

3D Scientific Visualization with Blender[®]

3D Scientific Visualization with Blender[®]

Brian R Kent, PhD

National Radio Astronomy Observatory, Charlottesville, VA, USA

Morgan & Claypool Publishers

Copyright © 2015 Morgan & Claypool Publishers

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publisher, or as expressly permitted by law or under terms agreed with the appropriate rights organization. Multiple copying is permitted in accordance with the terms of licences issued by the Copyright Licensing Agency, the Copyright Clearance Centre and other reproduction rights organisations.

Rights & Permissions

To obtain permission to re-use copyrighted material from Morgan & Claypool Publishers, please contact info@morganclaypool.com.

ISBN 978-1-6270-5612-0 (ebook)

ISBN 978-1-6270-5611-3 (print)

ISBN 978-1-6270-5614-4 (mobi)

DOI 10.1088/978-1-6270-5612-0

Version: 20150301

IOP Concise Physics

ISSN 2053-2571 (online)

ISSN 2054-7307 (print)

A Morgan & Claypool publication as part of IOP Concise Physics

Published by Morgan & Claypool Publishers, 40 Oak Drive, San Rafael, CA, 94903, USA

IOP Publishing, Temple Circus, Temple Way, Bristol BS1 6HG, UK

This book is dedicated to Linda Kent.

Contents

Preface	x
Acknowledgments	xi
Author biography	xii
1 Introduction	1-1
1.1 Visualization in the sciences	1-1
1.2 What is Blender?	1-2
1.3 Rendering engines	1-4
1.4 Community support	1-4
1.5 Types of data visualization in the sciences	1-5
Bibliography	1-6
2 The interface and windowing set-up	2-1
2.1 Interface introduction	2-1
2.1.1 3D view port	2-1
2.1.2 Using the keyboard	2-1
2.1.3 Quad view	2-2
2.1.4 UV-view	2-2
2.1.5 Object tools toolbar	2-3
2.1.6 Transform toolbar	2-4
2.1.7 Data outliner	2-5
2.1.8 Properties panel	2-6
2.1.9 Animation time line	2-9
2.2 Windowing set-up for Python	2-9
2.3 Data types and Python modules	2-9
2.4 Python libraries	2-10
Bibliography	2-10
3 Meshes, models and textures	3-1
3.1 Structure of 3D mesh objects	3-1
3.1.1 Example: building a simple model	3-1
3.2 2D materials and textures	3-3
3.2.1 Example: adding a texture to a sphere	3-3

3.3	3D materials and textures	3-6
3.3.1	Example: creating points for a 3D scatter plot	3-7
3.3.2	Example: creating a wireframe mesh	3-8
	Bibliography	3-9
4	Lighting	4-1
4.1	Lighting and color selection in Blender	4-1
4.1.1	Example: how to illuminate simulated terrain	4-1
4.1.2	Material emission	4-2
	Bibliography	4-3
5	Animation	5-1
5.1	Keyframes	5-1
5.1.1	Example: how to rotate and animate an object	5-1
5.1.2	Example: using the graph editor	5-3
5.1.3	Example: adding a cyclic modifier	5-5
5.2	Frame rates and rendering output	5-5
5.2.1	Output formats	5-5
5.3	Node compositing	5-5
6	Point of view: camera control	6-1
6.1	Projection differences: a matter of perspective	6-1
6.2	Camera keyframes	6-1
6.2.1	Example: tracking to an object	6-1
6.2.2	Example: an object following a path	6-4
	Bibliography	6-6
7	Python scripting	7-1
7.1	Scripting in blender	7-1
7.1.1	Blender and Python libraries	7-1
7.1.2	Example: reading formatted ASCII data into blender	7-2
7.1.3	Example: adding data points to an object as vertices	7-3
7.1.4	Example: animating multiple time steps	7-4
8	Projects and 3D examples	8-1
8.1	3D scatter plot	8-1
8.2	<i>N</i> -body simulation	8-3

8.3	Magnetic fields	8-10
8.4	Lagrangian equilibrium and zero-velocity curves	8-14
8.5	Geophysics: planetary surface mapping	8-20
8.6	Volumetric rendering and data cubes	8-21
8.7	Physics module and rigid body dynamics	8-26
	Bibliography	8-30
Appendix A Blender keyboard shortcuts		A-1

Preface

Observations, experiments and simulations in the sciences often have high impact visuals that never cease to provide insight into the Universe. This book is the result of exploring comprehensive software packages in 3D visualization for research in astronomy and astrophysics. Surveying the broader scope of a variety of scientific fields, we can see that the incredible results of our experiments can benefit in communicating results to both the scientific community as well as a broader audience.

This book is intended for the individual investigator or ambitious student to begin learning the fundamentals of 3D visualization for science. We utilize Blender, an agile and comprehensive software package for data visualization and animation. The application programming interface allows scientists to import and manipulate their data through the Python language. After introducing the fundamentals, the book concludes with a number of illustrative projects to bring various components together culminating in a series of visualizations.

A website¹ is also available as a companion to the book with example files and videos providing the user with step-by-step instructions on principles and projects outlined in the text. It is my hope that these will serve as a launching point for scientists to take their data visualizations to the next level.

¹ www.cv.nrao.edu/~bkent/blender/

Acknowledgments

Blender[®] is a registered trademark of the Blender Foundation. The author would like to thank Matthew Wood for directing me to the IOP Concise Physics series with Jeanine Burke and Joel Claypool of Morgan & Claypool Publishers, David Thomas for the Lagrange point calculations and Linda Kent for her encouragement. Members of the astronomical community and ACM SIGGRAPH conference provided useful queries that helped in developing the ideas of this book. The National Radio Astronomy Observatory is a facility of the National Science Foundation operated under cooperative agreement by Associated Universities, Inc.

Author biography

Brian R Kent



Brian R Kent, PhD is a scientist with the National Radio Astronomy Observatory in Charlottesville, Virginia. His publications and studies in astrophysics and computing include scientific visualizations of a variety of theoretical and observational phenomena. He is interested in visualizing data for scientific analysis as well as reaching a broad audience with the stunning visuals that modern 3D graphics can provide. Dr Kent received his PhD in Astronomy and Space Sciences from Cornell University. His website is: <http://www.cv.nrao.edu/~bkent/>

Chapter 1

Introduction

1.1 Visualization in the sciences

Science is all-encompassing with its ability to provide a logical path to facts, revolutionize technology and change the world as we know it, as well as inspire the next generation of technological advancements. Through the scientific method we form questions and develop hypotheses, and visualization of data plays a key role in testing and analysis. Insight into experiments is often found by how we present scientific and engineering results, concepts and ideas to all audiences, both non-scientific and scientific.

Modern computing has given the scientific community tools to take data exploration to the next level. The mining of large data sets requires efficient data organization as well as visualization. At times this is due to the sheer disk size of data, limited by memory or processing power. Developments in graphics processing units (GPUs), both for driving high-end graphics applications and threaded processing, allow for sophisticated high resolution images and animations to be created. In other scenarios the data may have a complex phase space to explore. Data may have N -dimensional tables or correlations among quantities that only reveal themselves through 3D visuals.

Many cutting-edge studies in a variety of scientific disciplines benefit from scientific visualization.

