frequency, Energy, and Temperature.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Convert from meters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wavelength</td>
<td>X meters</td>
</tr>
<tr>
<td>Frequency</td>
<td>X meters = $2.99792 \times 10^8$ / X Hz (sec$^{-1}$)</td>
</tr>
<tr>
<td>Temperature</td>
<td>X meters = 0.0143878 / X K (Kelvin)</td>
</tr>
<tr>
<td>Energy</td>
<td>X meters = $1.239817567 \times 10^{-6}$ / X eV (electron volt)</td>
</tr>
</tbody>
</table>
Examples (Wavelength):
- Gamma rays: \(10^{-12} \text{ to } 10^{-10} \text{ m}\)
- X-rays: \(10^{-10} \text{ to } 10^{-6} \text{ m}\)
- Ultraviolet: \(10^{-6} \text{ to } 10^{-4} \text{ m}\)
- Visible light: (about \(\text{10}^{-5} \text{ m}\))
- Infrared: \(10^{-8} \text{ to } 10^{-4} \text{ m}\)
- Microwaves: \(10^{-4} \text{ to } 1 \text{ m}\)
- Radio: \(1 \text{ to } 10^4 \text{ m}\)

Table A.3: Units for Angles.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 hour</td>
<td>15 degrees</td>
</tr>
<tr>
<td>1 degree</td>
<td>1 degree</td>
</tr>
<tr>
<td>1 radian</td>
<td>(\frac{180}{\pi} \text{ degrees})</td>
</tr>
<tr>
<td>1 arc minute</td>
<td>(\frac{1}{60} \text{ degrees})</td>
</tr>
<tr>
<td>1 arc second</td>
<td>(\frac{1}{3,600} \text{ degrees})</td>
</tr>
<tr>
<td>1 milli arc second</td>
<td>(\frac{1}{3,600,000} \text{ degrees})</td>
</tr>
</tbody>
</table>

A.2 Basic Statistics

This section includes statistics for the Sun, the Moon, the Nine planets of the Solar system, asteroids, comets, and meteor showers, etc.
Table A.4: Statistics of the Sun from [1, 102].

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass</td>
<td>$1.989 \times 10^{33}$ g (332,946 Earth masses)</td>
</tr>
<tr>
<td>Radius</td>
<td>696,510 km (109 Earth radii)</td>
</tr>
<tr>
<td>Volume</td>
<td>$1.412 \times 10^{27}$ cm$^3$ (1.3 million Earths)</td>
</tr>
<tr>
<td>Density (center)</td>
<td>151.3 g/cm$^3$</td>
</tr>
<tr>
<td>Density (mean)</td>
<td>1.409 g/cm$^3$</td>
</tr>
<tr>
<td>Density (photosphere)</td>
<td>2.334 \times 10^{11} bar</td>
</tr>
<tr>
<td>Pressure (center)</td>
<td>2.334 \times 10^{11} bar</td>
</tr>
<tr>
<td>Pressure (photosphere)</td>
<td>0.0001 bar</td>
</tr>
<tr>
<td>Pressure (corona)</td>
<td>15,657,000 K</td>
</tr>
<tr>
<td>Pressure (photosphere)</td>
<td>5,780 K</td>
</tr>
<tr>
<td>Pressure (corona)</td>
<td>2,000,000 to 3,000,000 K</td>
</tr>
<tr>
<td>Temperature (center)</td>
<td>15,557,000 K</td>
</tr>
<tr>
<td>Temperature (photosphere)</td>
<td>5,780 K</td>
</tr>
<tr>
<td>Temperature (corona)</td>
<td>2,000,000 to 3,000,000 K</td>
</tr>
<tr>
<td>Luminosity</td>
<td>$3.854 \times 10^{33}$ erg/s</td>
</tr>
<tr>
<td>Solar constant</td>
<td>$1.368 \times 10^6$ erg/s cm$^{-2} = 1,368$ W/m$^2$</td>
</tr>
<tr>
<td>Magnitude (absolute bolometric)</td>
<td>4.72</td>
</tr>
<tr>
<td>Magnitude (absolute visual)</td>
<td>4.79</td>
</tr>
<tr>
<td>Spectral type</td>
<td>G2V</td>
</tr>
<tr>
<td>Color indices</td>
<td>(B-V) 0.62, (U-B) 0.10</td>
</tr>
<tr>
<td>Constituents</td>
<td>Hydrogen 92.1 percent, Helium 7.8 percent, All others 0.1 percent</td>
</tr>
<tr>
<td>Escape velocity</td>
<td>620 km/s</td>
</tr>
</tbody>
</table>

Table A.5: Statistics of the Moon from [1].

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lunar mass(g)</td>
<td>$7.349 \times 10^{25}$</td>
</tr>
<tr>
<td>Equatorial radius(km)</td>
<td>1,738</td>
</tr>
<tr>
<td>Ellipticity</td>
<td>0.002</td>
</tr>
<tr>
<td>Gravity (m/s$^2$)</td>
<td>1.62</td>
</tr>
<tr>
<td>Sidereal period</td>
<td>27.3217 d</td>
</tr>
<tr>
<td>Moon-Earth Distance</td>
<td>384,400 km</td>
</tr>
<tr>
<td>Escape Vel. (km/s)</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
### Table A.6: Statistics of the Nine Planets from [2].

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Mercury</th>
<th>Venus</th>
<th>Earth</th>
<th>Mars</th>
<th>Jupiter</th>
<th>Saturn</th>
<th>Uranus</th>
<th>Neptune</th>
<th>Pluto</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dist. Sun (10^6 km)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max.</td>
<td>69.7</td>
<td>109.0</td>
<td>132</td>
<td>249</td>
<td>815.7</td>
<td>1.507</td>
<td>3.004</td>
<td>4.537</td>
<td>7.375</td>
</tr>
<tr>
<td>mean</td>
<td>57.9</td>
<td>108.2</td>
<td>149.6</td>
<td>227.9</td>
<td>778</td>
<td>1.427</td>
<td>2.870</td>
<td>4.497</td>
<td>5.900</td>
</tr>
<tr>
<td>min.</td>
<td>45.9</td>
<td>107.4</td>
<td>147</td>
<td>208.7</td>
<td>740.9</td>
<td>1.347</td>
<td>2.735</td>
<td>4.456</td>
<td>4.425</td>
</tr>
<tr>
<td>Orbital period</td>
<td>88.97d</td>
<td>224.7d</td>
<td>365.3d</td>
<td>687.0d</td>
<td>11.86y</td>
<td>29.46y</td>
<td>84.01y</td>
<td>164.8y</td>
<td>247.7y</td>
</tr>
<tr>
<td>Synodic period</td>
<td>115.9d</td>
<td>583.9d</td>
<td>-</td>
<td>779.9d</td>
<td>398.9d</td>
<td>378.1d</td>
<td>369.7d</td>
<td>367.5d</td>
<td>366.7d</td>
</tr>
<tr>
<td>Rotation period (eq.)</td>
<td>58.64d</td>
<td>243.16d</td>
<td>23h56m</td>
<td>24h37m</td>
<td>9h50m</td>
<td>10h14m</td>
<td>17h14m</td>
<td>16h7m</td>
<td>6.375d</td>
</tr>
<tr>
<td>Orbital eccentricity</td>
<td>0.206</td>
<td>0.007</td>
<td>0.017</td>
<td>0.093</td>
<td>0.048</td>
<td>0.056</td>
<td>0.047</td>
<td>0.009</td>
<td>0.248</td>
</tr>
<tr>
<td>Orbital incln. (°)</td>
<td>7.0</td>
<td>3.4</td>
<td>0.0</td>
<td>1.8</td>
<td>1.3</td>
<td>2.5</td>
<td>0.8</td>
<td>1.8</td>
<td>17.15</td>
</tr>
<tr>
<td>Axial incln. (°)</td>
<td>2</td>
<td>178</td>
<td>23.44</td>
<td>24.0</td>
<td>3.1</td>
<td>26.7</td>
<td>98</td>
<td>28.8</td>
<td>122.5</td>
</tr>
<tr>
<td>Escape vel. (km/s)</td>
<td>4.25</td>
<td>10.36</td>
<td>11.18</td>
<td>5.03</td>
<td>60.22</td>
<td>32.26</td>
<td>22.5</td>
<td>23.9</td>
<td>1.18</td>
</tr>
<tr>
<td>Mass (Earth=1)</td>
<td>0.055</td>
<td>0.815</td>
<td>1</td>
<td>0.11</td>
<td>317.9</td>
<td>95.2</td>
<td>14.6</td>
<td>17.2</td>
<td>0.002</td>
</tr>
<tr>
<td>Volume (Earth=1)</td>
<td>0.056</td>
<td>0.86</td>
<td>1</td>
<td>0.15</td>
<td>1319</td>
<td>744</td>
<td>67</td>
<td>57</td>
<td>0.01</td>
</tr>
<tr>
<td>Avg. Density (water=1)</td>
<td>5.44</td>
<td>5.25</td>
<td>5.52</td>
<td>3.94</td>
<td>1.33</td>
<td>0.71</td>
<td>1.27</td>
<td>2.06</td>
<td>2.03</td>
</tr>
<tr>
<td>Gravity (Earth=1)</td>
<td>0.38</td>
<td>0.90</td>
<td>1</td>
<td>0.38</td>
<td>2.64</td>
<td>1.16</td>
<td>1.17</td>
<td>1.2</td>
<td>0.06</td>
</tr>
<tr>
<td>Surface temp. (°C)</td>
<td>+427</td>
<td>+480</td>
<td>+22</td>
<td>-23</td>
<td>-150</td>
<td>-180</td>
<td>-214</td>
<td>-220</td>
<td>-230</td>
</tr>
<tr>
<td>Equatorial diam. (km)</td>
<td>4,878</td>
<td>12,104</td>
<td>12,756</td>
<td>6,794</td>
<td>143,884</td>
<td>120,536</td>
<td>51,118</td>
<td>50,538</td>
<td>2,324</td>
</tr>
<tr>
<td>Maximum magnitude</td>
<td>-1.9</td>
<td>-4.4</td>
<td>-</td>
<td>-2.8</td>
<td>-2.6</td>
<td>-0.3</td>
<td>+5.6</td>
<td>-7.7</td>
<td>+14</td>
</tr>
<tr>
<td>Satellites found(2001)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>28</td>
<td>30</td>
<td>21 (24)*</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

**Note:**
- Earth's mass = 5.974 x 10^24 kg, Earth's radius = 6,378 km, Earth's gravity = 9.78 m/s²
### Table A.7: Statistics of selected asteroids from \([1, 102]\).

<table>
<thead>
<tr>
<th>Number (ID)</th>
<th>Name</th>
<th>Year Found</th>
<th>Discoverer</th>
<th>Diameter (km)</th>
<th>Mean dist. from Sun (A.U.)</th>
<th>Orbital period (years)</th>
<th>Orbital eccen.</th>
<th>Orbital incl. (°)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ceres</td>
<td>1801</td>
<td>G. Piazzi</td>
<td>933</td>
<td>2.768</td>
<td>4.60</td>
<td>0.08</td>
<td>10.6</td>
</tr>
<tr>
<td>2</td>
<td>Pallas</td>
<td>1802</td>
<td>Olbers</td>
<td>583</td>
<td>2.77</td>
<td>4.6</td>
<td>0.23</td>
<td>34.8</td>
</tr>
<tr>
<td>3</td>
<td>Juno</td>
<td>1804</td>
<td>Harding</td>
<td>249</td>
<td>2.67</td>
<td>4.4</td>
<td>0.26</td>
<td>13.0</td>
</tr>
<tr>
<td>4</td>
<td>Vesta</td>
<td>1807</td>
<td>Olbers</td>
<td>555</td>
<td>2.36</td>
<td>3.6</td>
<td>0.09</td>
<td>7.1</td>
</tr>
<tr>
<td>5</td>
<td>Astraea</td>
<td>1845</td>
<td>Hencke</td>
<td>116</td>
<td>2.58</td>
<td>4.1</td>
<td>0.19</td>
<td>5.3</td>
</tr>
<tr>
<td>6</td>
<td>Hebe</td>
<td>1847</td>
<td>Hencke</td>
<td>206</td>
<td>2.42</td>
<td>3.8</td>
<td>0.20</td>
<td>14.8</td>
</tr>
<tr>
<td>7</td>
<td>Iris</td>
<td>1847</td>
<td>Hind</td>
<td>222</td>
<td>2.39</td>
<td>3.7</td>
<td>0.23</td>
<td>5.5</td>
</tr>
<tr>
<td>8</td>
<td>Flora</td>
<td>1847</td>
<td>Hind</td>
<td>160</td>
<td>2.20</td>
<td>3.3</td>
<td>0.16</td>
<td>5.9</td>
</tr>
<tr>
<td>9</td>
<td>Metis</td>
<td>1848</td>
<td>Graham</td>
<td>168</td>
<td>2.39</td>
<td>3.7</td>
<td>0.12</td>
<td>5.6</td>
</tr>
<tr>
<td>10</td>
<td>Hygiea</td>
<td>1849</td>
<td>DeGasperis</td>
<td>443</td>
<td>3.14</td>
<td>5.6</td>
<td>0.12</td>
<td>3.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number (ID)</th>
<th>Name</th>
<th>Year Found</th>
<th>Discoverer</th>
<th>Diameter (km)</th>
<th>Mean dist. from Sun (A.U.)</th>
<th>Orbital period (years)</th>
<th>Orbital eccen.</th>
<th>Orbital incl. (°)</th>
</tr>
</thead>
<tbody>
<tr>
<td>253</td>
<td>Mathilde</td>
<td>1885</td>
<td>J. Palisa</td>
<td>57 × 50</td>
<td>2.646</td>
<td>4.30</td>
<td>0.27</td>
<td>6.7</td>
</tr>
<tr>
<td>433</td>
<td>Eros</td>
<td>1893</td>
<td>G. Witt, A. Charlois</td>
<td>35 × 13</td>
<td>1.458</td>
<td>1.76</td>
<td>0.22</td>
<td>10.8</td>
</tr>
<tr>
<td>243</td>
<td>Ida</td>
<td>1905</td>
<td>J. Palisa</td>
<td>56 × 15</td>
<td>2.862</td>
<td>4.8</td>
<td>0.04</td>
<td>1.1</td>
</tr>
<tr>
<td>624</td>
<td>Hektor</td>
<td>1907</td>
<td>A. Kopff</td>
<td>225</td>
<td>5.203</td>
<td>11.86</td>
<td>0.02</td>
<td>18.2</td>
</tr>
<tr>
<td>951</td>
<td>Gaspra</td>
<td>1916</td>
<td>G. Neujmin</td>
<td>18 × 9</td>
<td>2.209</td>
<td>3.28</td>
<td>0.17</td>
<td>4.1</td>
</tr>
<tr>
<td>1221</td>
<td>Amor</td>
<td>1932</td>
<td>E. Delporte</td>
<td>(1)</td>
<td>1.919</td>
<td>2.66</td>
<td>0.84</td>
<td>11.9</td>
</tr>
<tr>
<td>1862</td>
<td>Apollo</td>
<td>1932</td>
<td>K. Reinmuth</td>
<td>1.5</td>
<td>1.471</td>
<td>1.78</td>
<td>0.56</td>
<td>6.4</td>
</tr>
<tr>
<td>2101</td>
<td>Adonis</td>
<td>1936</td>
<td>E. Delporte</td>
<td>(1)</td>
<td>1.874</td>
<td>2.57</td>
<td>0.76</td>
<td>1.4</td>
</tr>
<tr>
<td>1937 UB</td>
<td>Hermes</td>
<td>1937</td>
<td>K. Reinmuth</td>
<td>(1)</td>
<td>(1.644)</td>
<td>(2.11)</td>
<td>(0.62)</td>
<td>6.1</td>
</tr>
<tr>
<td>2062</td>
<td>Aten</td>
<td>1976</td>
<td>E. Helin</td>
<td>0.9</td>
<td>0.967</td>
<td>0.95</td>
<td>0.18</td>
<td>18.9</td>
</tr>
</tbody>
</table>

### Table A.8: Statistics of some periodic comets from \([2]\).

<table>
<thead>
<tr>
<th>Name</th>
<th>Period (yrs)</th>
<th>Dist. from Sun (A.U.) min</th>
<th>Dist. from Sun (A.U.) max</th>
<th>Orbital inclination (°)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encke</td>
<td>3.3</td>
<td>0.34</td>
<td>4.09</td>
<td>12.0</td>
</tr>
<tr>
<td>Grigg-Skjellerup</td>
<td>5.1</td>
<td>1.00</td>
<td>4.94</td>
<td>21.1</td>
</tr>
<tr>
<td>Tempel 2</td>
<td>5.3</td>
<td>1.36</td>
<td>4.68</td>
<td>12.5</td>
</tr>
<tr>
<td>D'Arrest</td>
<td>6.2</td>
<td>1.17</td>
<td>5.61</td>
<td>16.7</td>
</tr>
<tr>
<td>Pons-Winnecke</td>
<td>6.3</td>
<td>1.25</td>
<td>5.61</td>
<td>22.3</td>
</tr>
<tr>
<td>Giacobini-Zinner</td>
<td>6.5</td>
<td>0.99</td>
<td>5.98</td>
<td>31.7</td>
</tr>
<tr>
<td>Finlay</td>
<td>6.9</td>
<td>1.10</td>
<td>6.19</td>
<td>2.6</td>
</tr>
<tr>
<td>Faye</td>
<td>7.4</td>
<td>1.62</td>
<td>5.98</td>
<td>9.1</td>
</tr>
<tr>
<td>Tuttle</td>
<td>13.8</td>
<td>1.02</td>
<td>10.46</td>
<td>54.4</td>
</tr>
<tr>
<td>Halley</td>
<td>76.1</td>
<td>0.59</td>
<td>35.33</td>
<td>162.2</td>
</tr>
<tr>
<td>Swift-Tuttle</td>
<td>133</td>
<td>0.96</td>
<td>51.7</td>
<td>113.4</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
### Table A.9: Statistics of selected meteor showers from [1] and [2].