Astronomy. Telescope observations across the electromagnetic spectrum now generate terabytes of data every hour. Storing and managing the raw data is a challenge, let alone visualizing the processed data that have been pushed through the necessary calibration procedures. A survey of a million galaxies from a single wavelength regime will have metadata that need to be appropriately organized and indexed to search efficiently [1, 2]. Wide-band radio spectroscopy with radio telescopes gives 2D information about emission from the sky as well as frequency information along a third axis [3].

Physics. Cutting edge projects such as the Large Hadron Collider require advanced visualization software [4]. Theoretical models in particle physics, lattices in solid state physics, the physics of high-temperature plasmas

and gravitational waves all benefit from new visualization techniques [5]. Visualizing scientific apparatus before they are designed can help to optimize the engineering and design of experiments.

Chemistry/Biology. Complex molecules and molecular dynamics during reactions can be examined with 3D animations [6]. Studying proteins can reveal their structure using data from nuclear magnetic resonance (NMR) studies. GPU-accelerated processing has allowed for scalable solutions for visualizing complex viruses [7].

Geography/Planetary Science. A wealth of mapping data exist, not only for the planet Earth, but other surveyed planets in our Solar System. Using special data storage models and GPU processing, 3D maps of planetary surfaces can now be rendered in real time [8].

Medicine. Visualizing computed axial tomography (CAT scans) can show a transparent view of organic structures. Classification and feature extraction while performing diagnoses benefits greatly from be able to view real time 3D movies of internal organs [9].

1.2 What is Blender?

Blender is open source software that allows a user to create high quality animations of 3D models and data. It has wide usage in the video game and entertainment industries. The software is also extremely useful for generating high quality scientific visualizations. With its well organized Python application programming interface (API), it can be scripted to load data from numerical simulations. The power of Blender becomes evident when the user is given complete control over the camera angle, field of view and rendering aspects of the final animation.

Blender's traditional user base has been 3D graphics specialists working in modeling and animation. However, with the intuitive graphical user interface (GUI) and availability of Python libraries for reading a variety of scientific data, Blender brings an exciting and unique visualization suite to the modern scientific workbench. Developer/creator Ton Roosendaal and the Blender Foundation have created a community to maximize the amount of on-line material for developers and users¹. The goal of this book is to provide the reader with an interesting and practical introduction to Blender by way of science examples, each showing important software features and 3D graphics techniques.

Figure 1.1 shows the structure of Blender and its main capabilities. This book will examine each of the following topics and then use them in concert together for a number of example visualization projects:

- Meshes and models
- Lighting
- Animation
- Camera control
- Scripting
- Composites and rendering

¹<http://www.blender.org/forum/>

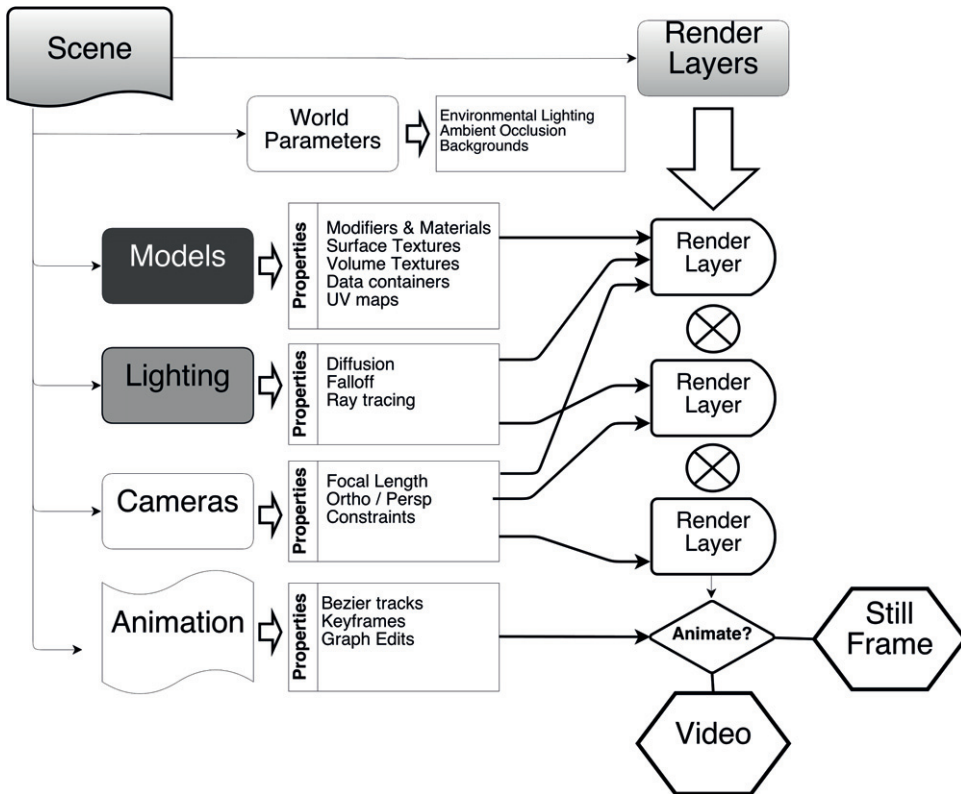


Figure 1.1. This flow chart shows how a Blender workflow can be used in scientific visualization. Different aspects of the Blender GUI and API will be analyzed throughout the book to show how various pieces can fit together in a project [10]. Copyright 2013 Brian R Kent, publications of the Astronomical Society of the Pacific.

There are also external rendering engines that can be used with Blender. While the examples and exercises in this book will use the internal Blender engines Render and Cycle, the reader is encouraged to examine other rendering options for different scenarios².

Blender can take advantage of multi-core central processing units (CPUs). The software also provides hooks for NVidia CUDA and OpenCL utilizing accelerated GPU rendering. This can decrease rendering times by a factor of three or four [10].

Blender also contains two powerful tools for building visualizations—the node compositor and video sequencer. The node compositor allows multi-layered visualizations to be combined into a final animation, through the use of nodes—a visual type of flow-chart style access to the Blender API and functionality. The video sequencer is useful for editing and combining animations for presentation.

²<http://www.blender.org/download/external-renderers/>

The software has modest desktop and laptop hardware requirements, depending on the level of complexity for the visualization. The software runs equally well on Windows, Mac OS X and many flavors of Linux. Hardware requirements, on-line help and documentation, along with the software itself can be found at <http://www.blender.org>.

Regardless of whether a desktop or laptop computer is used, it is highly recommended that a three-button mouse with a scroll wheel be used in Blender sessions. The numerical keypad on a standard keyboard also comes in handy when manipulating the 3D view, which will be described in a later section. If the user is on a laptop, then the top row numerical keys can be configured in the menu File → User Preferences → Input and by checking ‘Emulate Numpad’. We will cover how these numerical keys are used in a later section.

1.3 Rendering engines

While the vast majority of visualizations can be handled with the internal Blender rendering engine and its next-generation engine Cycles, a number of popular third-party applications are worth exploring.

Render. Most general purpose data visualizations that scientists will encounter can be found with the default renderer. This engine supports GPU processing and volumetric rendering. <http://wiki.blender.org/index.php/Doc:2.6/Manual/Render>

Cycles. The rendering path can be configured within the node-based compositor in Cycles, allowing for rapid prototyping of a scene. More realistic caustics are added with its rendering algorithm. <http://wiki.blender.org/index.php/Doc:2.6/Manual/Render/Cycles>

LuxRender. This open-source third party rendering engine has plug-ins for Blender and achieves fast ray tracing with GPU processing (see the references in [11]; <http://www.luxrender.net/>).