<table>
<thead>
<tr>
<th>Name</th>
<th>Date of Return</th>
<th>Maximum</th>
<th>Radiant</th>
<th>Associated Comets</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quadrantids</td>
<td>1-6 Jan</td>
<td>4 Jan</td>
<td>Boo</td>
<td></td>
<td>Quite fast, blue</td>
</tr>
<tr>
<td>Corona Australids</td>
<td>14-18 Mar</td>
<td>16 Mar</td>
<td>CrA</td>
<td></td>
<td>Fast, brilliant</td>
</tr>
<tr>
<td>Lyrids (April)</td>
<td>19-24 Apr</td>
<td>21 Apr</td>
<td>Lyr</td>
<td></td>
<td>Fast, persistent</td>
</tr>
<tr>
<td>η-Aquarids</td>
<td>1-8 May</td>
<td>5 May</td>
<td>Aqr</td>
<td></td>
<td>Blue</td>
</tr>
<tr>
<td>Lyrids (June)</td>
<td>10-21 Jun</td>
<td>15 Jun</td>
<td>Lyr</td>
<td></td>
<td>Yellow, very slow</td>
</tr>
<tr>
<td>Ophiuchids</td>
<td>17-26 Jun</td>
<td>20 Jan</td>
<td>Oph</td>
<td></td>
<td>Slow, long paths</td>
</tr>
<tr>
<td>Capricornids</td>
<td>10 Jun - 15 Aug</td>
<td>25 Jul</td>
<td>Cap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>δ-Aquarids</td>
<td>15 Jul - 15 Aug</td>
<td>28 Jul</td>
<td>Aqr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pisces Australids</td>
<td>15 Jul - 20 Aug</td>
<td>30 Jul</td>
<td>PsA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capricornids</td>
<td>15 Jul - 25 Aug</td>
<td>1 Aug</td>
<td>Cap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perseids</td>
<td>25 Jul - 18 Aug</td>
<td>12 Aug</td>
<td>Per</td>
<td></td>
<td>Bright, exploding</td>
</tr>
<tr>
<td>Cygnids</td>
<td>18-22 Aug</td>
<td>20 Aug</td>
<td>Cyg</td>
<td></td>
<td>Fast, persistent</td>
</tr>
<tr>
<td>Orionids</td>
<td>16-26 Oct</td>
<td>21 Oct</td>
<td>Ori</td>
<td>IP/Halley</td>
<td>Slow, brilliant</td>
</tr>
<tr>
<td>Taurids</td>
<td>10 Oct - 30 Nov</td>
<td>1 Nov</td>
<td>Taur</td>
<td>2P/Encke</td>
<td></td>
</tr>
<tr>
<td>Leonids</td>
<td>15-19 Nov</td>
<td>17 Nov</td>
<td>Leo</td>
<td>55P/Tempel-Tuttle</td>
<td>Fast, persistent</td>
</tr>
<tr>
<td>Phoenicids</td>
<td>4 Dec</td>
<td>14 Dec</td>
<td>Phe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gemmids</td>
<td>7-15 Dec</td>
<td>14 Dec</td>
<td>Gem</td>
<td>3200 Phaethon</td>
<td>White</td>
</tr>
<tr>
<td>Ursids</td>
<td>17-24 Dec</td>
<td>22 Dec</td>
<td>UMi</td>
<td>8P/Tuttle</td>
<td></td>
</tr>
</tbody>
</table>

### Table A.10: Table for Greek Alphabet (both lowercase and uppercase).

| ALPHABET |  | IOTA |  | RHO |  |  |
|----------|  |      |  |     |  |  |
| ALPHA    | α | A    | ρ | ρ  | P |
| BETA     | β | B    | ε | ε  | ε |
| GAMMA    | γ | Γ    | µ | µ  | µ |
| DELTA    | δ | Δ    | ζ | ζ  | ζ |
| EPSILON  | ε | Ε    | η | η  | η |
| ZETA     | ζ | Ζ    | θ | θ  | θ |
| ETA      | η | Η    | ι | ι  | ι |
| THETA    | θ | Θ    | ρ | ρ  | ρ |
Table A.11: 88 Constellations in the sky: Note that constellations refer to a region in the sky, not the lines connecting stars.

<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Name</th>
<th>Popular Name</th>
<th>Best time</th>
<th>α-star (Bayer-Flamsteed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Andromeda</td>
<td>The Princess of Ethiopia</td>
<td>Oct-Nov</td>
<td>Sirrah (HD3558)</td>
</tr>
<tr>
<td>ANT</td>
<td>Antlia</td>
<td>The Air Pump</td>
<td>Mar-Apr</td>
<td>(HD90610)</td>
</tr>
<tr>
<td>APS</td>
<td>Apus</td>
<td>The Bird of Paradise</td>
<td>Jul</td>
<td>(HD129078)</td>
</tr>
<tr>
<td>AQR</td>
<td>Aquarius</td>
<td>The Water Carrier</td>
<td>Aug-Oct</td>
<td>Sadalsymek (HD216131)</td>
</tr>
<tr>
<td>AQL</td>
<td>Aquila</td>
<td>The Eagle</td>
<td>Jul-Aug</td>
<td>Altair (HD187642)</td>
</tr>
<tr>
<td>ARA</td>
<td>Ara</td>
<td>The Altar</td>
<td>Jun-Jul</td>
<td>(HD158427)</td>
</tr>
<tr>
<td>ARI</td>
<td>Ariës</td>
<td>The Ram</td>
<td>Nov-Dec</td>
<td>Hamal (HD12929)</td>
</tr>
<tr>
<td>AUR</td>
<td>Auriga</td>
<td>The Charioteer</td>
<td>Jan-Feb</td>
<td>Capella (HD34029)</td>
</tr>
<tr>
<td>BOO</td>
<td>Boötes</td>
<td>The Bear Driver</td>
<td>May-Jun</td>
<td>Arcturus (HD124897)</td>
</tr>
<tr>
<td>CAE</td>
<td>Caelum</td>
<td>The Grinding Tool</td>
<td>Dec-Jan</td>
<td>(HD29875)</td>
</tr>
<tr>
<td>CAM</td>
<td>Camelopardalus</td>
<td>The Giraffe</td>
<td>Dec-May</td>
<td>(HD30442)</td>
</tr>
<tr>
<td>CNC</td>
<td>Cancer</td>
<td>The Crab</td>
<td>Feb-Mar</td>
<td>(HD76756)</td>
</tr>
<tr>
<td>CVN</td>
<td>Canes Venatici</td>
<td>The Hunting Dogs</td>
<td>Apr-May</td>
<td>Cor Caroli (HD112412)</td>
</tr>
<tr>
<td>CMA</td>
<td>Canis Major</td>
<td>The Great Dog</td>
<td>Jan-Feb</td>
<td>Sirius (HD48915)</td>
</tr>
<tr>
<td>CMI</td>
<td>Canis Minor</td>
<td>The Little Dog</td>
<td>Feb</td>
<td>Procyon (HD61421)</td>
</tr>
<tr>
<td>CAP</td>
<td>Capricornus</td>
<td>The Sea Goat</td>
<td>Aug-Sep</td>
<td>Giedi (HD192947)</td>
</tr>
<tr>
<td>CAR</td>
<td>Carina</td>
<td>The Kiel</td>
<td>Jan-Apr</td>
<td>Canopus (HD45348)</td>
</tr>
<tr>
<td>CAS</td>
<td>Cassiopeia</td>
<td>Cassiopeia</td>
<td>Oct-Dec</td>
<td>Schedar (HD3712)</td>
</tr>
<tr>
<td>CEN</td>
<td>Centaurus</td>
<td>The Centaur</td>
<td>Apr-Jun</td>
<td>Rigil Kent (HD128620)</td>
</tr>
<tr>
<td>CEP</td>
<td>Cepheus</td>
<td>Cepheus</td>
<td>Sep-Oct</td>
<td>Alderamin (HD203280)</td>
</tr>
<tr>
<td>CET</td>
<td>Cetus</td>
<td>The Sea Monster, Whale</td>
<td>Oct-Dec</td>
<td>Menkar (HD18884)</td>
</tr>
<tr>
<td>CHA</td>
<td>Chamaeleon</td>
<td>The Chamaeleon</td>
<td>May-Jun</td>
<td>(HD71243)</td>
</tr>
<tr>
<td>CIR</td>
<td>Circinus</td>
<td>The Compasses</td>
<td>May-Jun</td>
<td>(HD12898)</td>
</tr>
<tr>
<td>COL</td>
<td>Columba</td>
<td>The Dove</td>
<td>Apr-May</td>
<td>Phact (HD37795)</td>
</tr>
<tr>
<td>COM</td>
<td>Coma Berenices</td>
<td>Berenice's Hair</td>
<td>Apr-May</td>
<td>(HD114378)</td>
</tr>
<tr>
<td>CRA</td>
<td>Corona Australis</td>
<td>The Southern Crown</td>
<td>Jul-Aug</td>
<td>(HD178253)</td>
</tr>
<tr>
<td>CRB</td>
<td>Corona Borealis</td>
<td>The Northern Crown</td>
<td>Jul</td>
<td>Gemma (HD13906)</td>
</tr>
<tr>
<td>CRV</td>
<td>Corvus</td>
<td>The Crow</td>
<td>Apr-May</td>
<td>Alchiba (HD105452)</td>
</tr>
<tr>
<td>CRT</td>
<td>Crater</td>
<td>The Cup</td>
<td>Apr</td>
<td>Alkes (HD95272)</td>
</tr>
<tr>
<td>CRU</td>
<td>Cruc</td>
<td>The Southern Cross</td>
<td>Apr-May</td>
<td>Acron (HD108248)</td>
</tr>
<tr>
<td>CYG</td>
<td>Cygnus</td>
<td>The Swan</td>
<td>Aug-Sep</td>
<td>Deneb (HD197345)</td>
</tr>
<tr>
<td>DEL</td>
<td>Delphinus</td>
<td>The Dolphin</td>
<td>Aug-Sep</td>
<td>(HD196867)</td>
</tr>
<tr>
<td>DOR</td>
<td>Dorado</td>
<td>The Swordfish</td>
<td>Dec-Jan</td>
<td>(HD29305)</td>
</tr>
<tr>
<td>DRA</td>
<td>Draco</td>
<td>The Dragon</td>
<td>Mar-Sep</td>
<td>Thuban (HD123299)</td>
</tr>
<tr>
<td>EQU</td>
<td>Equuleus</td>
<td>The Foal</td>
<td>Sep</td>
<td>Kialpha (HD202447)</td>
</tr>
<tr>
<td>ERI</td>
<td>Eridanus</td>
<td>The River Eridanus</td>
<td>Dec-Jan</td>
<td>Achernar (HD10144)</td>
</tr>
<tr>
<td>FOR</td>
<td>Fornax</td>
<td>The Furnace</td>
<td>Nov-Dec</td>
<td>(HD20010)</td>
</tr>
<tr>
<td>GEM</td>
<td>Gemini</td>
<td>The Twins</td>
<td>Jan-Feb</td>
<td>Castor (HD60179)</td>
</tr>
<tr>
<td>GRU</td>
<td>Grus</td>
<td>The Crane</td>
<td>Sep-Oct</td>
<td>Alnair (HD209952)</td>
</tr>
<tr>
<td>HER</td>
<td>Hercules</td>
<td>The Hercules</td>
<td>Jul-Aug</td>
<td>Ras Algethi (HD156014)</td>
</tr>
<tr>
<td>HOR</td>
<td>Horologium</td>
<td>The Pendulum Clock</td>
<td>Nov-Dec</td>
<td>(HD26967)</td>
</tr>
<tr>
<td>HYA</td>
<td>Hydra</td>
<td>The Sea Serpent</td>
<td>Jan-Feb</td>
<td>Alphard (HD81797)</td>
</tr>
<tr>
<td>HYI</td>
<td>Hydrus</td>
<td>The Water Snake</td>
<td>Oct-Dec</td>
<td>(HD12311)</td>
</tr>
<tr>
<td>IND</td>
<td>Indus</td>
<td>The Indian</td>
<td>Aug-Oct</td>
<td>(HD196171)</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Name</th>
<th>Popular Name</th>
<th>Best time</th>
<th>α-star (Bayer-Flamsteed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAC</td>
<td>Lacerta</td>
<td>The Lizard</td>
<td>Sep-Oct</td>
<td>(HD213558)</td>
</tr>
<tr>
<td>LEO</td>
<td>Leo</td>
<td>The Lion</td>
<td>Mar-Apr</td>
<td>Regulus (HD87901)</td>
</tr>
<tr>
<td>LMI</td>
<td>Leo Minor</td>
<td>The Lion Club</td>
<td>Mar-Apr</td>
<td>No α; β (HD90337)</td>
</tr>
<tr>
<td>LEP</td>
<td>Lepus</td>
<td>The Hare</td>
<td>Jun</td>
<td>Arneb (HD36673)</td>
</tr>
<tr>
<td>LIB</td>
<td>Libra</td>
<td>The Scutum</td>
<td>May-Jun</td>
<td>Zubenekenuobi</td>
</tr>
<tr>
<td>LUP</td>
<td>Lupus</td>
<td>The Wolf</td>
<td>May-Jun</td>
<td>(HD129056)</td>
</tr>
<tr>
<td>LYN</td>
<td>Lynx</td>
<td>The Lynx</td>
<td>Jan-Mar</td>
<td>(HD80895)</td>
</tr>
<tr>
<td>LYR</td>
<td>Lyra</td>
<td>The Lyre</td>
<td>Jul-Aug</td>
<td>Vega (HD172167)</td>
</tr>
<tr>
<td>MEN</td>
<td>Meastra</td>
<td>The Table Mountain</td>
<td>Dec-Feb</td>
<td>(HD43834)</td>
</tr>
<tr>
<td>MIC</td>
<td>Microscopium</td>
<td>The Microscope</td>
<td>Aug-Sep</td>
<td>(HD198232)</td>
</tr>
<tr>
<td>MON</td>
<td>Monoceros</td>
<td>The Unicorn</td>
<td>Jan-Feb</td>
<td>(HD61935)</td>
</tr>
<tr>
<td>MUS</td>
<td>Musca</td>
<td>The Fly</td>
<td>Apr-May</td>
<td>(HD109668)</td>
</tr>
<tr>
<td>NOR</td>
<td>Norma</td>
<td>The Level</td>
<td>Jun</td>
<td>No α, β; γ (HD146143)</td>
</tr>
<tr>
<td>OCT</td>
<td>Octans</td>
<td>The Octant</td>
<td>Oct</td>
<td>(HD199532)</td>
</tr>
<tr>
<td>OPH</td>
<td>Ophiuchus</td>
<td>The Serpent Holder</td>
<td>Jun-Jul</td>
<td>Rasalhague (HD159561)</td>
</tr>
<tr>
<td>ORI</td>
<td>Orion</td>
<td>The Great Hunter</td>
<td>Dec-Jan</td>
<td>Betelgeuse (HD399801)</td>
</tr>
<tr>
<td>PAV</td>
<td>Pavon</td>
<td>The Peacock</td>
<td>Sep-Oct</td>
<td>Pescocchio (HD19324)</td>
</tr>
<tr>
<td>PEG</td>
<td>Pegasus</td>
<td>The Winged Horse</td>
<td>Sep-Oct</td>
<td>Markab (HD218045)</td>
</tr>
<tr>
<td>PER</td>
<td>Perseus</td>
<td>Perseus</td>
<td>Nov-Dec</td>
<td>Mirafak (HD20902)</td>
</tr>
<tr>
<td>PHE</td>
<td>Phoenix</td>
<td>The Phoenix</td>
<td>Oct-Nov</td>
<td>Ankaa (HD22651)</td>
</tr>
<tr>
<td>PIC</td>
<td>Pictor</td>
<td>The Painter</td>
<td>Dec-Feb</td>
<td>(HD50241)</td>
</tr>
<tr>
<td>PSC</td>
<td>Piscis</td>
<td>The Fishes</td>
<td>Oct-Dec</td>
<td>(HD12446)</td>
</tr>
<tr>
<td>PSA</td>
<td>Piscis Australis</td>
<td>The Southern Fish</td>
<td>Sep-Oct</td>
<td>Fomalhaut (HD216956)</td>
</tr>
<tr>
<td>PUP</td>
<td>Puppis</td>
<td>The Stern</td>
<td>Jan-Feb</td>
<td>No α, β; γ (HD66811)</td>
</tr>
<tr>
<td>PYX</td>
<td>Pyxis</td>
<td>The Mariner’s Compass</td>
<td>Feb-Mar</td>
<td>(HD74575)</td>
</tr>
<tr>
<td>RET</td>
<td>Reticulum</td>
<td>The Net</td>
<td>Dec</td>
<td>(HD27256)</td>
</tr>
<tr>
<td>SGB</td>
<td>Sagitta</td>
<td>The Arrow</td>
<td>Aug</td>
<td>(HD185758)</td>
</tr>
<tr>
<td>SGR</td>
<td>Sagitarius</td>
<td>The Archer</td>
<td>Jul-Aug</td>
<td>Rukbat (HD181869)</td>
</tr>
<tr>
<td>SCO</td>
<td>Scorpius</td>
<td>The Scorpion</td>
<td>Jun-Jul</td>
<td>Antares (HD148478)</td>
</tr>
<tr>
<td>SCL</td>
<td>Sculptor</td>
<td>The Sculptor</td>
<td>Oct-Nov</td>
<td>(HD57373)</td>
</tr>
<tr>
<td>SCT</td>
<td>Scutum</td>
<td>The Shield</td>
<td>Jul-Aug</td>
<td>(HD171443)</td>
</tr>
<tr>
<td>SER</td>
<td>Serpens</td>
<td>The Serpent</td>
<td>Jul-Aug</td>
<td>Unukalhai (HD140573)</td>
</tr>
<tr>
<td>SEX</td>
<td>Sextans</td>
<td>The Sextant</td>
<td>Mar-Apr</td>
<td>(HD87887)</td>
</tr>
<tr>
<td>TAU</td>
<td>Taurus</td>
<td>The Bull</td>
<td>Dec-Jan</td>
<td>Aldebaran (HD)</td>
</tr>
<tr>
<td>TEL</td>
<td>Telescopium</td>
<td>The Telescope</td>
<td>Jul-Aug</td>
<td>(HD169467)</td>
</tr>
<tr>
<td>TRI</td>
<td>Triangulum</td>
<td>The Triangle</td>
<td>Nov-Dec</td>
<td>Mothallah (HD11443)</td>
</tr>
<tr>
<td>TRA</td>
<td>Triangulum Austral</td>
<td>The Southern Triangle</td>
<td>Jun-Jul</td>
<td>Atria (HD150798)</td>
</tr>
<tr>
<td>TUC</td>
<td>Tucana</td>
<td>The Toucan</td>
<td>Sep-Nov</td>
<td>(HD211416)</td>
</tr>
<tr>
<td>UMA</td>
<td>Ursa Major</td>
<td>The Great Bear</td>
<td>Feb-May</td>
<td>Deneb (HD95689)</td>
</tr>
<tr>
<td>UMI</td>
<td>Ursa Minor</td>
<td>The Little Bear</td>
<td>May-Jun</td>
<td>Polaris (HD8890)</td>
</tr>
<tr>
<td>VEL</td>
<td>Vela</td>
<td>The Sails</td>
<td>Feb-Apr</td>
<td>No α, β; γ (HD68273)</td>
</tr>
<tr>
<td>VIR</td>
<td>Virgo</td>
<td>The Virgin</td>
<td>Apr-Jun</td>
<td>Spica (HD116658)</td>
</tr>
<tr>
<td>VOL</td>
<td>Volans</td>
<td>The Flying Fish</td>
<td>Jan-Mar</td>
<td>(HD78045)</td>
</tr>
<tr>
<td>VUL</td>
<td>Vulpecula</td>
<td>The Fox</td>
<td>Aug-Sep</td>
<td>(HD183439)</td>
</tr>
</tbody>
</table>
### Table A.12: The 26 brightest stars in the sky (data from the Hipparcos Catalogue).