Mitsuba. A physically based rendering engine that can handle volume rendering. <http://www.mitsuba-renderer.org/>

YafaRay. Ray tracing and the ability to save files to high dynamic range images formats are features of Yafaray (EXR [12], <http://www.yafaray.org/>).

1.4 Community support

Blender has wide ranging community support, knowledge-bases and forums to help both new users starting out and advanced Blender aficionados looking to take their 3D visualization skills to the next level. Some popular websites include:

Blender Guru <http://www.blenderguru.com/>

Blender Artists <http://blenderartists.org/>

NASA 3D Model repository <http://nasa3d.arc.nasa.gov/>

BlenderNation <http://www.blendernation.com/>

BlenderCookie <http://cgcookie.com/blender>

1.5 Types of data visualization in the sciences

There are different kinds of scientific visualization to consider, each presenting unique technical scenarios when working with 3D graphics [13]. Each of the visualization types will be utilized to explain and expose useful features of Blender.

Solid models/surfaces/rigid body simulations. Surfaces work well for 2D waveforms, rigid body mechanics and 3D contour surfaces. They are also useful when working with geographic information system (GIS) maps. These types of visualizations are often textured to mimic a real world surface and externally illuminated. If glass or reflective surfaces are used, then ray tracing will increase the rendering time.

Data cubes/transparent/translucent rendering. If data are 3D and need to be rendered above some given noise level with a transfer function, then transparent rendering can be used. This is useful in medical and astronomical imaging. For medical imaging, the investigator wants to see multiple layers from a CAT scan. With astronomical imaging, the 3D nature of the data cube reveals positions on the sky as well as a Doppler shifted frequency, which can show the dynamics of rotating objects.

3D catalogs. These kinds of visualizations are akin to 3D scatter plots. They are useful for catalogs with three parameters in basic curvilinear coordinates, including Cartesian, cylindrical, or spherical systems.

N-body simulations. These are useful for displaying the results of gravitational interactions between point masses. The data structure utilization in Blender is similar to that for catalogs, except that *shape keys* are used to animate the simulation. Shape keys are used to increase the efficiency and reduce the memory footprint of the metadata in the Blender file.

Soft body simulations. These are used for simulating internally the physics of deformable objects within Blender. The results are often used to simulate the look of said objects visually and are useful for numerical simulations that cannot be solved analytically.

Surface/terrain maps. GIS maps can be loaded and color coded to show differences in terrain. Vertex painting can be utilized to overlay various features and color coded maps. An example will be given in a later chapter using Martian mapping data.

Phenomenological models. Blender has some utility in creating molecular models, as seen with the BioBlender project³ [14]. Chemical interactions can be animated and studied.

This book introduces important Blender concepts and, along the way, shows users the critical parts of the interface that they will need to know to accomplish their visualizations.

³<http://www.bioblender.eu/>

Bibliography

- [1] Szalay A S 2002 The SDSS Skyserver: public access to the sloan digital sky server data, arXiv: cs/0202013
- [2] Gray J 2002 Data mining the SDSS Skyserver database, arXiv: cs/0202014
- [3] Kent B R 2013 Visualizing scientific graphics for astronomy and astrophysics *Proc. ACM SIGGRAPH'13 (Anaheim, CA, 21–25 July 2013)* (New York: ACM) article no. 113
- [4] Fedorko I, Lefebure V, Lenkes D and Pera M O 2012 New data visualization of the LHC Era Monitoring (Lemon) system *J. Phys.: Conf. Ser.* **396** 042019
- [5] Sanders B C, Senden T and Springel V 2008 Focus on visualization in physics *New J. Phys.* **10** 125001
- [6] Moreland J L, Gramada A, Buzko O V, Zhang Q and Bourne P E 2005 The Molecular Biology Toolkit (MBT): a modular platform for developing molecular visualization applications *BMC Bioinform* **6** 21
- [7] Stone J E, Vandivort K L and Schulten K 2013 GPU-accelerated molecular visualization on petascale supercomputing platforms *Proc. 8th Int. Workshop on Ultrascale Visualization (Denver, CO, 17 November 2013)* (New York: ACM) 6
- [8] Myint S, Jain A, Cameron J and Lim C 2011 Large terrain modeling and visualization for planets *4th IEEE Int. Conf. on Space Mission Challenges for Information Technology (SMC-IT) (Palo Alto, CA, 2–4 August 2011)* (Piscataway, NJ: IEEE) 177–83
- [9] Linsen L, Hagen H, Hamann B and Hege H-C (ed) 2012 *Visualization in Medicine and Life Sciences* vol 2 (Berlin: Springer)
- [10] Kent B R 2013 Visualizing astronomical data with Blender *Publ. Astron. Soc. Pac.* **125** 731–48
- [11] Price R L, Puchala J C, Rovito T V and Priddy K L 2011 Physics accurate layered sensing model *Proc. 2011 IEEE National Aerospace and Electronics Conference (NAECON) (Dayton, OH, 20–22 July 2011)* (Piscataway, NJ: IEEE) 291–6
- [12] Debevec P 1998 Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography *Proc. 25th Annual Conf. on Computer Graphics and Interactive Techniques, Orlando, FL, 19–24 July 1998* (New York: ACM) 189–98
- [13] Friendly M 2006 A brief history of data visualization *Handbook of Computational Statistics: Data Visualization* vol 3, ed C Chen, W Härdle and A Unwin (Heidelberg: Springer)
- [14] Andrei R M, Callieri M, Zini M F, Loni T, Maraziti G, Pan M C and Zoppè M 2012 Intuitive representation of surface properties of biomolecules using BioBlender *BMC Bioinform.* **13** S16

Chapter 2

The interface and windowing set-up

2.1 Interface introduction

For the exercises and descriptions in this book, we shall use the default GUI that loads with Blender. The interface that loads with Blender by default is geared toward a workflow of manipulating data within the 3D view port, animation bar and object metadata, as well as materials, textures and other object and mesh properties. We will augment this for using the Python API and writing data loading scripts. The interface is highly customizable and the reader is encouraged to experiment with GUI organization depending on their particular screen set-up, preferences and visualization goals. The main GUI is shown and described in figure 2.1.

2.1.1 3D view port

The 3D view port is where most of the interaction with the actual data object takes place. When Blender first opens, the default view shows a triplet of arrows at the origin (figure 2.2). They are colored red, green and blue for the X -, Y - and Z -axes, respectively. To rotate the scene about the white circle at the center of the axes, click, hold and drag the middle mouse wheel. Rotating the middle mouse wheel forward and back will zoom into and out of the scene. The default view of the scene is in perspective, in that parallel lines will converge to a vanishing point on a horizon line (just as a road in a drawing created in a single or two-point perspective will converge to a distant point). The left mouse button moves the red and white 3D cursor around the view port. Any new meshes or objects will be added at the location of the 3D cursor.