<table>
<thead>
<tr>
<th>Name</th>
<th>HD</th>
<th>RA</th>
<th>DEC</th>
<th>Const.</th>
<th>Vmag(m)</th>
<th>Dist.(ly)</th>
<th>B-V</th>
<th>SpType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>48915</td>
<td>-26.72</td>
<td>~8.5 min</td>
<td>G2V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sirius</td>
<td>45348</td>
<td>6.40H</td>
<td>-52.70</td>
<td>CAR</td>
<td>-0.62</td>
<td>313.54</td>
<td>0.16</td>
<td>F0I</td>
</tr>
<tr>
<td>Canopus</td>
<td>124907</td>
<td>14.26H</td>
<td>19.18</td>
<td>BOO</td>
<td>-0.05</td>
<td>36.71</td>
<td>1.24</td>
<td>K2I</td>
</tr>
<tr>
<td>Arcturus</td>
<td>128620</td>
<td>14.66H</td>
<td>-60.84</td>
<td>CEN</td>
<td>-0.01</td>
<td>4.39</td>
<td>0.71</td>
<td>G2V</td>
</tr>
<tr>
<td>Rigilken</td>
<td>172167</td>
<td>18.62H</td>
<td>38.78</td>
<td>Lyr</td>
<td>0.03</td>
<td>25.30</td>
<td>-0.00</td>
<td>A0V</td>
</tr>
<tr>
<td>Vega</td>
<td>34029</td>
<td>5.28H</td>
<td>46.00</td>
<td>AUR</td>
<td>0.08</td>
<td>42.21</td>
<td>0.80</td>
<td>M1V</td>
</tr>
<tr>
<td>Capella</td>
<td>34085</td>
<td>5.24H</td>
<td>-8.20</td>
<td>ORI</td>
<td>0.18</td>
<td>801.42</td>
<td>-0.03</td>
<td>B8I</td>
</tr>
<tr>
<td>Rigel</td>
<td>61421</td>
<td>7.65H</td>
<td>5.22</td>
<td>CMI</td>
<td>0.40</td>
<td>11.41</td>
<td>0.43</td>
<td>F5I</td>
</tr>
<tr>
<td>Procyon</td>
<td>39801</td>
<td>5.92H</td>
<td>7.41</td>
<td>ORI</td>
<td>0.45</td>
<td>447.25</td>
<td>1.50</td>
<td>M2I</td>
</tr>
<tr>
<td>Betelgeuse</td>
<td>10144</td>
<td>1.63H</td>
<td>-57.24</td>
<td>ERI</td>
<td>0.45</td>
<td>143.91</td>
<td>-0.16</td>
<td>B3V</td>
</tr>
<tr>
<td>Achernar</td>
<td>122451</td>
<td>14.06H</td>
<td>-60.37</td>
<td>Cen</td>
<td>0.61</td>
<td>529.52</td>
<td>-0.23</td>
<td>B1I</td>
</tr>
<tr>
<td>Hadar (Agena)</td>
<td>116658</td>
<td>13.42H</td>
<td>-11.16</td>
<td>Vir</td>
<td>0.98</td>
<td>263.46</td>
<td>-0.23</td>
<td>B1V</td>
</tr>
<tr>
<td>Altair</td>
<td>187642</td>
<td>19.85H</td>
<td>8.87</td>
<td>Aql</td>
<td>0.76</td>
<td>16.77</td>
<td>0.22</td>
<td>A7I</td>
</tr>
<tr>
<td>Acrux</td>
<td>108248</td>
<td>12.44H</td>
<td>-63.10</td>
<td>Cru</td>
<td>0.77</td>
<td>322.12</td>
<td>-0.24</td>
<td>B1I</td>
</tr>
<tr>
<td>Aldabaran</td>
<td>29139</td>
<td>4.60H</td>
<td>16.51</td>
<td>Tau</td>
<td>0.87</td>
<td>65.14</td>
<td>1.54</td>
<td>K5I</td>
</tr>
<tr>
<td>Spica</td>
<td>39801</td>
<td>5.92H</td>
<td>7.41</td>
<td>ORI</td>
<td>0.45</td>
<td>447.25</td>
<td>1.50</td>
<td>M2I</td>
</tr>
<tr>
<td>Antares</td>
<td>62509</td>
<td>7.76H</td>
<td>28.03</td>
<td>Gem</td>
<td>1.16</td>
<td>33.72</td>
<td>0.99</td>
<td>K0I</td>
</tr>
<tr>
<td>Pollux</td>
<td>216956</td>
<td>22.96H</td>
<td>-29.62</td>
<td>PSA</td>
<td>1.17</td>
<td>25.08</td>
<td>0.14</td>
<td>A3V</td>
</tr>
<tr>
<td>Mimosa</td>
<td>11123</td>
<td>12.80H</td>
<td>-59.69</td>
<td>Cru</td>
<td>1.25</td>
<td>354.16</td>
<td>-0.24</td>
<td>B1I</td>
</tr>
<tr>
<td>Proxima</td>
<td>71681</td>
<td>14.66H</td>
<td>-60.85</td>
<td>Cen</td>
<td>1.35</td>
<td>4.49</td>
<td>0.90</td>
<td>K1V</td>
</tr>
<tr>
<td>Regulus</td>
<td>87901</td>
<td>10.14H</td>
<td>11.97</td>
<td>Lod</td>
<td>1.56</td>
<td>77.52</td>
<td>-0.09</td>
<td>B7V</td>
</tr>
<tr>
<td>Adhara</td>
<td>52089</td>
<td>6.98H</td>
<td>-28.97</td>
<td>CMA</td>
<td>1.50</td>
<td>433.33</td>
<td>-0.21</td>
<td>B2I</td>
</tr>
<tr>
<td>Castor</td>
<td>69179</td>
<td>7.58H</td>
<td>31.89</td>
<td>Gem</td>
<td>1.58</td>
<td>51.57</td>
<td>0.03</td>
<td>A2V</td>
</tr>
<tr>
<td>Gacrux</td>
<td>108903</td>
<td>12.52H</td>
<td>-57.11</td>
<td>Cru</td>
<td>1.59</td>
<td>87.97</td>
<td>1.60</td>
<td>M4I</td>
</tr>
<tr>
<td>Shaula</td>
<td>158926</td>
<td>17.56H</td>
<td>-37.10</td>
<td>Sco</td>
<td>1.62</td>
<td>729.42</td>
<td>-0.23</td>
<td>B2I</td>
</tr>
</tbody>
</table>

### Table A.13: Statistics of the Milky Way Galaxy from [102].

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass</td>
<td>$&gt; 2 \times 10^7 M_\odot$</td>
</tr>
<tr>
<td>Disc diameter</td>
<td>30 kpc</td>
</tr>
<tr>
<td>Disc thickness (stars)</td>
<td>1 kpc</td>
</tr>
<tr>
<td>Disc thickness (gas and dust)</td>
<td>200 pc</td>
</tr>
<tr>
<td>Halp diameter</td>
<td>50 kpc</td>
</tr>
<tr>
<td>Sun's distance from Galactic centre</td>
<td>8.5 kpc</td>
</tr>
<tr>
<td>Sun's rotational velocity</td>
<td>220 km/s</td>
</tr>
<tr>
<td>Sun's rotational period</td>
<td>$240 \times 10^6$ yr</td>
</tr>
<tr>
<td>Direction of Galactic centre (2000.0)</td>
<td>$\alpha = 17h 45.7\min \delta = -29^\circ 00'$</td>
</tr>
<tr>
<td>Direction of North Galactic Pole (2000.0)</td>
<td>$\alpha = 12h 51.4\min \delta = +27^\circ 08'$</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table A.14: List of all objects in the Messier catalog (asterisks(*) mark galaxies).

<table>
<thead>
<tr>
<th>Messier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>The Crab Nebula supernova remnant in Taurus</td>
</tr>
<tr>
<td>M2</td>
<td>globular cluster in Aquarius</td>
</tr>
<tr>
<td>M3</td>
<td>globular cluster in Canes Venatici</td>
</tr>
<tr>
<td>M4</td>
<td>globular cluster in Scorpius</td>
</tr>
<tr>
<td>M5</td>
<td>globular cluster in Serpens Caput</td>
</tr>
<tr>
<td>M6</td>
<td>The Butterfly Cluster open cluster in Scorpius</td>
</tr>
<tr>
<td>M7</td>
<td>Ptolemy's Cluster open cluster in Scorpius</td>
</tr>
<tr>
<td>M8</td>
<td>The Lagoon Nebula diffuse nebula in Sagittarius</td>
</tr>
<tr>
<td>M9</td>
<td>globular cluster in Ophiuchus</td>
</tr>
<tr>
<td>M10</td>
<td>globular cluster in Ophiuchus</td>
</tr>
<tr>
<td>M11</td>
<td>The Wild Duck Cluster open cluster in Scutum</td>
</tr>
<tr>
<td>M12</td>
<td>globular cluster in Ophiuchus</td>
</tr>
<tr>
<td>M13</td>
<td>Great Hercules Globular Cluster globular cluster in Hercules</td>
</tr>
<tr>
<td>M14</td>
<td>globular cluster in Ophiuchus</td>
</tr>
<tr>
<td>M15</td>
<td>globular cluster in Pegasus</td>
</tr>
<tr>
<td>M16</td>
<td>open cluster associated with the Eagle Nebula or Star Queen Nebula IC 4703 in Serpens Cauda</td>
</tr>
<tr>
<td>M17</td>
<td>The Omega or Swan or Horseshoe or Lobster Nebula diffuse nebula in Sagittarius</td>
</tr>
<tr>
<td>M18</td>
<td>open cluster in Sagittarius</td>
</tr>
<tr>
<td>M19</td>
<td>globular cluster in Ophiuchus</td>
</tr>
<tr>
<td>M20</td>
<td>The Trifid Nebula diffuse nebula in Sagittarius</td>
</tr>
<tr>
<td>M21</td>
<td>open cluster in Sagittarius</td>
</tr>
<tr>
<td>M22</td>
<td>globular cluster in Sagittarius</td>
</tr>
<tr>
<td>M23</td>
<td>open cluster in Sagittarius</td>
</tr>
<tr>
<td>M24</td>
<td>Milky Way Patch star cloud with open cluster NGC 6603 in Sagittarius</td>
</tr>
<tr>
<td>M25</td>
<td>open cluster in Sagittarius</td>
</tr>
<tr>
<td>M26</td>
<td>open cluster in Scutum</td>
</tr>
<tr>
<td>M27</td>
<td>The Dumbbell Nebula planetary nebula in Vulpecula</td>
</tr>
<tr>
<td>M28</td>
<td>globular cluster in Sagittarius</td>
</tr>
<tr>
<td>M29</td>
<td>open cluster in Cygnus</td>
</tr>
<tr>
<td>M30</td>
<td>globular cluster in Capricornus</td>
</tr>
<tr>
<td>M31 *</td>
<td>The Andromeda Galaxy spiral galaxy in Andromeda</td>
</tr>
<tr>
<td>M32 *</td>
<td>Satellite galaxy of M31 elliptical galaxy in Andromeda</td>
</tr>
<tr>
<td>M33 *</td>
<td>The Triangulum Galaxy (also Pinwheel) spiral galaxy in Triangulum</td>
</tr>
<tr>
<td>M34</td>
<td>open cluster in Perseus</td>
</tr>
<tr>
<td>M35</td>
<td>open cluster in Gemini</td>
</tr>
<tr>
<td>M36</td>
<td>open cluster in Auriga</td>
</tr>
<tr>
<td>M37</td>
<td>open cluster in Auriga</td>
</tr>
<tr>
<td>M38</td>
<td>open cluster in Auriga</td>
</tr>
<tr>
<td>M39</td>
<td>open cluster in Cygnus</td>
</tr>
<tr>
<td>M40</td>
<td>Double Star WNC4 in Ursa Major</td>
</tr>
<tr>
<td>M41</td>
<td>open cluster in Canis Major</td>
</tr>
<tr>
<td>M42</td>
<td>The Great Orion Nebula diffuse nebula in Orion</td>
</tr>
<tr>
<td>M43</td>
<td>part of the Orion Nebula (de Mairan's Nebula) diffuse nebula in Orion</td>
</tr>
<tr>
<td>M44</td>
<td>Praesepe, the Beehive Cluster open cluster in Cancer</td>
</tr>
<tr>
<td>M45</td>
<td>Subaru, the Pleiades—the Seven Sisters open cluster in Taurus</td>
</tr>
<tr>
<td>Number</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>M46</td>
<td>open cluster in Puppis</td>
</tr>
<tr>
<td>M47</td>
<td>open cluster in Puppis</td>
</tr>
<tr>
<td>M48</td>
<td>open cluster in Hydra</td>
</tr>
<tr>
<td>M49</td>
<td>elliptical galaxy in Virgo</td>
</tr>
<tr>
<td>M50</td>
<td>open cluster in Monoceros</td>
</tr>
<tr>
<td>M51</td>
<td>The Whirlpool Galaxy in Canes Venatici</td>
</tr>
<tr>
<td>M52</td>
<td>open cluster in Cassiopeia</td>
</tr>
<tr>
<td>M53</td>
<td>globular cluster in Coma Berenices</td>
</tr>
<tr>
<td>M54</td>
<td>globular cluster in Sagittarius</td>
</tr>
<tr>
<td>M55</td>
<td>globular cluster in Sagittarius</td>
</tr>
<tr>
<td>M56</td>
<td>globular cluster in Lyra</td>
</tr>
<tr>
<td>M57</td>
<td>The Ring Nebula planetary nebula in Lyra</td>
</tr>
<tr>
<td>M58</td>
<td>spiral galaxy in Virgo</td>
</tr>
<tr>
<td>M59</td>
<td>elliptical galaxy in Virgo</td>
</tr>
<tr>
<td>M60</td>
<td>elliptical galaxy in Virgo</td>
</tr>
<tr>
<td>M61</td>
<td>spiral galaxy in Virgo</td>
</tr>
<tr>
<td>M62</td>
<td>globular cluster in Ophiuchus</td>
</tr>
<tr>
<td>M63</td>
<td>Sunflower galaxy spiral galaxy in Canes Venatici</td>
</tr>
<tr>
<td>M64</td>
<td>Blackeye galaxy spiral galaxy in Coma Berenices</td>
</tr>
<tr>
<td>M65</td>
<td>spiral galaxy in Leo</td>
</tr>
<tr>
<td>M66</td>
<td>spiral galaxy in Leo</td>
</tr>
<tr>
<td>M67</td>
<td>open cluster in Cancer</td>
</tr>
<tr>
<td>M68</td>
<td>globular cluster in Hydra</td>
</tr>
<tr>
<td>M69</td>
<td>globular cluster in Sagittarius</td>
</tr>
<tr>
<td>M70</td>
<td>globular cluster in Sagittarius</td>
</tr>
<tr>
<td>M71</td>
<td>globular cluster in Sagitta</td>
</tr>
<tr>
<td>M72</td>
<td>globular cluster in Aquarius</td>
</tr>
<tr>
<td>M73</td>
<td>open cluster in Aquarius</td>
</tr>
<tr>
<td>M74</td>
<td>spiral galaxy in Pisces</td>
</tr>
<tr>
<td>M75</td>
<td>globular cluster in Sagittarius</td>
</tr>
<tr>
<td>M76</td>
<td>The Little Dumbell, Cork, or Butterfly planetary nebula in Perseus</td>
</tr>
<tr>
<td>M77</td>
<td>spiral galaxy in Cetus</td>
</tr>
<tr>
<td>M78</td>
<td>diffuse reflection nebula in Orion</td>
</tr>
<tr>
<td>M79</td>
<td>globular cluster in Lepus</td>
</tr>
<tr>
<td>M80</td>
<td>globular cluster in Scorpions</td>
</tr>
<tr>
<td>M81</td>
<td>Bode's Galaxy (nebula) spiral galaxy in Ursa Major</td>
</tr>
<tr>
<td>M82</td>
<td>The Cigar Galaxy irregular galaxy in Ursa Major</td>
</tr>
<tr>
<td>M83</td>
<td>The Southern Pinwheel Galaxy spiral galaxy in Hydra</td>
</tr>
<tr>
<td>M84</td>
<td>lenticular galaxy in Virgo</td>
</tr>
<tr>
<td>M85</td>
<td>lenticular galaxy in Coma Berenices</td>
</tr>
<tr>
<td>M86</td>
<td>lenticular galaxy in Virgo</td>
</tr>
<tr>
<td>M87</td>
<td>Virgo A elliptical galaxy in Virgo</td>
</tr>
<tr>
<td>M88</td>
<td>spiral galaxy in Coma Berenices</td>
</tr>
<tr>
<td>M89</td>
<td>elliptical galaxy in Virgo</td>
</tr>
<tr>
<td>M90</td>
<td>spiral galaxy in Virgo</td>
</tr>
<tr>
<td>M91</td>
<td>spiral galaxy in Coma Berenices</td>
</tr>
<tr>
<td>M92</td>
<td>globular cluster in Hercules</td>
</tr>
<tr>
<td>M93</td>
<td>open cluster in Puppis</td>
</tr>
<tr>
<td>M94</td>
<td>spiral galaxy in Canes Venatici</td>
</tr>
<tr>
<td>M95</td>
<td>spiral galaxy in Leo</td>
</tr>
</tbody>
</table>
A. Astronomy–Related