2.1.2 Using the keyboard

Keyboard shortcuts are vital for manipulating 3D data to a position that one needs. Figure 2.3 shows the numerical keypad and what action each key can perform. The 1, 3 and 7 keys will show the top, bottom and side views. 2, 4, 8 and 6 will rotate

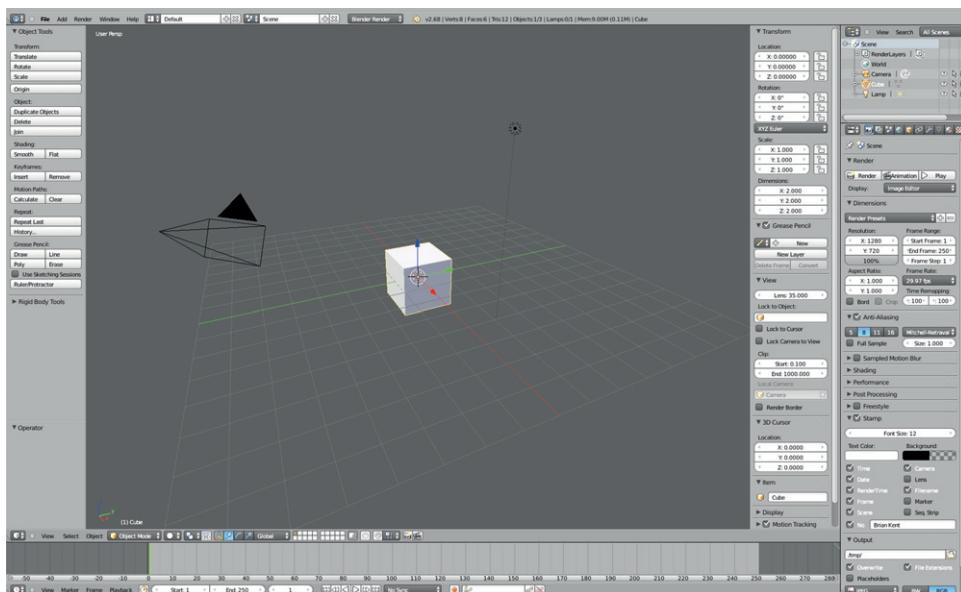


Figure 2.1. The default Blender GUI that the user will see upon start-up. The main 3D view port allows the user to interactively manipulate and create 3D objects. The left-hand side shows the Object Tools toolbar and the right-hand side the Transform toolbar. The bottom of the screen shows the animation ‘tape deck’ and the far right shows the data outliner and rendering, materials and texture tools.

the view in 15° increments. Using the keypad is advantageous for exact control and positioning of the view. The central 5 key switches between orthographic and perspective modes. Both views have advantages in planning and executing scientific visualizations. A list of keyboard shortcuts is given in appendix A.

Object movement can occur via translation, rotation and scaling (figure 2.4), either in the GUI or via keyboard shortcuts. The bottom of the GUI has a number of drop-down menus for switching between Object and Mesh Edit modes, selecting the rendering style (wireframe, solid, textured, full) and changing the reference frame (figure 2.5). In Mesh Edit mode single or groups of vertices, lines, or faces can be selected (figure 2.6).

2.1.3 Quad view

A particularly useful GUI set-up is quad view. This can be accomplished with CTRL-ALT-Q, and then shows isometric views along the X, Y and Z lines of sight, as well as the view from the active camera (figure 2.7).

2.1.4 UV-view

The UV-editing GUI can be accessed via the drop-down menu at the top of the Blender screen. This splits the main window interface into a 3D view port and UV-editing interface that can be utilized when mapping data to mesh surfaces. This is typically

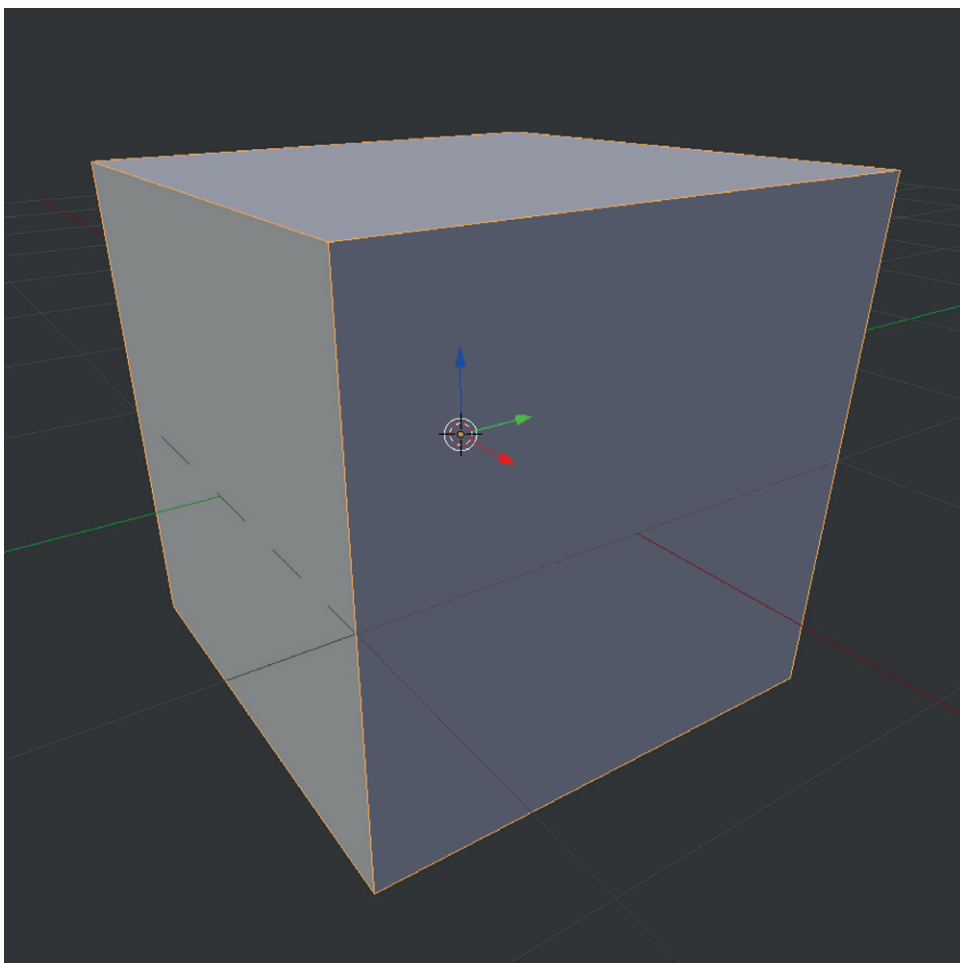


Figure 2.2. A mesh cube with the origin arrow triplet colored red, green, and blue for the X -, Y -, and Z -axes, respectively. Controls are available for translation, rotation and scaling. The user can grab these handles to manipulate an entire mesh object, or a single vertex, line, or face.

accomplished by entering Mesh Edit mode with the TAB key, right-clicking to select the desired vertices and then pressing the U key to map the image to the surface with one of the projection methods.

2.1.5 Object tools toolbar

The Object Tools toolbar together with the TAB key allows the user to modify the properties of mesh objects. The toolbar will change depending on the object selected; the properties for a selected camera or lighting element will be different than a mesh object (figure 2.8(a)). In later sections we will take advantage of azimuthal symmetry and demonstrate the utility of the spin tool (and others) to save time in building models.

Num	/	*	-
7 Top	8 Rotate Up	9	+
4 Rotate Left	5 Persp / Ortho	6 Rotate Right	
1 Front	2 Rotate Down	3 Side	Enter
0		.	

Figure 2.3. A schematic keypad view of a standard keyboard. The 2, 4, 6 and 8 keys allow the user to rotate the selected object or data container in 15° increments. The 3, 1 and 7 keys shift the view along the line of sight for the X -, Y - and Z -axes, respectively. The 5 key switches the 3D view port between orthographic and perspective modes.