<table>
<thead>
<tr>
<th>Name</th>
<th>α</th>
<th>δ</th>
<th>Type</th>
<th>m&lt;sub&gt;pg&lt;/sub&gt;</th>
<th>M&lt;sub&gt;pg&lt;/sub&gt;</th>
<th>Distance [kpc]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milky Way Galaxy</td>
<td></td>
<td></td>
<td>Sb or Sc</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>NGC224 = M31</td>
<td>0 42.7</td>
<td>41 16</td>
<td>Sb</td>
<td>4.33</td>
<td>-20.3</td>
<td>650</td>
</tr>
<tr>
<td>NGC598 = M33</td>
<td>1 33.9</td>
<td>30 39</td>
<td>Sc</td>
<td>6.19</td>
<td>-18.5</td>
<td>740</td>
</tr>
<tr>
<td>LMC</td>
<td>5 23.7</td>
<td>-69 45</td>
<td>Irr or SBc</td>
<td>0.86</td>
<td>-17.8</td>
<td>50</td>
</tr>
<tr>
<td>SMC</td>
<td>0 52.7</td>
<td>-72 49</td>
<td>Irr</td>
<td>2.86</td>
<td>-16.2</td>
<td>60</td>
</tr>
<tr>
<td>NGC205 = M110</td>
<td>0 40.4</td>
<td>41 41</td>
<td>E6</td>
<td>8.89</td>
<td>-15.8</td>
<td>650</td>
</tr>
<tr>
<td>NGC221 = M32</td>
<td>0 42.7</td>
<td>40 52</td>
<td>E2</td>
<td>9.06</td>
<td>-15.6</td>
<td>650</td>
</tr>
<tr>
<td>NGC6822</td>
<td>19 45.0</td>
<td>-14 48</td>
<td>Irr</td>
<td>9.21</td>
<td>-15.3</td>
<td>520</td>
</tr>
<tr>
<td>NGC185</td>
<td>0 38.9</td>
<td>48 20</td>
<td>E0</td>
<td>10.29</td>
<td>-14.4</td>
<td>650</td>
</tr>
<tr>
<td>IC1613</td>
<td>1 04.8</td>
<td>2 07</td>
<td>Irr</td>
<td>10.00</td>
<td>-14.4</td>
<td>740</td>
</tr>
<tr>
<td>NGC147</td>
<td>0 33.2</td>
<td>48 30</td>
<td>dE4</td>
<td>10.57</td>
<td>-14.1</td>
<td>650</td>
</tr>
<tr>
<td>Fornax</td>
<td>2 39.9</td>
<td>-34 32</td>
<td>dE</td>
<td>9.1</td>
<td>-12</td>
<td>190</td>
</tr>
<tr>
<td>AndI</td>
<td>0 45.7</td>
<td>38 01</td>
<td>dE</td>
<td>13.5</td>
<td>-11</td>
<td>650</td>
</tr>
<tr>
<td>AndII</td>
<td>1 16.5</td>
<td>33 26</td>
<td>dE</td>
<td>13.5</td>
<td>-11</td>
<td>650</td>
</tr>
<tr>
<td>AndIII</td>
<td>0 35.3</td>
<td>36 31</td>
<td>dE</td>
<td>13.5</td>
<td>-11</td>
<td>650</td>
</tr>
<tr>
<td>AndIV</td>
<td>0 42.4</td>
<td>40 35</td>
<td>dE</td>
<td>–</td>
<td>–</td>
<td>650</td>
</tr>
<tr>
<td>LeoI</td>
<td>10 08.5</td>
<td>12 19</td>
<td>dE</td>
<td>11.27</td>
<td>-10</td>
<td>230</td>
</tr>
<tr>
<td>Sculptor</td>
<td>1 00.2</td>
<td>-33 42</td>
<td>dE</td>
<td>10.5</td>
<td>-9.2</td>
<td>90</td>
</tr>
<tr>
<td>LeoII</td>
<td>11 13.5</td>
<td>22 09</td>
<td>dE</td>
<td>12.85</td>
<td>-9</td>
<td>230</td>
</tr>
<tr>
<td>Draco</td>
<td>17 20.1</td>
<td>57 55</td>
<td>dE</td>
<td>–</td>
<td>–</td>
<td>80</td>
</tr>
<tr>
<td>Ursa Minor</td>
<td>15 08.8</td>
<td>67 12</td>
<td>dE</td>
<td>–</td>
<td>?</td>
<td>80</td>
</tr>
<tr>
<td>Carina</td>
<td>6 41.6</td>
<td>-50 58</td>
<td>dE</td>
<td>–</td>
<td>?</td>
<td>170</td>
</tr>
<tr>
<td>LGS3</td>
<td>1 03.8</td>
<td>21 53</td>
<td>?</td>
<td>–</td>
<td>?</td>
<td>650</td>
</tr>
<tr>
<td>Sextans</td>
<td>10 13.0</td>
<td>-1 36</td>
<td>dE</td>
<td>–</td>
<td>–</td>
<td>85</td>
</tr>
</tbody>
</table>

Table A.15: Members of the Local Group of Galaxies from [102].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table A.16: Brightest galaxies derived from Brent Tully’s galaxy catalog (sorted by the apparent magnitude).

<table>
<thead>
<tr>
<th>NGC</th>
<th>Common Name</th>
<th>RA</th>
<th>DEC</th>
<th>Hubble Type</th>
<th>m</th>
<th>M</th>
<th>Dist</th>
<th>Diam</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMC</td>
<td>5.36</td>
<td>-69.91</td>
<td>9</td>
<td>0.8</td>
<td>-17.9</td>
<td>54.9</td>
<td>9.4</td>
</tr>
<tr>
<td>292</td>
<td>SMC</td>
<td>20.99</td>
<td>-53.56</td>
<td>9</td>
<td>2.4</td>
<td>-16.6</td>
<td>64.4</td>
<td>13.8</td>
</tr>
<tr>
<td>224</td>
<td>M31 Andromeda</td>
<td>0.68</td>
<td>41.05</td>
<td>3</td>
<td>4.3</td>
<td>-20.4</td>
<td>854.0</td>
<td>46.8</td>
</tr>
<tr>
<td></td>
<td>IC342</td>
<td>3.78</td>
<td>68.31</td>
<td>5</td>
<td>6.1</td>
<td>-20.7</td>
<td>2249.6</td>
<td>24.5</td>
</tr>
<tr>
<td>598</td>
<td>M33 Pinwheel (Triangular)</td>
<td>1.56</td>
<td>30.66</td>
<td>5</td>
<td>6.3</td>
<td>-19.0</td>
<td>1126.8</td>
<td>21.8</td>
</tr>
<tr>
<td>3031</td>
<td>Boole’s Galaxies</td>
<td>9.91</td>
<td>69.20</td>
<td>2</td>
<td>6.7</td>
<td>-20.5</td>
<td>2715.1</td>
<td>17.8</td>
</tr>
<tr>
<td>5128</td>
<td>Hamburger Galaxy</td>
<td>13.42</td>
<td>-43.01</td>
<td>-3</td>
<td>7.7</td>
<td>-20.4</td>
<td>4260.7</td>
<td>35.9</td>
</tr>
<tr>
<td>5236</td>
<td>M83 Southern Pinwheel Galaxy</td>
<td>13.62</td>
<td>-29.89</td>
<td>5</td>
<td>7.9</td>
<td>-21.0</td>
<td>5935.8</td>
<td>22.8</td>
</tr>
<tr>
<td>253</td>
<td>Sculptor (Silver Coin)</td>
<td>0.80</td>
<td>-25.17</td>
<td>6</td>
<td>7.9</td>
<td>-18.6</td>
<td>2038.8</td>
<td>16.0</td>
</tr>
<tr>
<td>5457</td>
<td>M101 Pinwheel Galaxy</td>
<td>14.05</td>
<td>54.37</td>
<td>6</td>
<td>8.3</td>
<td>-20.2</td>
<td>4888.4</td>
<td>44.1</td>
</tr>
<tr>
<td></td>
<td>M104 Sombrero Galaxy</td>
<td>12.66</td>
<td>-11.71</td>
<td>1</td>
<td>9.0</td>
<td>-20.5</td>
<td>7993.5</td>
<td>19.4</td>
</tr>
<tr>
<td>4258</td>
<td>—</td>
<td>12.31</td>
<td>47.31</td>
<td>3</td>
<td>9.1</td>
<td>-20.9</td>
<td>10048.9</td>
<td>49.8</td>
</tr>
<tr>
<td>6744</td>
<td>NGC6744 Fav</td>
<td>19.16</td>
<td>-63.85</td>
<td>4</td>
<td>9.1</td>
<td>-20.1</td>
<td>7099.9</td>
<td>43.3</td>
</tr>
<tr>
<td>3034</td>
<td>M82 Cigar Galaxy</td>
<td>9.92</td>
<td>69.68</td>
<td>7</td>
<td>9.2</td>
<td>-18.1</td>
<td>2808.6</td>
<td>8.6</td>
</tr>
<tr>
<td>4826</td>
<td>M64 Blackeye (Sleeping Beauty)Galaxy</td>
<td>12.94</td>
<td>21.71</td>
<td>3</td>
<td>9.3</td>
<td>-18.8</td>
<td>4152.8</td>
<td>13.0</td>
</tr>
<tr>
<td>4945</td>
<td>—</td>
<td>13.09</td>
<td>-49.50</td>
<td>7</td>
<td>9.3</td>
<td>-18.9</td>
<td>4215.0</td>
<td>25.1</td>
</tr>
<tr>
<td>4472</td>
<td>M49 Vir</td>
<td>12.50</td>
<td>7.98</td>
<td>-5</td>
<td>9.3</td>
<td>-21.1</td>
<td>11876.4</td>
<td>33.9</td>
</tr>
<tr>
<td>5055</td>
<td>M63 Sunflower Galaxy</td>
<td>13.26</td>
<td>42.05</td>
<td>4</td>
<td>9.3</td>
<td>-19.5</td>
<td>5864.6</td>
<td>22.6</td>
</tr>
<tr>
<td>1316</td>
<td>Fornax A</td>
<td>3.38</td>
<td>-37.22</td>
<td>-2</td>
<td>9.4</td>
<td>-21.5</td>
<td>15086.8</td>
<td>49.4</td>
</tr>
<tr>
<td>1291</td>
<td>—</td>
<td>3.28</td>
<td>-41.08</td>
<td>0</td>
<td>9.4</td>
<td>-20.0</td>
<td>7541.6</td>
<td>21.5</td>
</tr>
<tr>
<td>1068</td>
<td>M77 Cetus A</td>
<td>2.71</td>
<td>0.01</td>
<td>3</td>
<td>9.5</td>
<td>-20.4</td>
<td>9546.8</td>
<td>21.1</td>
</tr>
<tr>
<td>676</td>
<td>—</td>
<td>1.82</td>
<td>5.90</td>
<td>0</td>
<td>9.5</td>
<td>-21.5</td>
<td>15916.9</td>
<td>19.8</td>
</tr>
<tr>
<td>4486</td>
<td>M87 Smoking Gun (Virgo)</td>
<td>12.51</td>
<td>12.37</td>
<td>-5</td>
<td>9.5</td>
<td>-21.0</td>
<td>13101.0</td>
<td>31.8</td>
</tr>
<tr>
<td>2903</td>
<td>—</td>
<td>9.34</td>
<td>21.49</td>
<td>4</td>
<td>9.6</td>
<td>-20.0</td>
<td>8334.1</td>
<td>30.6</td>
</tr>
<tr>
<td>3627</td>
<td>M66 Leo Triplet</td>
<td>11.34</td>
<td>12.95</td>
<td>3</td>
<td>9.6</td>
<td>-19.8</td>
<td>7778.4</td>
<td>19.8</td>
</tr>
<tr>
<td>1313</td>
<td>—</td>
<td>3.22</td>
<td>-66.47</td>
<td>6</td>
<td>9.7</td>
<td>-16.3</td>
<td>1560.3</td>
<td>4.1</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
A.3 Datasets for Virtual Astronomy

This section lists various available datasets for virtual astronomy. For each dataset, we detail the nature of the data, the source of the data, how to make use of the data in your program, and the data credit, etc.

A. The Solar system and Earth.


   - [Description]
     The dataset provides coordinates of the Sun, Moon, and nine planets relative to the Earth over a period of time. It is provided by the JPL (Jet Propulsion Laboratory).

   - [Source]
     Data, documentation, and C/Fortran programs from

   - [Implementation]
     First, download corresponding dataset: DE200, DE403, DE405, or DE406.
     - DE200: from 1599 DEC 09 to 2169 MAR 31
       It is the basis of the Astronomical Almanac since 1984.
     - DE403: from 1949 DEC 14 to 2050 JAN 02
       It is a standard for IERS (International Earth Rotation Service).
     - DE405: from 1599 DEC 09 to 2201 FEB 20
       This dataset includes both nutation and libration; it was created during May-June 1997, based on the International Celestial Reference Frame (ICRF).
A. Astronomy-Related

- DE406: from -3001 FEB 04 to +3000 MAY 06

It is the New "JPL Long Ephemeris," but does not have nutation nor libration. It is claimed that the interpolating accuracy of this dataset is no worse than 25 meters for any planet and no worse than 1 meter for the Moon. In addition, due to its huge storage size, the whole dataset is divided into several 300-year blocks, each of size about 10MB.

Then, get the source program from the same JPL website:

Following the instruction manual from the above ftp site, we can decode the data and apply Tchebyshev interpolation to compute the positional information (position and velocity) of the Sun, the Moon, and planets.

- [Credit]
  - Dataset by Standish, E. M.
  - Programs by David Hoffman (david.a.hoffman@jsc.nasa.gov) (newer version) and Kourosh Arfa-Kaboodvand (older version)

- [Note]
  In our research, we created the Ephemeris API library to take care of this JPL dataset; it encapsulates David Hoffman’s code as a C++ class and adds in data caching capability, etc.