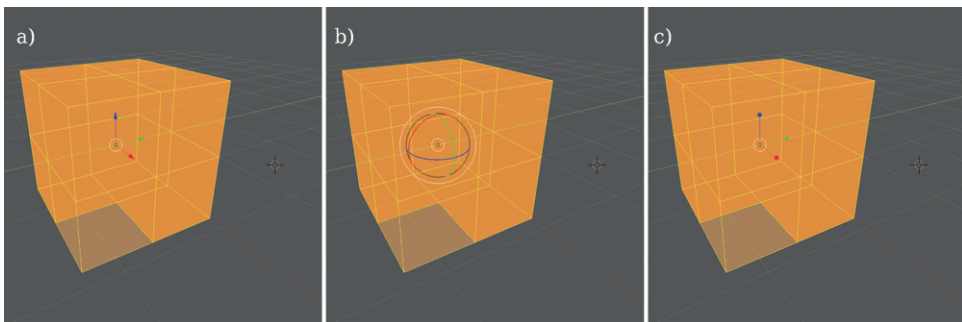


Figure 2.4. Three Blender mesh objects showing the mouse handles for (a) translation, (b) rotation and (c) scaling. These GUI elements allow the user to manipulate and position objects in the 3D view port [1]. Copyright 2013 Brian R Kent, publications of the Astronomical Society of the Pacific.

2.1.6 Transform toolbar

The Transform toolbar is an important part of the GUI in that it allows the user to precisely manipulate and refine the location, rotation and scale of a given object. This is often used in conjunction with the keypad controls. Mesh positions can be locked and the 3D cursor can be moved. The viewing distance, known as the ‘clip’, allows control of how far from the currently 3D view point the user can see (figure 2.8(b)). This is particularly useful in crowded visualization scenes.

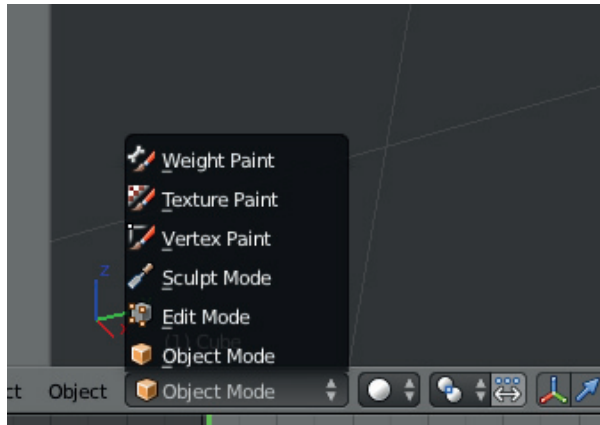


Figure 2.5. The drop-down menu for mode selection. The exercises in this book will focus primarily on the Object and Mesh Edit modes.

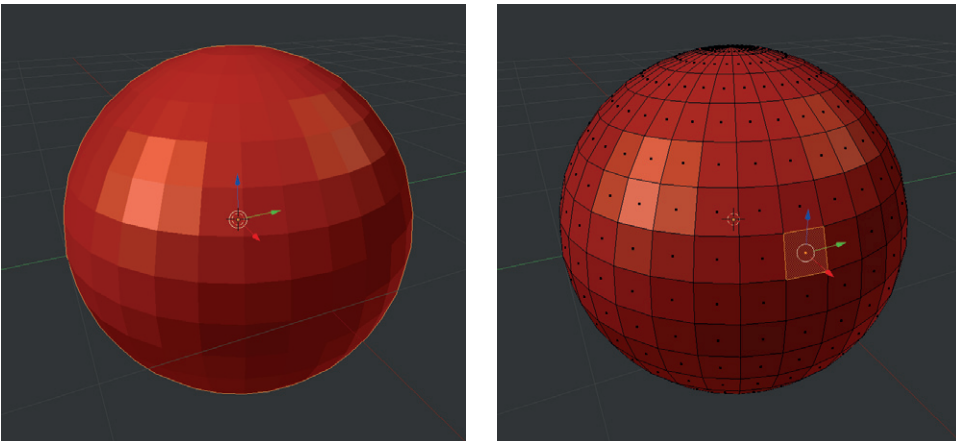


Figure 2.6. The left panel shows a UV-sphere in Object mode. The right panel shows the same sphere with a single face selected in Mesh Edit mode.

2.1.7 Data outliner

The data outliner (figure 2.9) shows all objects that exist in a scene. It depicts in a hierarchical format which camera is active and what objects are currently contributing to a scene. Objects can be toggled on or off in both the scene and/or final render. This is useful for having alignment guides in the GUI that are absent in the final render. Objects that have common attributes can be grouped together for ease of organization. When working with a visualization scene, the user can select which elements are visible in the 3D view port or are required in the final render. Some objects might be used for alignment or reference when building a scene, but will not

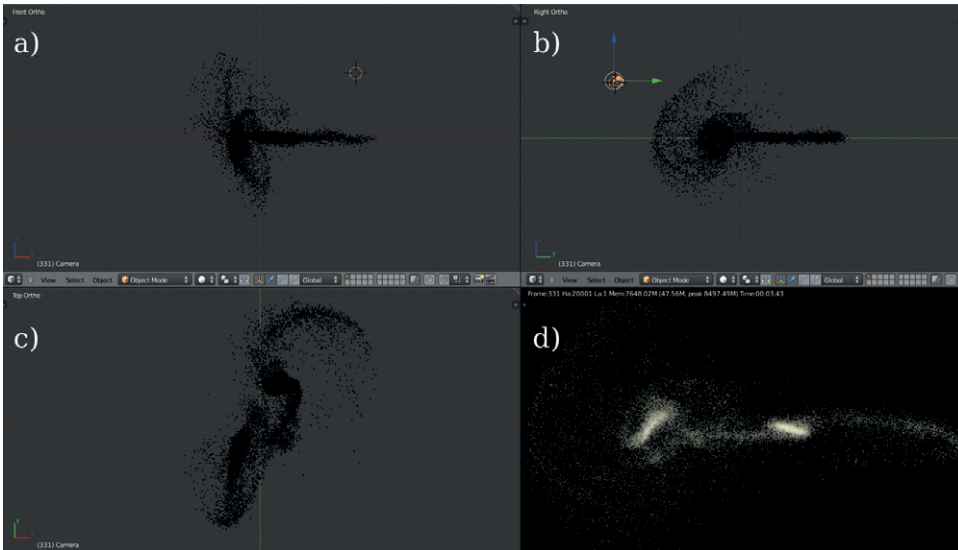


Figure 2.7. Example Blender GUI configured in quad view showing an N -body simulation of colliding galaxies with the top, side and front views, as well as the view from the currently selected camera. The quad view is useful for visual analysis from multiple directions and what the field of view for the final render will show [1]. Copyright 2013 Brian R Kent, publications of the Astronomical Society of the Pacific.

be used in an animation or render. At other times, a particular reference data object might be too computationally expensive to put into a test render and be temporarily excluded from the scene until the final render is created.

2.1.8 Properties panel

The Properties panel is a multi-tabbed panel on the right-hand side of the GUI. These include the tabs for rendering, layers, scenes, world controls, object controls, constraints, modifiers, vertex controls, materials, textures, particle generators and physics simulations (figure 2.10). Some of the tabs listed here will be covered in more detail as their features are needed in later sections.

Render. This tab sets the visualization size and resolution, file animation types and metadata. Presets are available for common video types and formats.

Scene layers. This tab is used for masking and blanking out layers in the visualization scene.

Scene properties. Fundamental scene properties including scaling units can be set here. This is also the tab where the rigid body physics simulation parameters are controlled.

World. Setting environmental lighting and background colors can be controlled here depending on whether the visualization will be published as a graphic in a journal or if it will be better suited for a high-definition animation.

Object. GUI configuration and relations to other objects and layers can be controlled here.

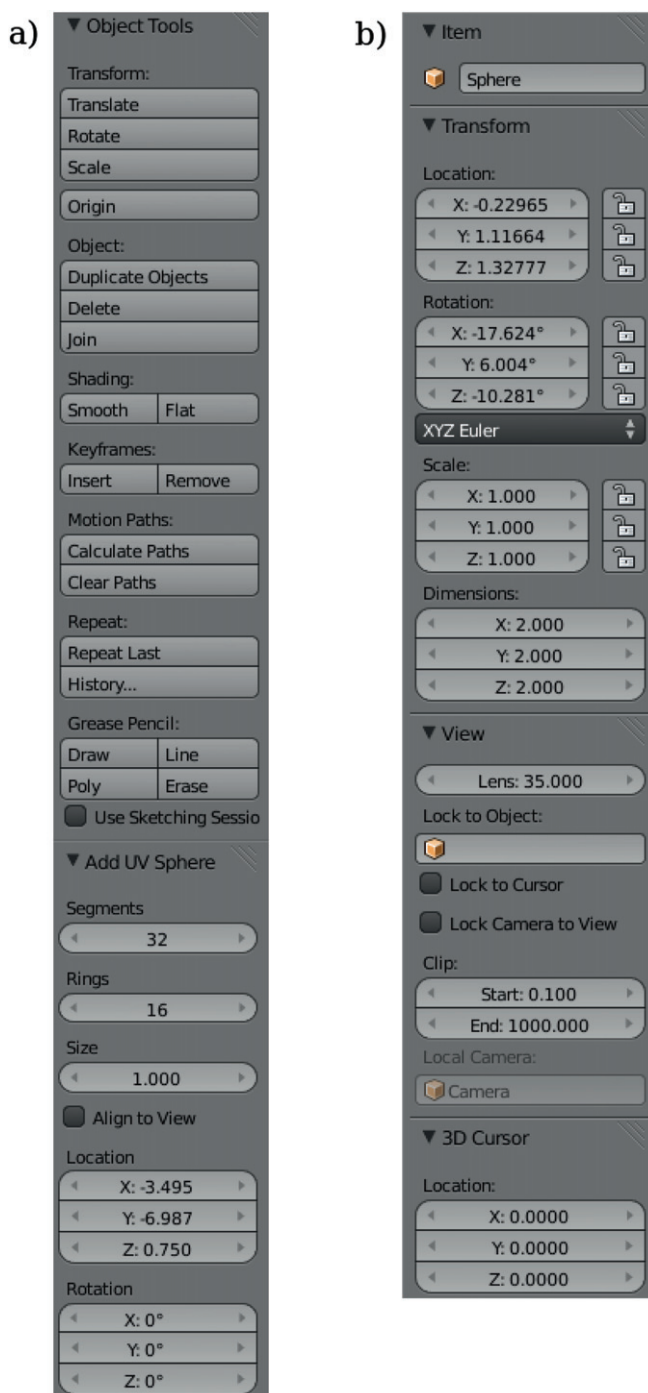


Figure 2.8. The Object Tools and Transform toolbars which are used to precisely position objects and cursors in the 3D view port. Each of the object properties can be keyframed from the Transform toolbar. By pressing the TAB key and entering Mesh Edit mode, individual elements can be manipulated via the Object Tools toolbar.

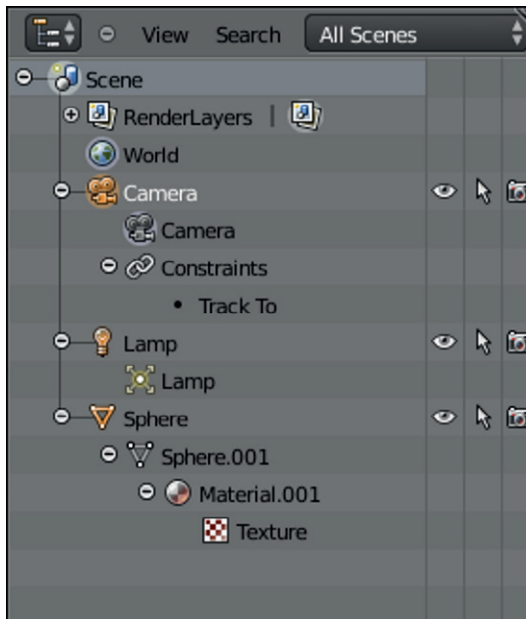


Figure 2.9. The data outliner in Blender gives a global view of all the objects and associations between them. Child/parent relationships, rendering status and 3D view port status are all reflected in this useful GUI element.



Figure 2.10. The Properties panel where materials, textures, rendering and scene controls can be manipulated. From left to right the panel includes rendering, layers, scenes, world controls, object controls, constraints, modifiers, vertex controls, materials, textures, particle generators and physics simulations.

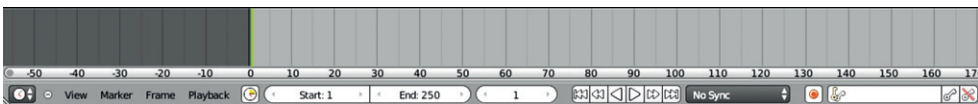


Figure 2.11. The Blender animation time line that allows for scaling the frames of an animation as well as viewing keyframe markers. From left to right the buttons indicate the start and end frames, current frame, beginning frame, previous keyframe, reverse play, forward play, next keyframe and end of the animation.

Constraints. Mesh objects can have their motion or positions restrained or locked relative to other objects or defined criteria. This is useful when building camera tracks.

Modifiers. Modifiers can be used to extend data objects without increasing the number of vertices or polygons. This is useful when making grids that need to render quickly.

Vertex groups and shape keys. Shape keys are useful for keyframing animations with large numbers of particles, including smooth particle hydrodynamics and N -body simulations.

Materials. An important part of scientific visualization involves selecting between surfaces, grids and point materials to convey a variety of visualization elements. These could include a range of phenomena, such as planetary surfaces, fields lines, or data best conveyed through 3D scatter plots.

Textures. Textures are useful for loading topographic maps and 3D data cubes, which can be brought into layered textures or voxel data structures, respectively.

Particles. The built in Blender particle generator can be used for phenomenological models and tracers for N -body and smoothed particle hydrodynamics (SPH) simulations.

Physics. This engine allows the user to set up forces, gravitational and magnetic fields, and use the rigid and soft body dynamics features.

2.1.9 Animation time line

In order to facilitate moving through a simulation or animation, the Blender GUI provides animation controls (figure 2.11). The animation time line is similar to a set of classic tape deck controls. Rewind, play, fast forward, and click and drag to see all animation and camera movements occurring in the 3D view port. In addition, once keyframes have been set and an object is selected, the user can jump between those keyframes and see them as yellow lines on the time line.