- [Description]
  The NORAD dataset provides the positional information of various man-made satellites over the Earth and the Kunstman dataset records the launching information of satellites.
A. Astronomy-Related

- [Source]
  Download data and documentation from
  http://www.celestrak.com/NORAD/elements,
  C/C++/Java programs from
  http://www.projectpluto.com/sat.code.htm,
  and the Kunstman dataset from

- [Implementation]
  In the NORAD dataset, each satellite consists of three lines of text data: The
  first line is the name of the satellite; the next two lines include the satellite type
  and various orbital information of the satellite. After decoding this two-line el­
  ement, we can apply one of the five protocols to compute the satellite trajectory:
  SGP, SGP4, and SGP8 are for near earth satellites whereas SDP4 and SDP8 are
  for deep-space satellites. Details for the two-line element sets can be found in
  the Space Track Report Number 3.

  In practice, the coding for the data parsing and the trajectory computation
  can be found in the above “satcode.htm” webpage. In addition to the satellite
  position data from NORAD, we have the Kunstman dataset that offers launch­
  ing information about satellites. This includes launching year and site, country
  ownership information, etc.

- [Credit]
  Mathematical models for satellite trajectory used in the NORAD dataset were
  derived by F. Hoots and R. L. Roehrich [92]. The program for decoding and
  computing the orbital information in the NORAD dataset is by Neoklis Kyri­
  azis, Thierry Marais, and the Project Pluto team. The Kunstman data is recorded
  by the BWGS (Belgian Working Group Satellites) (Contact email:
  bwgs_belgianworkinggroupssatellites@hotmail.com.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
- [Note]

Furthermore, John N. Huffman of AVL at IU made wavefront “obj” models for different types of satellites. Note that a wavefront obj model is a data format for storing three-dimensional polygonal objects. In our implementation, we integrated the Kunstman dataset as well as these wavefront models with the satellite trajectory program — this is the satellite library for OpenGL rendering we used in this research work.

3. The Heliosphere: Boundary of the Solar Wind [109, 108]

- [Description]

This includes three-dimensional models of the Termination shock, the Heliopause, and the Bow shock based on physical simulations; Renderings can be found in Section 8.6.

- [Source]

From Timur Linde, University of Chicago (t-linde@uchicago.edu).

- [Implementation]

In this research, we apply animated textures to the Bow shock model (by Eric Wernert) and different coloring to the three surface models.

- [Credit]

Geometric models kindly provided by Timur Linde, University of Chicago.

4. World Time Zone Map

- [Description]

This dataset includes data points for country and time zone boundaries.

- [Source]

- **[Implementation]**
  First, the dataset can be downloaded from
  
  www.manifold.net/download/freemaps.html,
  
  and then follow the instructions to decode the data.

- **[Credit]**
  From Manifold Net Ltd.

- **[Note]**
  - Distributed under the Free Software Foundation’s General Public License (GPL)
  - More information about time zones can be found in
    

5. **Location of the International Date Line (IDL)**

- **[Description]**
  This is basically a table, listing the coordinates (latitude, longitude) of the points marking the ends of the circle segments defining the location of the International Date Line in major 20th-century atlases.

- **[Source]**
  From www.phys.uu.nl/~vgent/idl/idl.htm.

- **[Implementation]**
  Grep the table presented at the end of the above webpage that lists the (latitude, longitude) coordinates.

- **[Credit]**
  Website by Robert H. van Gent

- **[Note]**
Astronomy-Related

The Kiribati adjustment of the IDL was made in 1995, but some map publishers still have not adopted these changes.


Here we list the coordinates (latitude, longitude) of the IDL:

\[
\begin{align*}
(90, 0) &\to (75, -180) \\
(68, -169) &\to (65.5, -169) \\
(58.95, 180) &\to (53, 170) \\
(48, 179.44) &\to (48, 180) \\
(2, -180) &\to (-1, -177) \\
(-1, -159) &\to (3, -159) \\
(3, -161) &\to (5, -161) \\
(5, -156) &\to (-5, -149) \\
(-12, -149) &\to (-12, -154) \\
(-7.5, -154) &\to (-7.5, -178) \\
(-15, -172.5) &\to (-65, -172.5) \\
(-51, -180) &\to (-60, -180) \\
(-90, 0) &
\end{align*}
\]

Coordinates (Latitude and Longitude) along the International Date Line (IDL) from North Pole to South Pole.

B. Stars.

1. The Bright Star Catalog (BSC) [90, 91]

   - [Description]

   There are two popular versions of this catalog: BSC4 and BSC5. This catalog contains all bright stars visible to the human eye; typically, for BSC5, there
are 9110 data items in the catalog, of which 14 are not stars, so there are 9096 stars altogether. An important property of this dataset is that it provides the trigonometric parallax of the stars, which permits us to locate three-dimensional locations of stars; it also provides proper motion and radial velocities. However, since the quality and quantity for parallax provided by the Hipparcos Catalogue is better, this catalog superseded by the Hipparcos data in our later research.

- **[Source]**

This catalog (BSC5) can be downloaded from

http://cdsweb.u-strasbg.fr/viz-bin/Cat?V/50

From the data center

http://cdsweb.u-strasbg.fr/CDS.html

Search for the “bright star catalog” dataset or go to the Yale university webpage and order the catalog:

www.astro.yale.edu/dept/resources/publications.html.

- **[Implementation]**

Basically, the dataset is in the form of a text file in which each star takes an entry (a line) in the file, with fixed columns for each data item. A description of the file format and the meaning of each data column can be found in the above webpage.

- **[Credit]**

By Yale Professor Dorrit Hoffleit et al.

2. The Hipparcos (High Precision Parallax Collecting Satellite) Catalogue [53, 137]

- **[Description]**

Similar to the BSC, this catalogue contains a large quantity of very high quality astrometric and photometric data including proper motions, parallaxes, trigonometric, photometry, fundamental information, etc. There are 118218 records
(stars) in the main Catalogue.

- [Source]
The main Catalogue can be downloaded from

  http://cdsweb.u-strasbg.fr/viz-bin/Cat?I/239,

Go to data center

  http://cdsweb.u-strasbg.fr/CDS.html,

and search for the “hipparcos” dataset or go to the Hipparcos page at the European Space Agency (ESA):


- [Implementation]
Similar to BSC, this Catalogue is a fixed column text file. However, since it has far more records than the BSC file, the data size is huge (52MB). Therefore, in the actual implementation, it is better to load in necessary data and process the data, instead of loading everything into the memory. The description of the file format can be found in the “hipparcos.readme” file that can be downloaded from the archive or in the webpage above.

- [Credit]
By the European Space Agency (ESA).

- [Note]
The radial velocity is not available in the Hipparcos Catalogue, so we need the mean radial velocity catalog below.

3. Catalog of Constellation Boundary Data [3]

- [Description]
This dataset provides list of points outlining the boundaries of the standard eighty-eight constellations.
Astronomy-Related

- **[Source]**

   Download the constellation boundary data from
   
   http://cdsweb.u-strasbg.fr/viz-bin/Cat?VI/49
   
   or go to data center
   
   http://cdsweb.u-strasbg.fr/CDS.html
   
   and search for the constellation dataset.

- **[Implementation]**

   After downloading data files from the above website, the dataset itself is simply a text file listing these point entries. In the text file, each constellation boundary is stored as a list of points that constructs a closed line loop. Each line in the data file represents a single point in the line loop and there are four elements per data line: RA, DEC, constellation abbreviation (see table A.11 for the abbreviation name), and type.

- **[Credit]**

   This catalog was made by Davenhall A.C. and Leggett S.K. in 1989. Note that the modern list of 88 constellations was adopted in 1922 by the International Astronomical Union and the associated boundary lines were then drawn up by Eugene Delporte along arcs of RA and DEC in 1875.

- **[Note]**

   - There are two types of points: O and I. O means original points whereas I means interpolated points. Basically, each closed line loop corresponding to a constellation is a polygon on the sky — original points are vertices of the polygon while interpolated points are points along edges of the polygon.
   
   - There are two data files (RA/DEC): one for year 1875 and the other for year 2000.
Specifically, the constellation Serpens is divided into two parts by the constellation Ophiuchus, it is broken into two closed line loops in the data file, denoted as SER1 and SER2.

4. General Catalog of mean radial velocities [10]

- [Description]
This catalog presents mean radial velocities for Galactic stars which supplements previous WEB catalog [48]. This catalog contains new mean velocities for 36145 stars, among which 20574 are new. Only observations obtained with adequate spectral resolution and standardized to the IAU or Wilson [180] velocity systems have been used. It is the latest record of mean radial velocities.

- [Source]
Obtain the data from

  http://cdsweb.u-strasbg.fr/viz-bin/Cat?III/213 ,

or go to data center

  http://cdsweb.u-strasbg.fr/CDS.html ,

and search for the “mean radial velocity” dataset.

- [Implementation]
Important data items available in this catalog are the mean radial velocity, its error, and the associated HD and HIC numbers. HD and HIC numbers refer to Henry Draper and Hipparcos numbers respectively.

  We can merge this dataset with the Hipparcos Catalogue using the common HD and HIC numbers. Afterwards, we can compute the space velocity (and its error) of associated stars using the derived equations in Subsection 9.2. Furthermore, it is important to notice the local standard of rest and peculiar motions described in Subsection 9.2 when plotting three-dimensional star velocities.

- [Credit]

5. Extra-solar planets dataset [148, 112, 113]

- [Description]
  This dataset provides various information about extra-solar planets. Up to May 2003, about 102 extra-solar planets had been discovered, but this number increased to 110 in July 2003. This is an active research area in astronomy and the extra-solar planet dataset continues to grow monthly.

- [Source]
  Open the webpage at exoplanets.org:
  
  http://exoplanets.org/almanacframe.html,
  
  and download the text version of the dataset.

- [Implementation]
  In this dataset, each extra-solar planet has seven attributes: extra-solar planet ID, name of the associated star, planet mass (relative to mass_{Jupiter} which is 1), orbital period (in days), semi-major axis (in A.U.), eccentricity, and K (m/s).

- [Credit]
  This catalog is compiled by “California & Carnegie Planet Search”.

- [Note]
  - Some of the stars are named “Not a Planet”, which means that astronomers cannot yet observe any associated planets for these stars.
  - These planet orbits are very small in size compared to the stellar scale; when we navigate among the stars, like orbits in our Solar system, orbits of extra-solar planets will not be visible. Thus, we need a tuning parameter to scale the size of these orbits to make them artificially visible. Typically, a factor of 100,000 is needed to make them visible at interstellar scale.
C. The Milky Way Galaxy.

1. The Optical Night sky: Mellinger image [119]

   - [Description]
     This two-dimensional all-sky panoramic image shows the visible (optical) range of the night sky. The xy-dimensions of the image span Galactic $l$ and $b$, respectively. The highest available resolution at present is $14k \times 7k$ (in pixel units), in $(l,b)$ alignment.

   - [Source]
     The data is obtainable from
     http://home.arcor-online.de/axel.mellinger.

   - [Implementation]
     This all-sky image is used as an environment-map to represent the panoramic view of the Milky Way Galaxy.

   - [Credit]

2. Multi-wavelength all-sky images [119, 73, 85, 45, 157, 147, 40, 86, 44, 88, 64, 111, 87, 89]

   - [Description]
     In addition to the Mellinger image, which corresponds to the optical range, we gathered panoramic images for various ranges in the electromagnetic (EM) spectrum:

     (a) Egret Data (Gamma) [85]
- [Source]
  http://lheawww.gsfc.nasa.gov/docs/gamcosray/EGRET/pictures.html, or http://skyview.gsfc.nasa.gov/cgi-bin/survey.pl#
  EGRET, or http://antwrp.gsfc.nasa.gov/apod/ap980321.html
- [Credit]
  Hartman R. C. et.al. [85]

(b) ROSAT (Hard X-ray) [45]
- [Source]
- [Credit]
  S. Digel, S. Snowden, the ROSAT Project, and MPE

(c) ROSAT PSPC All-Sky Survey (Soft X-ray) [157]
- [Source]
  http://www.xray.mpe.mpg.de/rosat/survey/sxrb/12/fits.html
  or http://heasarc.gsfc.nasa.gov/docs/rosat/survey
- [Credit]
  Snowden S. L. et.al. [157]

(d) Dust Map (Infra-red) [147]
- [Source]
- [Credit]
  Schlegel D. J., Finkbeiner D. P., and Davis M.

(e) CO Survey (Microwave) [40]
- [Source]
A. Astronomy-Related


- [Credit]
  Dame T. M., Hartmann D., and Thaddeus P.

(f) Neutral Hydrogen (HI) spectrum [44, 86]

- [Source]

- [Credit]
  Independently by "Dickey J. M. and Lockman F. J. [44]" and "D. Hartmann and W. B. Burton [86]"

(g) 408MHz (Radio) [88]

- [Source]

- [Credit]
  Haslam C. G. T., Stoffel H., Salter C. J., and Wilson W. E.

(h) LISM (Local Interstellar Medium) [73]

- [Source]
  PCF construct using data from [73]: http://spacsun.rice.edu/~twg/lism.html
- [Credit]
  Genova R., Beckman J. E., Molaro P., and Vladilo G.

(i) **Halpha emission (Microwave)** [64]

- [Source]
  
  http://www.astro.princeton.edu/~dfink/halpha(or directly from
  http://www.astro.princeton.edu/~dfink/halpha/jpegs/com-
  big.jpg

- [Credit]
  Finkbeiner D. P.

(j) **300 CO clouds (Graphical Map)** [87, 111]

- [Source]
  
  We constructed the image of Molecular Gas (CO) map from [87, 111]

- [Credit]
  “Hartmann D., Magnani L., and Thaddeus P. [87]” and “Magnani L., Hart-
  mann D., Holcomb S. L., Smith L. E., and Thaddeus P. [111]”

(k) **Polarization Data (Stellar polarization catalogs)** [89]

- [Source]
  
  PCF construct using data [89] from http://vizier.u-strasbg.fr/
  viz-bin/VizieR?-source=II/226

- [Credit]
  Heiles C.

3. **The Wainscoat/Dwek G2 Galactic bulge model** (or the Goddard Milky Way model) [172, 49]

- [Description]
  
  This is a point-based representation of the three-dimensional Milky Way Galaxy
produced by Monte Carlo sampling, in which there are two versions: one with 40,000 data points and one with 4,000 data points. The model itself is stored in VRML (Virtual Reality Modeling Language) format. This is part of the “Interactive Universe” project at NASA:


This project is documented in the National Space Science Data Center On-line news: Dec 1997 issue, URL:


- [Source]
The model was obtained, may be obtained from

http://rumba.gsfc.nasa.gov/iuniverse/galaxy

OR


- [Implementation]
We convert this point-based model to a Particle model for display, see Section 10.4 for detail.

- [Credit]
Goddard Space Flight Center at NASA (Wainscoat et al. [172] and Dwek et al. [49]).

---

D. Galaxies.

1. Brent Tully’s Galaxy Catalog [169, 168]
- [Description]
  This dataset catalog about 35,000 galaxies; each has the galaxy ID/name, coordinate, galaxy diameter, absolute magnitude, distance in velocity units, etc.

- [Source]
  Contact: Brent Tully (tully@ifa.Hawaii.edu).

- [Implementation]
  Similar to the Hipparcos dataset, this catalog is a fixed column text file with each galaxy record corresponding to an individual line in the file. Each galaxy record includes the following attributes: the galaxy ID/name, galaxy coordinate, galaxy diameter, absolute magnitude, distance in velocity units, hold position angle (see Chapter 11), ratio of principal axes, etc.

- [Credit]
  Brent Tully at University of Hawaii.

2. Cosmic Background Radiation (CMBR) [15, 13] and Wilkinson Microwave Anisotropy Probe (WMAP) [14]

- [Description]
  These are all-sky panoramic images of the fluctuations in the cosmic background radiation (CMBR). Detailed information about the earlier COBE project and the more recent WMAP project can be found in http://lambda.gsfc.nasa.gov/product/cobe.

- [Source]
  Obtain the data from

- [Implementation]
These all-sky images are used as environment-maps to represent the event horizon of the visible Universe before the formation of galaxies.

- [Credit]

The COBE (Cosmic Background Explorer) team and the WMAP (Wilkinson Microwave Anisotropy Probe) mission at NASA.
A.4 Coordinate Systems

Astronomers use a variety of coordinate systems for data at different scales. Therefore, during the development of the virtual astronomy system, we had to support a variety of astronomical coordinate systems. In this section, we detail some common coordinate systems in astronomy as well as the appropriate transformation methods applicable to computer graphics.

The Terrestrial Coordinate system

![Terrestrial Coordinate System Diagram](image)

Figure A.1: The Terrestrial Coordinate system: Longitude and Latitude.

- **Aim:**
  Locate places on Earth using longitude and latitude.

- **Fact:**
  The Earth rotates about its axis, an imaginary line through the North and South Poles.
The Equator is the great circle on Earth's surface midway between the two poles. The Prime Meridian is a half-circle on Earth's surface from the North Pole to the South Pole through the Royal Observatory in Greenwich, England. Figure A.1 depicts the coordinate system.

- **Longitude:**
  Angle, east or west, from the prime meridian along the Equator to the point nearest to the chosen location (range: $180^\circ W$ to $180^\circ E$).

- **Latitude:**
  Angle, north or south, between the chosen location and the Equator, whose latitude is $0^\circ$ (range: $90^\circ S$ to $90^\circ N$).