2.2 Windowing set-up for Python

Any of the GUI elements mentioned so far can be swapped and changed with the drop-down window selector present in a corner of each window element. In addition, a window can be separated by SHIFT-clicking in the upper right-hand corner of a window element. In the new window, the same corner can be clicked and dragged to split the window in two. The top element can be changed to a Python script editor and the bottom to a Python terminal console¹. With the Blender Python API this can be used to script changes to scene objects and mesh constructs, import data into Blender container objects and run tasks in batch that would otherwise be arduous to carry out in the GUI.

2.3 Data types and Python modules

A number of important data formats and key Python modules should be considered when using Blender. These are useful during data import or in performing numerical calculations.

OBJ file (type text/plain). An OBJ file is a simple ASCII text file that contains information about 3D geometry. From the UV coordinate positions of each vertex and normal, a 3D model can be created [2].

¹wiki.blender.org/index.php/Doc:2.6/Manual/Extensions/Python/Console

FITS. The scientific data format used in astronomy—the flexible image transport system (FITS). These files can contain 2D and 3D imaging, spectra and time-series [3].

GIS Shapefiles. The vector data format for geographic information systems [4]². <https://github.com/domlysz/BlenderGIS>

MDL Molfile. A chemistry file with atomic coordinates and bond formation for building molecules [5].

Image formats. JPEG, GIF, PNG and BMP files can all be imported for use as texture images for 3D surfaces using the UV editor and materials/textures tabs.

2.4 Python libraries

numpy. Libraries for numerical analysis. Numpy is included with Blender by default. <http://www.numpy.org/>

scipy. Math tools and libraries that make use of numpy. Scientific python can be found at <http://www.scipy.org/>.

astropy. An excellent set of core utilities for astronomers using Python [6]. <http://www.astropy.org/>

BeautifulSoup. A useful utility of parsing XML and HTML tables of data. <http://www.crummy.com/software/BeautifulSoup/>

Bibliography

- [1] Kent B R 2013 Visualizing astronomical data with Blender *Publ. Astron. Soc. Pac.* **125** 731–48
- [2] Murray J D and van Ryper W 1996 *Encyclopedia of Graphics File Formats* 2nd edn (Paris: O'Reilly)
- [3] Hanisch R J, Farris A, Greisen E W, Pence W D, Schlesinger B M, Teuben P J, Thompson R W and Warnock A III 2001 Definition of the Flexible Image Transport System (FITS) *Astron. Astrophys.* **376** 359–80
- [4] Scianna A 2013 Building 3D GIS data models using open source software *Appl. Geomatics* **5** 119–32
- [5] Dalby A, Nourse J G, Hounshell W D, Gushurst A K I, Grier D L, Leland B A and Laufer J 1992 Description of several chemical structure file formats used by computer programs developed at Molecular Design Limited *J. Chem. Inform Comput. Sci* **32** 244–55
- [6] Robitaille T P *et al* 2013 Astropy: a community python package for astronomy *Astron. Astrophys.* **558** A33

²www.esri.com/library/whitepapers/pdfs/shapefile.pdf

Chapter 3

Meshes, models and textures

3.1 Structure of 3D mesh objects

3D objects in Blender are often referred to as meshes. Each mesh is fundamentally composed of vertices, lines and faces. A *vertex* is a singularity construct—it has no volume associated with it—and is defined by its X – Y – Z coordinates in the global 3D view port. These coordinates can be transformed to another coordinate system or reference frame. Vertices can be connected via *lines* and a closed loop of vertices and lines can form the boundary for a solid, flat surface called a *face*.

A mesh has geometric properties that include an origin about which the object can rotate and/or revolve. The individual vertices, lines and faces can be edited in Mesh Edit mode, accessed via the Mode Selection drop-down menu or the TAB key (figure 2.5). Each mesh has a selection mode via those vertices, lines and faces.

3.1.1 Example: building a simple model

A base mesh can be created by Add → Mesh from the menu at the top of the screen. The basic meshes are shown in figure 3.1. These can be deformed, changed and extended to fit the scientific visualization required.

- Create a new cube mesh by clicking Add → Mesh → Cube. Alternatively, the keyboard shortcut SHIFT–A can be used.
- Rotate the cube with the R key in the plane normal to the line of sight.
- Scale the cube with the S key.
- Translate the cube in the plane normal to the line of sight with the G key.
- The Transform toolbar on the right-hand side of the GUI allows for exact positioning of the mesh object.

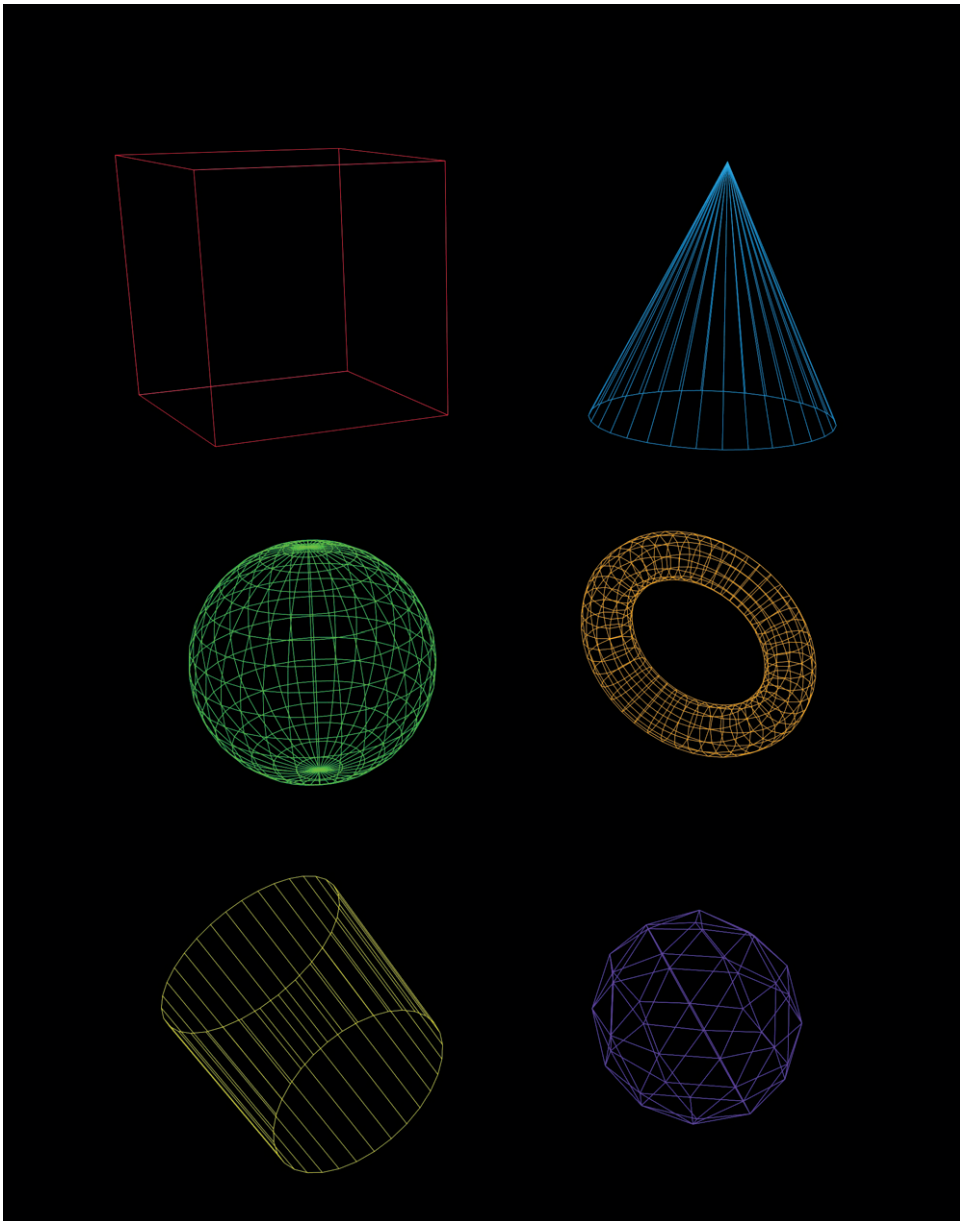


Figure 3.1. Blender mesh examples that form the basis of data containers and grid constructs in scientific visualization. These basic shapes—a cube, cone, UV-sphere, torus, cylinder and icosphere—can be manipulated in the Blender 3D view port.