**The Equatorial (or Celestial) Coordinate system**

![Equatorial Coordinate System Diagram](image)

Figure A.2: The Equatorial Coordinate system: Right Ascension (RA) and Declination (DEC).
A. **Astronomy-Related**

- **Aim:**
  Locate points on the sky relative to the Earth using Right Ascension (RA) and Declination (DEC).

- **Fact:**
  The sky above the Earth can be thought of as a Celestial sphere with the Earth at the center. We can project the North Pole, the South Pole, and the Equator from Earth's center onto this Celestial sphere to create the *North Celestial Pole* (NCP), the *South Celestial Pole* (SCP), and the *Celestial Equator* respectively. Figure A.2 depicts the coordinate system.

- **Vernal Equinox:**
  Since Greenwich is not a fixed point on the sky, we cannot use it as a reference point for the Equatorial coordinate system. Astronomers therefore chose to define the vernal equinox as the reference location. Basically, it is the Sun's direction from Earth mapped on the Celestial sphere corresponding to the first day of Spring, usually around the 21st of March.

- **Right Ascension (RA):**
  Similar to longitude, RA is the angle measured eastward along the Celestial Equator from the vernal equinox (range: 0h to 24h (in hours), one hour being 15 degrees).

- **Declination (DEC):**
  Similar to latitude, DEC is the vertical angle from the Celestial Equator (range: $-90^\circ$ to $+90^\circ$ (in degrees)) (see Figure A.2).

**Note:**  "Equinox" means "equal night" in Latin; during the vernal and autumnal equinoxes, daytime and nighttime are approximately the same length.
The Ecliptic Coordinate system

Aim:
Locate points on the sky relative to the orientation of the Earth's orbit around the Sun.

Fact:
Basically, the annual motion of the Sun is eastward on the Celestial sphere and its path sweeps out a circle on the sphere such that the path intersects with the Celestial Equator at the vernal and autumnal equinoxes. This path is called the Ecliptic. Because the Earth's rotational axis is tilted by $23.5^\circ$ relative to its orbital plane around the Sun, the Ecliptic is tilted by $23.5^\circ$ from the Celestial Equator. Figure A.3 depicts the geometry.

Note: The Ecliptic passes through the twelve zodiacal constellations: Virgo, Libra, Scorpius, Sagittarius, Capricornus, Aquarius, Pisces, Aries, Taurus, Gemini, Cancer, and Leo.
The Galactic Coordinate system

Figure A.4: The Galactic Coordinate system: Galactic Longitude (l) and Galactic Latitude (b).

- **Aim:**
  Locate points on the sky relative to the Milky Way using Galactic Longitude (l) and Galactic Latitude (b).

- **Fact:**
  Our Solar system is located in a spiral arm of the Milky Way galaxy, which is a large disk (galactic disk) similar to a pizza. The Galactic Equator is the intersection between the plane of Milky Way (galactic plane) and the Celestial sphere. Like Greenwich and the vernal equinox, the direction from the Earth (or the Sun) to the galactic center serves as the reference location in this coordinate system.

- **Galactic Longitude (l):**
Similar to longitude, Galactic Longitude is the angle ($l$) measured eastward (anti-clockwise about the North Galactic Pole) along the Galactic Equator relative to the direction towards the galactic center (range: $0^\circ$ to $360^\circ$ (in degrees)).

Galactic Latitude ($b$):
Similar to latitude, Galactic Latitude is the vertical angle ($b$) from the Galactic Equator (range: $-90^\circ$ to $+90^\circ$ (in degrees)) (see Figure A.4).

The Super-Galactic Coordinate system

![Diagram of Super-Galactic Coordinate system](image)

Figure A.5: The Super-Galactic Coordinate system (non-stippled axes) with respect to the Galactic Coordinate system (stippled axes).

Aim:
Locate galaxies on the sky, used specifically at the scale of galaxies.

Fact:
This coordinate system is for recording galaxy coordinates, for example, in Brent Tully's galaxy catalog. Figure A.5 depicts the Super-Galactic coordinate system: the red/green/blue axes correspond to $X/Y/Z$ axes whereas the stippled and non-stippled
axes correspond to galactic (Gx/Gy/Gz) and super-galactic (SGx/SGy/SGz) axes, respectively. We can see that the galactic Z-axis is very close to the super-galactic Y-axis. Further clarification on the exact meaning of this coordinate system can be found in the *Third Reference Catalogue of Bright Galaxies* [41] by de Vaucouleurs et al. 1991.

**Transformations between coordinate systems**

Note that the transformations among the above coordinate systems are stable in time except for the terrestrial coordinate system, whose transformations to other coordinate systems change rapidly with time. Furthermore, astronomical datasets we encounter usually are stored in Equatorial, Galactic, or Super-Galactic coordinate systems. To convert coordinates between these systems, astronomers commonly make use of spherical geometry (see the book “Spherical Astronomy [75]” for detail). However, in computer graphics, it is more convenient to use $3 \times 3$ or $4 \times 4$ rotation matrices for the transformations. Here we list the derived transformation matrices:

- **From Equatorial to Galactic**:
  \[
  \begin{pmatrix}
  -0.054802 & 0.493969 & -0.867751 & 0.000000 \\
  -0.873421 & -0.444861 & -0.198078 & 0.000000 \\
  -0.483873 & 0.747056 & 0.455822 & 0.000000 \\
  0.000000 & 0.000000 & 0.000000 & 1.000000
  \end{pmatrix}
  \]

- **From Galactic to Equatorial**:
  \[
  \begin{pmatrix}
  -0.054802 & -0.873421 & -0.483873 & 0.000000 \\
  0.493969 & -0.444861 & 0.747056 & 0.000000 \\
  -0.867751 & -0.198078 & 0.455822 & 0.000000 \\
  0.000000 & 0.000000 & 0.000000 & 1.000000
  \end{pmatrix}
  \]
A. Astronomy-Related

○ From Galactic to Super-Galactic:

\[
\begin{pmatrix}
-0.735742574804 & 0.677261296414 & 0.000000000000 & 0.0 \\
-0.074553778365 & -0.080991471307 & 0.903922590400 & 0.0 \\
0.673145302109 & 0.731271165817 & 0.110081262225 & 0.0 \\
0.0 & 0.0 & 0.0 & 1.0
\end{pmatrix}
\]

○ From Super-Galactic to Galactic:

\[
\begin{pmatrix}
-0.735742574804 & -0.074553778365 & 0.673145302109 & 0.0 \\
0.677261296414 & -0.080991471307 & 0.731271165817 & 0.0 \\
0.000000000000 & 0.993922590400 & 0.110081262225 & 0.0 \\
0.0 & 0.0 & 0.0 & 1.0
\end{pmatrix}
\]

Note that these matrices can be directly used in OpenGL to transform a spatial coordinate by left-multiplication (see OpenGL commands \texttt{glMultMatrixd} and \texttt{glLoadIdentity}); thus, these matrices are presented in column-major order. For example, the following code fragment implements the transformation from Super-Galactic to Galactic coordinates:

```c
#define SGAL_TO_GAL_MAT
-0.735742574804, -0.074553778365, 0.673145302109, 0.0, \
0.677261296414, -0.080991471307, 0.731271165817, 0.0, \
0.000000000000, 0.993922590400, 0.110081262225, 0.0, \
0.0, 0.0, 0.0, 1.0

double convertMat[16] = {
    GAL_TO_GAL_MAT
};

/* note: OpenGL takes transpose */
glPushMatrix();
glMultMatrixd(convertMat); 
    draw_something_in_supergalactic_coordinates(); 
glPopMatrix();
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Thus, if your virtual environment (corresponding to the current Modelview matrix) is in the Galactic coordinate system and you want to draw something in Super-Galactic coordinates, you can pick up the matrix corresponding to "From Galactic to Super-Galactic" and carry out the transformation as shown above (see the manpage of `glMultMatrixd` for details).

It is worth noting that nothing is at rest in the Universe; because of the precession of the equinoxes (with a 26,000-year period), though the difference in matrix values may not be very big, these matrices are subject to change every fifty years (to be updated when the 2050 Almanac standard is released). Note that the matrices given here correspond to the year 2000, the so-called J2000 coordinates.
A.5 Time and Calendar

This section summarizes time and calendar recording in the literature of astronomy.

- **Sidereal (means "by the stars")**
  A time-keeping system setup by astronomers relative to stars.

- **Meridian**
  Each location on Earth has a meridian, which is an imaginary great circle on Earth's surface linking the North and South Poles through the zenith (overhead) of the location.

- **Solar day (or Apparent Solar day)**
  Length of time that the Sun takes to return to the Meridian.

- **Sidereal day**
  Length of time that a star takes to return to the Meridian.

![Diagram showing the difference between Solar and Sidereal days](image)

Figure A.6: The Sidereal day is slightly shorter than the Solar day.
Note:
Because the Earth exhibits two simultaneous motions, a rotation about its own axis and an orbital motion about the Sun, the sidereal time is slightly different from the Solar time. As depicted in Figure A.6, since the Earth orbits the Sun, the stars return to the meridian earlier than the Sun; the sidereal day is actually shorter than the Solar day by 3 minutes 56 seconds.

- Mean Solar day (a day in daily life)
  Since the Earth is not rotating at uniform speed, the minimum and maximum Solar day can differ by as much as a minute. The average length of a Solar day is called the mean Solar day, which is the standard 24 hours in our daily life.

- Sidereal clock and Solar clock
  Both clocks have a 24-hour basis and they show exactly the same time only at one instant every year; that is the autumnal equinox. In the next day after the autumnal equinox, the sidereal clock will be \( \sim \) 4 minutes ahead of the Solar clock. Since 4 minutes \( \times 365 \approx 24 \) hours, the two clocks match again one year later. Also, it is interesting to note that since the sidereal clock is related to the star motion, the same star rises about 4 minutes earlier (in our Solar time) each day.

- Sidereal year
  Length of time that the Sun takes to return to a given position relative to the background stars.

- Solar year (or tropical year)
  Length of time that the Sun takes to return to the vernal equinox after one cycle around the Ecliptic. Note that the vernal equinox is one of the intersections between the Celestial Equator and the Ecliptic, marking the first day of Spring.
Note:
The Solar year is \( \sim 365.242 \) mean Solar days, while sidereal year is \( \sim 365.256 \) mean Solar days. The Sidereal year is slightly larger than Solar year due to the precession of the vernal equinox. Note that \( 365.256 - 365.242 \approx 1/26000 \), so 26,000 years is the precession period of the Earth.

- **Julian Calendar**
  Established by Julius Caesar in 46 B.C., this calendar has years that are normally 365 days in length, with an extra day inserted every fourth year to account for the \( 1/4 \) extra days accumulated every year.

- **Gregorian Calendar**
  This calendar is the standard calendar we currently use; it was introduced by Pope Gregory XIII in 1582 to replace the Julian calendar. The reason for the change was that the accumulated error of the Julian calendar over one and a half thousand years had resulted in the seasons being about 10 days out of phase. Thus, 10 days in 1582 were omitted: the 4th of October in the old system became the 15th of October in the new system. Britain and some parts of the world were not reformed until September 1752.

  Years that are evenly divisible by 4 are leap years, except that three out of four century years are not leap years. Thus, 1600 and 2000 are leap years, but not 1700, 1800, and 1900.

  Note that the average number of days in a year for this system is 365.2425 days (97 out of 400 years are leap years), which is closer to the true value (Solar year) than that of the old Julian Calendar.

- **Julian Date (JD)**
  This is a standard system for recording events on different calendars — because of its...
name, people usually get confused with the Julian Calendar. However, it is important to note that it is not related to the Julian Calendar. It sets noon on January 1st, 4714 B.C. at Greenwich as the starting time and measures the number of days since that moment.

- Julian Century (T)
  This counts the number of centuries after the J2000 epoch:
  \[ T = \frac{JD - 2415020}{36525} \]

- Greenwich Mean Time (GMT)
  It is the local time at the 0° meridian (the prime meridian), which passes through Greenwich, England; it is the same everywhere on Earth.

- Universal Time Coordinate (UTC)
  It is the mean Solar time along the Earth's prime meridian. Strictly speaking, it is a little bit different from the GMT because the apparent Solar day can vary but not the mean Solar day. However, UTC is widely regarded to be the same as GMT.

- Standard Computer Time
  The time in seconds since the Epoch (00:00:00 UTC, January 1, 1970).

Here is the pseudo code to convert from Gregorian Calendar to Julian date:

```
INPUT: year (Y), month (M), day (D),
       hour (H), min (N), seconds (S).

// Adjust year & month values (if necessary)
IF M < 3 THEN
   M = M + 12 ;
   Y = Y - 1 ;
END
A = 2 - FLOOR ( Y/100 ) + FLOOR ( Y/400 ) ;
```
// Compute the day fraction
D = D + (H / 24.0) + (N / 1440.0) + (S / 86400.0);

// Compute the Julian date
JD = FLOOR (365.25 * Y)
+ FLOOR (30.6001 * (M+1))
+ D
+ A;
+ 1720994.5;

RETURN JD;

To convert the Julian date to the Gregorian Calendar, we need a more complicated algorithm:

INPUT: julian date (JD).  // JD > 0

I = FLOOR (JD + 0.5);
P = (JD + 0.5) - 1;  // fractional part of JD+0.5

IF I < 2299161 THEN
    A = I;
ELSE
    X = FLOOR ((I - 1867216.25) / 36524.25);
    A = I + 1 + X - FLOOR (X / 4);
END

B = A + 1524;
C = FLOOR ((B - 122.1) / 365.25);
D = FLOOR (365.25 * C);
E = FLOOR ((B-D) / 30.6001);

day = B - D - FLOOR (30.6001 * E) + P;

IF E < 13.5 THEN
    month = E - 1;
ELSE
    month = E - 13;
END

IF month > 2.5 THEN
    year = C - 4716;
ELSE
year = C - 4715 ;
END

RETURN (day, month, year) ;

The pseudo-code here is modified from [102]. Note that FLOOR takes the integer part of a floating point number and the pseudo-code for converting Julian date to Gregorian Calendar works for (positive) JD.

A.6 Powers of Ten Tables

The following two tables list objects and phenomena in terms of a powers of ten scales in space and time, respectively.
Table A.17: Base 10 logarithms of scales of typical objects in the physical Universe in units of meters.

<table>
<thead>
<tr>
<th>Typical objects</th>
<th>Powers of 10 (meters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known Universe (Quasars, etc)</td>
<td>27</td>
</tr>
<tr>
<td>Super-cluster</td>
<td>25</td>
</tr>
<tr>
<td>Clusters of galaxies</td>
<td>24</td>
</tr>
<tr>
<td>Within Virgo cluster</td>
<td>23</td>
</tr>
<tr>
<td>Distance to Andromeda galaxy</td>
<td>22</td>
</tr>
<tr>
<td>Milky Way diameter</td>
<td>21</td>
</tr>
<tr>
<td>Spiral arm size</td>
<td>20</td>
</tr>
<tr>
<td>Distance to Orion arm</td>
<td>19</td>
</tr>
<tr>
<td>Stars within 50 light years</td>
<td>18</td>
</tr>
<tr>
<td>Distance to nearest stars</td>
<td>17</td>
</tr>
<tr>
<td>The Oort cloud boundary</td>
<td>16</td>
</tr>
<tr>
<td>Within the Oort cloud</td>
<td>15</td>
</tr>
<tr>
<td>Light week</td>
<td>14</td>
</tr>
<tr>
<td>Size of the Solar system</td>
<td>13</td>
</tr>
<tr>
<td>Orbit of Jupiter</td>
<td>12</td>
</tr>
<tr>
<td>Venus, Earth, and Mars</td>
<td>11</td>
</tr>
<tr>
<td>Distance the Earth travels in 4 days</td>
<td>10</td>
</tr>
<tr>
<td>Earth-Moon distance</td>
<td>9</td>
</tr>
<tr>
<td>Light second in vacuum</td>
<td>8</td>
</tr>
<tr>
<td>Earth diameter</td>
<td>7</td>
</tr>
<tr>
<td>California size</td>
<td>6</td>
</tr>
<tr>
<td>San Francisco Bay Area</td>
<td>5</td>
</tr>
<tr>
<td>San Francisco</td>
<td>4</td>
</tr>
<tr>
<td>A Park</td>
<td>3</td>
</tr>
<tr>
<td>A Building</td>
<td>2</td>
</tr>
<tr>
<td>A Car</td>
<td>1</td>
</tr>
<tr>
<td>A Man</td>
<td>0</td>
</tr>
<tr>
<td>Hand</td>
<td>-1</td>
</tr>
<tr>
<td>Skin</td>
<td>-2</td>
</tr>
<tr>
<td>A Pore in the Skin</td>
<td>-3</td>
</tr>
<tr>
<td>Micro-Organisms / Hair Thickness</td>
<td>-4</td>
</tr>
<tr>
<td>Size of a red blood cell</td>
<td>-5</td>
</tr>
<tr>
<td>The Nucleus of a Cell</td>
<td>-6</td>
</tr>
<tr>
<td>Virus size / Visible light wavelength</td>
<td>-7</td>
</tr>
<tr>
<td>DNA Structure</td>
<td>-8</td>
</tr>
<tr>
<td>DNA Molecules</td>
<td>-9</td>
</tr>
<tr>
<td>Atom size</td>
<td>-10</td>
</tr>
<tr>
<td>Inner electron cloud</td>
<td>-11</td>
</tr>
<tr>
<td>Within electron cloud</td>
<td>-12</td>
</tr>
<tr>
<td>Uranium Nucleus</td>
<td>-13</td>
</tr>
<tr>
<td>Carbon Nucleus</td>
<td>-14</td>
</tr>
<tr>
<td>A Proton</td>
<td>-15</td>
</tr>
<tr>
<td>Quarks</td>
<td>-16</td>
</tr>
<tr>
<td>Electron size</td>
<td>-18</td>
</tr>
<tr>
<td>Planck length</td>
<td>-35</td>
</tr>
</tbody>
</table>
Table A.18: Base 10 logarithms of scales of typical phenomena in units of seconds (from http://www.powersof10.com/powers/time/time.html).

<table>
<thead>
<tr>
<th>Typical Phenomena</th>
<th>Powers of 10 (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age of the COBE/WMAP map &quot;First Light&quot;</td>
<td>18</td>
</tr>
<tr>
<td>Age of Earth and Moon</td>
<td>17</td>
</tr>
<tr>
<td>Super-continent Pangaea</td>
<td>16</td>
</tr>
<tr>
<td>Oligocene epoch</td>
<td>15</td>
</tr>
<tr>
<td>A photon leaving the Sun’s center to reach the surface</td>
<td>14</td>
</tr>
<tr>
<td>Orbiting period of our Solar system around the Milky Way</td>
<td>14</td>
</tr>
<tr>
<td>Period at which ice ages come and go</td>
<td>13</td>
</tr>
<tr>
<td>The Earth precesses its axis every 25,800 years</td>
<td>12</td>
</tr>
<tr>
<td>About the time the Chinese calendar began years ago</td>
<td>11</td>
</tr>
<tr>
<td>Three centuries (back to Newton’s time)</td>
<td>10</td>
</tr>
<tr>
<td>Scale of human generation</td>
<td>9</td>
</tr>
<tr>
<td>Scale of the human learning curve (~ 3 years)</td>
<td>8</td>
</tr>
<tr>
<td>About one season</td>
<td>7</td>
</tr>
<tr>
<td>Two Earth weeks</td>
<td>6</td>
</tr>
<tr>
<td>Earth day (86,400 sec.)</td>
<td>5</td>
</tr>
<tr>
<td>Time unit in Ancient Chinese (2 hours)</td>
<td>4</td>
</tr>
<tr>
<td>Daily activities such as local walks, cleaning, etc.</td>
<td>3</td>
</tr>
<tr>
<td>A Gothic chamber clock, Germany (ca.1485)</td>
<td>2</td>
</tr>
<tr>
<td>A jet plane to travel 1.5 miles</td>
<td>1</td>
</tr>
<tr>
<td>Fundamental unit / Heartbeat</td>
<td>0</td>
</tr>
<tr>
<td>A beam of light to travel the circumference of the Earth</td>
<td>-1</td>
</tr>
<tr>
<td>Exposure for undistorted images of runners in Olympics</td>
<td>-2</td>
</tr>
<tr>
<td>Firing rate of nerves in humans and animals</td>
<td>-3</td>
</tr>
<tr>
<td>Sound waves within human hearing (200Hz-20000Hz)</td>
<td>-4</td>
</tr>
<tr>
<td>Rate at which enzymes break down in muscle cells</td>
<td>-5</td>
</tr>
<tr>
<td>General stroboscopic photography</td>
<td>-6</td>
</tr>
<tr>
<td>Exposure of the fastest stroboscopic camera</td>
<td>-7</td>
</tr>
<tr>
<td>A beam of light to travel across a boat</td>
<td>-8</td>
</tr>
<tr>
<td>Speed at which computers calculate</td>
<td>-9</td>
</tr>
<tr>
<td>A beam of light to travel across a piece of chalk</td>
<td>-10</td>
</tr>
<tr>
<td>A beam of light to travel across a blade of grass</td>
<td>-11</td>
</tr>
<tr>
<td>H-bond lifetime between water molecules in 25°C water</td>
<td>-12</td>
</tr>
<tr>
<td>Detail of general atomic clocks, NIST-7</td>
<td>-13</td>
</tr>
<tr>
<td>Detail of the world’s most accurate clock, NIST-7</td>
<td>-14</td>
</tr>
<tr>
<td>Individual atom vibrational frequency</td>
<td>-15</td>
</tr>
<tr>
<td>An electron to travel the surface of an atom</td>
<td>-16</td>
</tr>
<tr>
<td>Attosecond</td>
<td>-18</td>
</tr>
</tbody>
</table>
Vertex Programs for PHC Transformation

B.1 Vertex Program Overview

Vertex programming offers programmability in the transformation and lighting. Pro­
grammers can write vertex programs that execute on the graphics processing unit (GPU) to alleviate the workload on the CPU. The vertex programs we wrote for the generalized PHC transformation (see Chapter 5) adopt the standard vertex program version 1.0 (see the table below). Note that each variable below is a four components vectors $(x\ y\ z\ w)$.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Function</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARL</td>
<td>Address Register Load</td>
<td>ARL A0.x, src0.C ;</td>
</tr>
<tr>
<td></td>
<td>for Relative addressing</td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>Move values from source to dest.</td>
<td>MOV dest, src0 ;</td>
</tr>
<tr>
<td>MUL</td>
<td>Multiply (component-wise)</td>
<td>MUL dest, src0, src1 ;</td>
</tr>
<tr>
<td>ADD</td>
<td>Addition (component-wise)</td>
<td>ADD dest, src0, src1 ;</td>
</tr>
<tr>
<td>MAD</td>
<td>Multiply $(src_0 \times src_1)$ and Add $(+src_2)$ (component-wise)</td>
<td>MAD dest, src0, src1, src2 ;</td>
</tr>
<tr>
<td>RCP</td>
<td>Reciprocal of one component in src0</td>
<td>RCP dest, src0.C ;</td>
</tr>
<tr>
<td>RSQ</td>
<td>Reciprocal Square Root of the absolute value of the source</td>
<td>RSQ dest, src0.C ;</td>
</tr>
<tr>
<td>Instruction</td>
<td>Function</td>
<td>Syntax</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>DP3</td>
<td>Three-Component Dot Product</td>
<td>DP3 dest, src0, src1 ;</td>
</tr>
<tr>
<td>DP4</td>
<td>Four-Component Dot Product</td>
<td>DP4 dest, src0, src1 ;</td>
</tr>
<tr>
<td>DST</td>
<td>Compute Distance Vector (attenuation)</td>
<td>DST dest, src0.C1, src1.C2 ;</td>
</tr>
</tbody>
</table>

Assumes: src0.C1=d² and src1.C2=1/d
dest.x = 1, dest.y = d,
dest.z = d², dest.w = 1/d,

| MIN         | Minimum (component-wise) | MIN dest, src0, src1 ; |
| MAX         | Maximum (component-wise) | MAX dest, src0, src1 ; |
| SLT         | Set On Less Than (component-wise) | SLT dest, src0, src1 ; |
| SGE         | Set On Greater Than or Equal Than (component-wise) | SGE dest, src0, src1 ; |

dest.Cx = 1.0 if true else 0.0

dest.Cx = 1.0 if true else 0.0

<table>
<thead>
<tr>
<th>EXP</th>
<th>Exponential Base 2</th>
<th>EXP dest, src0.C ;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dest.x = (2^{\text{floor}(\text{src0.C})})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dest.y = (\text{src0.C} \cdot \text{floor}(\text{src0.C}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dest.z = (2^{(\text{src0.C})}) and dest.w = 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOG</th>
<th>Logarithm Base 2</th>
<th>LOG dest, src0.C ;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dest.x = (\text{Exponent}(\text{src0.C})) in ([-126, 127])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dest.y = (\text{Mantissa}(\text{src0.C})) in ([1, 2))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dest.z (\sim) (\log_2(</td>
<td>\text{src0.C}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LIT</th>
<th>Compute Light Coefficients</th>
<th>LIT dest, src0 ;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Assumes: src0.xyz=(N \cdot L, N \cdot H, shininess)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dest.x = 1.0 (ambient)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dest.y = (\text{CLAMP}(\text{src0.x}, 0, 1)) (diffuse)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dest.z = (specular), dest.w = 1.0</td>
<td></td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
B. Vertex Programs for PHC Transformation

For details about vertex programming, readers are referred to the SDK from www.nvidia.com or reference books [66, 52].

B.2 Vertex Program Example: \( \text{vp.nicest_cutoff.vp} \)

(\text{Nicest + Cutoff + Perspective Projection})

```cpp
// VP1.0

# Vertex Program [ NICEST + CUTOFF ]
#
# Copyright 2002, Trustees of Indiana University
#
# Just like calling these functions in glphc_noshader.cpp
# -> glPHC_Vertex_NOSHADER_SAFETY_NICEST
# -> glPHC_Normalize
# -> glPHC_FinalProject_cutoff
#
# ----------------------------------------------
#
# NOTE:
# We have c[0] ... c[9] and R0 ... R15
#
# ----------------------------------------------
#
# 1) Tracking Matrices
#
# c[0]...c[3] contains the projection matrix
# c[4]...c[7] contains the modelview matrix
# c[8]...c[11] contains the transpose of the projection matrix
# c[12]...c[15] contains the inverse transpose of the modelview matrix
#
# ----------------------------------------------
#
# 2) Tracking constants
#
# c[16] \quad x = \frac{\ln10}{\ln2} = \log2(10.0)
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
B. Vertex Programs for PHC Transformation

```
# y = log2 / log10 log10(2.0)
# x = 1.0
# w = 0.0
#
# c[17] x = FLT_MIN
# y = FLT_MAX
# z = -le-5
# w = 7.0
#
# 3) Tracking variables
#
# c[20] translational vector (__tran)
# (depends on safety hint)
#
# c[21] x = __log2SumNear_Cutoff
# y = __log2SumFar_Cutoff
# z = __log2Near
# w = __log2Far
#
# c[22] x = __mvLogScale
# y = __zFar
# z = __zNearRatio
# w = __zFarRatio
# (depends on __enableLinearRange)
# c[23] x = __tmpZNearBase x = __tmpZLogBase
# y = __tmpZCutoffBase OR y = __tmpZLogBase
# z = __tmpZFarBase z = __tmpZLogBase
# w = __tmpZFarBase w = __tmpZLogBase
#
# (depends on __enableLinearRange)
# c[24] x = __tmpZNearRatio x = __tmpZLogRatio
# y = __tmpZCutoffRatio OR y = __tmpZLogRatio
# z = __tmpZFarRatio z = __tmpZLogRatio
# w = __tmpZFarRatio w = __tmpZLogRatio
#
# c[30] material color
# c[31] light position (eye-space)
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
B. Vertex Programs for PHC Transformation

---

4) Input Information needed for this vertex program
v[0POS] contains the per-vertex position
---

5) Output Information of this vertex program
o[HPOS] output register for homogeneous position
---

By Philip Fu
Thu May 23 11:24 EST 2002

Note:

1) \(10^x = 2^y \) if \( y = x \) (Lnl0/Ln2)
MUL R1, RO.w, c[16].x;
EXP R1, R1.w;

2) \( \log_{10}(x) = \log_2(y) \rightarrow \log_2(y) = \log_2(x) \times \log_{10}(2.0) \rightarrow R1.z \)
LOG R1, RO.w;
MUL R1.z, R1.w, c[16].y;

3) how to: absolute values
MAX R1, RO, -RO;

4) Branching
# compute \( r0 = (r1 \geq r2) ? r3 : r4 \)
# SGE r0, r1, r2; # one if (r1 \geq r2) holds, zero otherwise
# ADD r1, r3, -r4;
# MAD r0, r0, r1, r4; # \( r0 = r0 \times (r3 - r4) + r4 = r0 \times r3 + (1 - r0) \times r4 \)

---

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
B. Vertex Programs for PHC Transformation

---

# Pre-store constants in registers because
# *** any given instruction may only access only a single constant register!!!

# (1) glPHC_Vertex_NOSHADER_SAFETY_NICEST
#
# -> output result to "R0" for the next phase
# such that R0 = { unit_x[0], unit_x[1], unit_x[2], v[OPOS].w };

---

# 1a) Transform to eye space and find the length

DP3 R1.x, c[4], v[OPOS];
DP3 R1.y, c[5], v[OPOS];
DP3 R1.z, c[6], v[OPOS];

DP3 R2.x, R1, R1;  # R2.x = length^2
RSQ R2.w, R2.x;    # R2.w = 1/length
MIN R2.w, R2.w, c[17].y;  # trick to avoid infinity to propagate

# two cases
SGE R5.x, R2.x, c[17].x;  # R5.x = length > 0.0
SLT R5.y, R2.x, c[17].x;  # R5.y = length == 0.0

# 1b) R1 = [a, a, a]

# R1.xyz = mult3D(a, 1/length);
MUL R1.xyz, R1, R2.w;

# R2.x = -log10(1/length) + v[3]
B. Vertex Programs for PHC Transformation

LOG R2, R2.w;
MAD R2.x, -R2.z, c[16].y, v[6DO$.w;)

# R2.y = -FLT_MAX * (length == 0.0)
MUL R2.y, R5.y, -c[17].y;

# R1.w = R2.y + (length > 0.0) * R2.x
MAD R1.w, R5.x, R2.x, R2.y;

# 1c) R2.w = ds = tran[3] - s_a;
# R2.x = ( tran[3] < s_a) => da < 0
# R2.y = ( tran[3] >= s_a) => ds >= 0
# R3.z = leng = pow(10.0,ds);
# R4.z = 1.0/leng

ADD R2.w, c[20].w, -R1.w;
SLT R2.w, c[20].w, R1.w;
SGE R2.y, c[20].w, R1.w;

MUL R2.w, R2.w, c[16].x;
EXP R3, R2.w;
MIN R3.z, R3.z, c[17].y;  # trick to avoid infinity to propagate
RCP R4, R3.z;

# le) CASE 2: | a | >= | t | (ds < 0.0) <-> R2.x is true
# - R4 = a
#
# a[0] = a[0] / leng + __tran[0];
# a[1] = a[1] / leng + __tran[1];

MAD R4, R2, R4.z, c[20];
MOV R4.w, c[20].w;

# le) CASE 2: | a | >= | t | (ds < 0.0) <-> R2.x is true
B. Vertex Programs for PHC Transformation

# - Rl = (a, a, a)
#
# a[0] += _tran[0] * leng;
# a[1] += _tran[1] * leng;

MAD Rl.xyz, c[20], R3.z, Rl;
MIN Rl.xyz, Rl, c[17].y;  # trick to avoid infinity to propagate

# 26 instructions

# (2) glPHC_Normalize
# -> input: "R0"
# -> output: "R0" for the next phase

# 2a) Rl = fabs(unit_x)

MAX Rl, R0, -R0;  # Rl = fabs(R0)

# 2b) R2.w = |R1| > 0.0 AND R2.z = |R1.z| > 0.0

DP3 Rl.w, R1, R1;  # R1.w = |R1|^2
SGE R2, R1, c[17].x;  # R2.z = |R1.z| > 0.0
B. Vertex Programs for PHC Transformation

# R2.w = |R1| > 0.0 (not a zero vector)

# 2c) R2.xyz is the selector on R1.xyz

# Fix R3.xy

ADD R2.xy, R2.w, -R2.z ; # R2.xy = |R1.z| == 0.0 & & |R1| > 0.0
SUB R3.x, R1.x, R1.y ; # R3.x = |R1.x| == |R1.y|
SLT R3.y, R1.x, R1.y ; # R3.y = |R1.x| < |R1.y|
MUL R2.xy, R2, R3 ; # R2.x = |R1.z| == 0.0
# & & |R1| > 0.0 & & |R1.x| > 0.0
# R2.y = |R1.z| == 0.0
# & & |R1| > 0.0 & & |R1.y| > 0.0

# 2d) R1.x = ratio and R0.xyz = unit_x[] *= ratio

DEF R3.x, R3, R1 ; # R3.x = selected leng
DCP R3, R1.x ; # R1.x = 1/leng
MUL R0.xyz, R3, R3.z ; # R0.xyz = unit_x[] * ratio
# (or zero if it is initially a zero vector)

### If R2.xyz are all zeros, is it ok?
### it is ok because R0.xyz is a zero vector iff R2.xyz are all zeros
### (assume zero * whatever = zero)

# 2e) s_x = R0.w = R2.w*R0.w + (1.0-R2.w)*(-FLT_MAX) = R2.w*R0.w + R2.y*(-FLT_MAX)

SLT R2.y, R1.w, c[17].x ; # R2.y = 1.0 - R2.w OR |R1|^2 == 0.0
MUL R0.w, R2.w, R0.w ;
MAD R0.w, R2.y, -c[17].y, R0.w ;

# 2f) s_x = log10(ratio)
# R0.w = log10(R3.z)

LOG R4, R3.z ;
MUL R4.z, R4.w, c[16].y ; # R4.z = log10(R3.z)
B. Vertex Programs for PHC Transformation

MAD R0.w, R2.w, -R4.z, R0.w ; # multiply with R2.w for safety as R2.w makes
# sure R1 is a non-zero vector

# 16 instructions

# (3) glPHC_Fin alProject_cutoff
#
# -> input: "RO"
# => output: "O [HPOS]
# "RO" is the eyespace vertex (multiply with zEye)

#### 3a) R1 - Selector (which region)
# R1.x = (s_x < _logZSumNear_Cutoff)
# R1.y = (s_x < _logZSumFar_Cutoff) and (_logZSumNear_Cutoff <= s_x)
# R1.z = (s_x < _logZFar ) and ( _logZSumFar_Cutoff <= s_x)
# R1.w = (s_x >= _logZFar )

# R3.x = R0.w - __logZSumNear_Cutoff
ADD R3.x, R0.w, -c[21].x ;

# R3.z = pow(10.0, R3.x)
B. Vertex Programs for PHC Transformation

MUL R3.x, R3.x, c[26].x ;
EXP R3, R3.x ;

# R2.x = R3.x * R1.y + R0.w * (1-R1.y) = (R3.z-R0.w)*R1.y + R0.w
ADD R2.x, R3.z, -R0.w ;
MAD R2.x, R2.x, R1.y, R0.w ;

# 3c) Compute R4.x = zNDC
MOV R5, c[24] ; # put ratio in R5
MAD R4, R5, R2.x, c[23] ; # base + s*x*ratio
DP4 R4.x, R4, R1 ; # select from the 4 cases

# 3d) Find R3.x = zEye and R0.xyz *= zEye

ADD R3.x, R4.x, c[2].z ;
MUL R3.x, R3.x, c[2].w ;

# R3.y = zEye = zNDC * __zFar (combine with the next instruction)
###MUL R3.y, R4.x, c[22].y ;

# R3.x = R3.x*(1.0-R1.w) + R3.y*R1.w - (R3.y-R2.x) * R1.w + R2.x
MAD R3.y, R4.x, c[22].y, -R3.x ; # R3.y = (R3.x-R3.y)
MAD R3.x, R3.y, R1.w, R3.x ;

# 3e) Projection -> o[HPOS] with zEye in R3.x

DP1 R1.x, R0, c[0] ;
DP1 R1.y, R0, c[1] ;
DP1 R1.z, R0, c[2] ;
DP3 R1.w, R0, c[3] ;
MAD c[HPOS], R1, R3.x, c[11] ;
B. Vertex Programs for PHC Transformation

# 22 instructions

# (4) Texture and color
#
# -> input: "RO" is the eyespace vertex (result from (3))

# 4a) Texture coordinate

MOV o[TEX0], v[TEX0];

# 4b) Transform the normal to eye-space.
# To do this, transform it by the inverse transpose of the MODELVIEW matrix.

DP4 R2.X, c[12], v[NORMAL];
DP4 R2.Y, c[13], v[NORMAL];
DP4 R2.Z, c[14], v[NORMAL];

# 4c) Compute normalized L = light position - vertex position.

ADD R1, c[31], -RO;
DSQ R1.W, R1, R1;
MUL R1.xyz, R1, R1.w;

# 4d) Compute N dot L and clamp the result.

DP3 R3, R1, R2;
MAX R3, R3, R10.w;
B. Vertex Programs for PHC Transformation

---

# 4e) Modulate the material color by NdotL.

MUL  o[COL0], t[30], r1 ,

# 11 instructions

---

B.3 Vertex Program Example: (vp.nicest.nocutoff.parallel.vp)

(Fastest + NoCutoff + Parallel Projection)

---

# (1) glPHC_Vertex_NOSHADER_SAFETY_FASTEST

# -> output result to "R0" for the next phase

# such that R0 = ( unit_x[0], unit_x[1], unit_x[2], v[OPOS].w );

---

# 1a) Find the pow(10, _mLogScale-v[3]) -> R1.z
B. Vertex Programs for PHC Transformation

ADD R1.x, c[22].x, -v[OPOS].w;
MUL R1, R1.x, c[16].x;
ZFP R1, R1.w;

# 1b) Find the eye coordinate -> RO.xyz
#
# unit_x[0] = dot3D (_tran_mvmat, v) + _tran[0] * pow10_ds
# unit_x[1] = dot3D (_tran_mvmat+c, v) + _tran[1] * pow10_ds

DP3 RO.x, v[OPOS], c[4];
DP3 RO.y, v[OPOS], c[5];
DP3 RO.z, v[OPOS], c[6];
MAD RO, c[201], R1.z, RO;
MOV RO.w, v[OPOS].w;

# 8 instructions

# (2) glPHC_Normalize
#
# -> input: "RO"
# -> output: "RO" for the next phase

# 2a) R1 = fabs(unit_x)

MAX R1, RO, -RO;  # R1 = labs(RO)

# 2b) R2.w = |R2| > 0.0 AND R2.z = |R1.z| > 0.0

DP3 R1.w, R1, R1;  # R1.w = |R1|^2
SGE R2, R1, c[17].x;  # R2.z = |R2.z| > 0.0
# R2.w = |Rl| > 0.0 (not a zero vector)

## 2c) \( R2.xyz \) is the selector on \( R1.xyz \)

* Fix \( R2.xy \)

\[
\text{ADD} \ R2.xy, \ R2.w, \ -R2.z ;
\]

\[
\text{SGE} \ R3.x, \ R1.x, \ R1.y ;
\]

\[
\text{SLT} \ R3.y, \ R1.x, \ R1.y ;
\]

\[
\text{MUL} \ R2.xy, \ R2, \ R3 ;
\]

## 2d) \( R3.z \) = ratio and \( R0.xyz = \text{unit}_x() \* \text{ratio} \)

\[
\text{DP} \ R3.x, \ R2, \ R1 ;
\]

\[
\text{RCP} \ R3, \ R3.z ;
\]

\[
\text{MUL} \ R0.xyz, \ R0, \ R3.z ;
\]

## 2e) \( s_x = R0.w - R2.w*R0.w + (1.0-R2.w)*(-\text{FLT.MAX}) + R2.w*R0.w + R2.y*(-\text{FLT.MAX}) \)

\[
\text{SLT} \ R2.y, \ R1.w, \ c[17].x ;
\]

\[
\text{MUL} \ R0.w, \ R2.w, \ R3.x ;
\]

## 2f) \( s_x -= \log_{10}(\text{ratio}) \)

\[
\text{LOG} \ R4, \ R3.z ;
\]

\[
\text{MUL} \ R4.z, \ R4.z, \ c[16].y ;
\]

### If \( R2.xyz \) are all zeros, is it ok?
### it is ok because \( R0.xyz \) is a zero vector iff \( R2.xyz \) are all zeros
### (assume zero = whatever = zero)

### 2e) \( s_x -= \log_{10}(\text{ratio}) \)

\[
\text{LOG} \ R4, \ R3.z ;
\]

\[
\text{MUL} \ R4.z, \ R4.z, \ c[16].y ;
\]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
B. Vertex Programs for PHC Transformation

\#ADD R0.w, R3.w, -R4.z ;
\#ADD R0.w, R1.w, -R4.z, R0.w ;  \# multiply with R2.w for safety as R2.w makes
\# sure R1 is a non-zero vector

\# IS instructions
\#
\#
\# (3) glPHC_FinalProject_nocutoff
\#
\# -> input: "RO"
\# -> output: "OHPOS1"
\#
\# 3a) Compute R5.x = zNDC:
\#
\# R1 is the selector
\# R1.x = (R0.w < \_logZNear) & R0.z <= 0.0
\# R1.y = (R0.w >= \_logZNear) & R0.z <= 0.0
\# R1.z = (R0.w < \_logZNear) & R0.z > 0.0
\# R1.w = (R0.w >= \_logZNear) & R0.z > 0.0

\# compute
\# R2.x = \_zLim * powf(2.0f, (float)(s_x-_logZNear)) * \_stim_one;
\# R2.y = s_x * \_tmpZLogRatio + \_tmpZLogBase;
\# R2.z = \_zLim_one_twice;
\# R2.w = \_zLim_one_twice

\#ADD R2.x, R3.w, -c[21].z ;  \# R2.x = s_x-_logZNear
\#EXP R3, R2.x ;  \# R3.x = 2^R2.x
\#ADD R2.x, R3.z, c[25].y, -c[25].z ;  \# R2.x = \_zLim +
B. Vertex Programs for PHC Transformation

```plaintext
# powf(2.0f, (float)(s_x - logZNear))
# - __zLim_one

MAD R2.y, R0.w, c[23].y, c[23].x ; # R2.y = s_x * __tmpZLogRatio
# + __tmpZLogBase

ADD R2.xw, -R2.xyxy, -c[25].w ;

# R5.x = zNDC (outVertex[2] = zNDC)
DP4 R5.x, R1, R2 ;

# 3b) Compute R1.x = zEye
# Update R0.w
# - if (s_x >= __parallel_limit)
#   a_x = __parallel_limit
SGE R2.x, R0.w, c[25].x ; # R2.x = s_x >= __parallel_limit)
ADD R2.y, -R0.w, c[25].x ; # R2.y = __parallel_limit - s_x
MAD R0.w, R2.y, R2.x, R0.w ; # R0.w = __parallel_limit - s_x)
#
#
#
# R1.x = powf(2.0f, (float)s_x)
MUL R1, R0.w, c[16].x ;
EXP R1, R1.w ;

# 3c) Projection
# w = dot3D (unit_x, __tran_projmat[12]) * zEye + __tran_projmat[15]
# outVertex[0] = (dot3D (unit_x, __tran_projmat12) * zEye + __tran_projmat[12]) / w
# outVertex[1] = (dot3D (unit_x, __tran_projmat[15]) * zEye + __tran_projmat[15]) / w

# Dot products
DP3 R1.x, R3, c[0] ; # R1.x = dot3D (unit_x, __tran_projmat[0])
DP3 R1.y, R3, c[1] ; # R1.y = dot3D (unit_x, __tran_projmat[4])
DP3 R1.z, R0, c[3] ; # R1.z = dot3D (unit_x, __tran_projmat[12])
#
# R1.xyz = R1.xyz * zEye + __tran_projmat[3,7,15]
```
B. Vertex Programs for PHC Transformation

```
MAD R1, R1.xyw, R1.z, c[1].xyw;

# Fix R1.z
MUL R1.z, R5.x, R1.w;

# Put R1 into o[HPOS]
MOV o[HPOS], R1;

# 22 instructions

# (4) Texture and color
#
# -> input: "RO" is the eyespace vertex (result from (3))

# 4a) texture coordinate
MOV o[TEX0], v[TEX0];

# 4b) Transform the normal to eye-space.
# To do this, transform it by the inverse transpose of the MODELVIEW matrix.
DP4 R2.x, c[12], v[NRML];
DP4 R2.y, c[13], v[NRML];
DP4 R2.z, c[14], v[NRML];

# 4c) Compute normalized L = light position - vertex position.
ADD R1, c[31], -RO;
DP3 R1.w, R1, R1;
RSQ R1.w, R1.w;
MUL R1.xyz, R1, R1.w;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
B. Vertex Programs for PHC Transformation

# 4d) Compute N dot L and clamp the result.

DP3 R3, R1, R2 ;
MAX R3, R3, R10.w ;

# 4e) Modulate the material color by NdotL.

MUL c[20], c[30], R3 ;

# 11 instructions

END
Bibliography


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.


[33] Nicholas Copernicus. *De Revolutionibus (Concerning the Revolutions) / On the Revolutions of the Celestial Spheres.* 1543. Poland.


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.


[67] Priscilla Frisch, Andrew J. Hanson, Chi-Wing Fu (animator), and Eric A. Wernert (animator). The Sun's Heliosphere & Heliopause. Astronomy Picture of the Day, 2002 June 24.


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.


Curriculum Vitae

CHI-WING FU

Address: Rm.215 Lindley Hall, Indiana University,
Bloomington, IN 47405, U.S.A.

E-mail: cwfu@acm.org or cwfu@cs.indiana.edu

OBJECTIVE

An Academic Position in Research and Teaching relating to the areas of Computer Graphics and Visualization.

EDUCATION

Indiana University, Bloomington, IN 47405, United States,
supervisor: Prof. Andrew J. Hanson
title: A Visualization Framework for Large-Scale Virtual Astronomy

1997-1999 Master of Philosophy (1999) in Computer Science and Engineering,
The Chinese University of Hong Kong.
supervisor: Prof. Pheng-Ann Heng
title: Interactive Illumination and Navigation Control in an Image Based Environment

1994-1997 Bachelor of Science (1997) in Computer Science (1st class honor),
The Chinese University of Hong Kong.
RESEARCH EXPERIENCE

Research Assistant for Prof. Andrew J. Hanson (Fall 1999 – Fall 2003)

- Created two animations, “Cosmic Clock” (shown in Siggraph 2000 Electronic Theater) and “Solar Journey” (for the NASA project NAG5-8163 and the SGI Planetarium Project for the Beijing Planetarium) based on our scnearviewer animation support system and the PET (path editing tool) animation editing system.

- Distance editing tool for modeling interstellar gas cloud interfaces such as Loop I and the Local Bubble.

- An assortment of interactive modeling and rendering programs for different kinds of astronomical data, e.g., stars, galaxies, the Milky Way, the solar system, satellites, etc.

- Developed various techniques for large scale visualization in astronomical environments, e.g., the power homogeneous coordinate (PHC), the depth rescaling method, etc.

- Developed a cross-platform program for visualizing high-dimensional mathematical functions.

- A quaternion demo for a Siggraph tutorial.

- Derived and implemented new applications of the MDL (Minimum Description Length) criteria for computer graphics modeling.
Collaboration with SGI (supervised by Prof. Andrew J. Hanson (Fall 2002 – ))

- Supporting the development of an SGI’s planetarium project, “Digital Universe,” to be used in the upcoming digital planetarium in Beijing — modeling and rendering of various astronomical data, e.g., our star rendering technology, etc.

- Preparing a playback animation, “Solar Journey,” as a planetarium show for the digital planetarium project.

Collaboration with the Chinese University of Hong Kong (2000–2003)

- Various projects concerning image-based relighting (collaborated by Prof. Tien-Tsin Wong and Dr. Siu-Hang Or).

- Data compression using spherical wavelets (collaborated with Prof. Chi-Sing Leung and Prof. Tien-Tsin Wong).

- Working on the “Binary-Space-Partitioned Images” project (collaborated with Prof. Tien-Tsin Wong, Prof. Chi-Keung Tang, Wai-Shun Tong, and Prof. Andrew J. Hanson).

- Medical visualization (collaborated with Prof. Tien-Tsin Wong, Prof. Pheng-Ann Heng, and Ping-Fu Fung).

TEACHING EXPERIENCE

Associate Instructor for the graphics course at Indiana University (Fall 2000)

- Courses: (B481) Undergraduate Computer Graphics and (B581) Advanced (Graduate) Computer Graphics.
- includes giving weekly tutorials, homework and exam grading, preparing homework assignments, setting up course homepages and the homework submission system, answering questions in course newsgroup, etc.

Teaching Assistant in the Chinese University of Hong Kong (Fall 1997 – Spring 1999)


- Includes giving weekly tutorials, homework and exams grading, preparing homework assignments, answering questions in newsgroups, maintaining course homepages, etc.

PUBLICATIONS

Journal Publications


Book Chapters


**Conference Publications**


**Media Publications**

• Solar Journey (to be presented in the Beijing planetarium)
  Producer and Director: Andrew J. Hanson,
  Animator: Chi-Wing Fu and Eric Wemert,
  Chinese Translation: Chi-Wing Fu and YingGang Li,
  Astronomy Advisor: Priscilla Frisch,
  Post-Production: David Rust
  Preparing for the *SGI Planetarium project* (to be distributed as video for NASA as well).
  *My contribution included the development of the animation and scripting system in OpenGL using our large scale visualization techniques as well as the modeling and rendering methods for stars, galaxies, etc.*

• The Satellites that Surround Earth
  Philip Chi-Wing Fu, Andrew J. Hanson, Priscilla Frisch,
  *Astronomy Picture of the Day*, 2003 July 14.,
  (available: [http://antwrp.gsfc.nasa.gov/apod/ap030714.html](http://antwrp.gsfc.nasa.gov/apod/ap030714.html)).

• The Sun's Heliosphere & Heliopause,
  Priscilla Frisch, Andrew J. Hanson, Philip Chi-Wing Fu (Animator), and Eric Wemert (Animator),
  *Astronomy Picture of the Day*, 2002 June 24.,
  (available: [http://antwrp.gsfc.nasa.gov/apod/ap020624.html](http://antwrp.gsfc.nasa.gov/apod/ap020624.html)).
• Cosmic Clock

Producer and Director: Andrew J. Hanson,
Animator: Chi-Wing Fu,
Post-Production: David Rust


*My contribution included the development of an interactive CAVE Program using both IRIS Performer and OpenGL, and making it into an animation.*

• The 2D Illusion of Attic, Warping Beethoven I, Warping Beethoven II, and Warp the City, Tien-Tsin Wong, Chi-Wing Fu, and Pheng-Ann Heng, in *Siggraph 99*’ Technical Slide Set.

• Image-Based Attic, Attic Under Directional Lights, Attic Under Spotlights, and Disco, Tien-Tsin Wong, Chi-Wing Fu, and Pheng-Ann Heng, in *Siggraph 98*’ Technical Slide Set.

**Theses**

• Master's Thesis


• PhD Thesis

  *A Visualization Framework for Large-Scale Virtual Astronomy*, Chi-Wing Fu, PhD., Indiana University, Bloomington, IN 47405, U.S.A., December 2003.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.