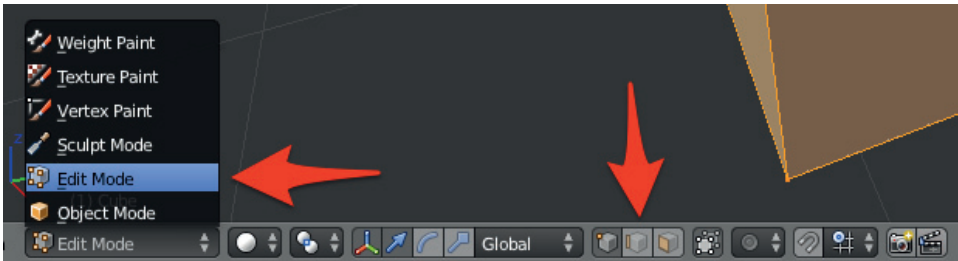


Figure 3.2. Blender Mesh Edit mode is selected and the vertex, line and face mode buttons are shown. This allows the user to manipulate individual parts of a mesh object.

We can further manipulate the individual elements of the mesh in a number of ways. This can help the user precisely position the scene elements and data objects of a visualization.

- At the bottom of the 3D view port, click the drop-down menu and choose ‘Edit Mode’. Alternatively, press the TAB key on the keyboard.
- The mesh element faces, lines and vertices will be highlighted in orange. The mode for vertex, line, or face selection is shown in figure 3.2. A mesh element can be selected with the secondary mouse button (usually the right mouse button). Multiple elements can be selected by holding down the SHIFT key.
- Groups of elements can be selected by hitting the B key on the keyboard (for box select) and then clicking and dragging a box over the selected points.
- Add vertices connected by lines by CTRL left-clicking where the new vertex needs to be placed.

3.2 2D materials and textures

Textures can be applied in a variety of scenarios. 2D textures can be utilized to give a face a more realistic surface appearance, apply mapping data, and change the visibility and color of the data. Bump mapping can also be applied to meshes to simulate 3D surfaces. This has important applications in increasing the speed of rendering times with lower polygon counts [1]. 2D materials and textures can be applied to single faces or an entire mesh object in a variety of projections in the UV-plane. A orthographic projection map of the Earth is shown in figure 3.3 and 3.4.¹

3.2.1 Example: adding a texture to a sphere

We can use the following procedure to project the map onto a sphere. This requires marking a seam on the sphere where the object will be separated. A projection of the mesh can then be matched to an image or a map.

- Create a UV-sphere mesh by clicking Add → Mesh → UV Sphere. Alternatively, the keyboard shortcut SHIFT-A can be used.

¹For excellent tutorials on both novice and advanced Blender usage visit <http://www.blenderguru.com/>.

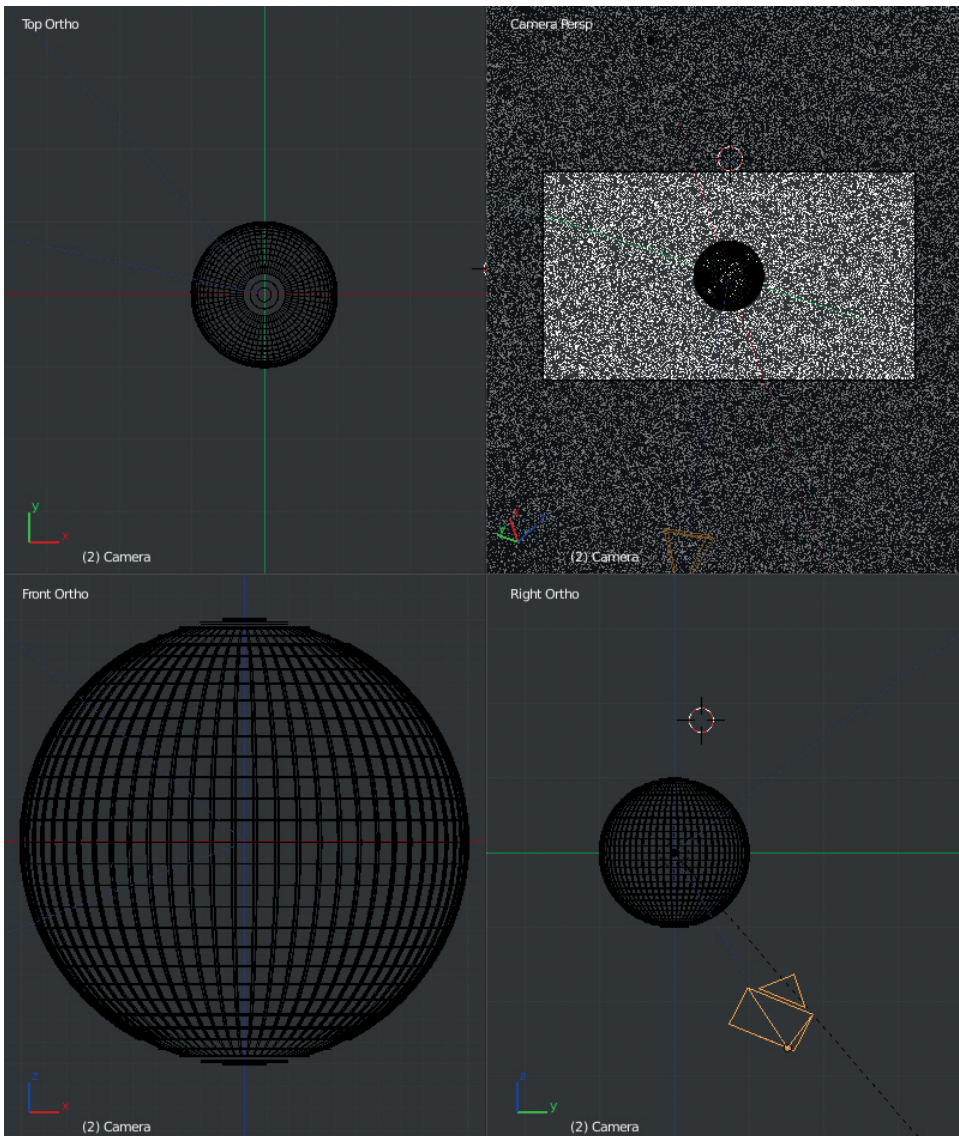


Figure 3.3. This view shows the set-up for projecting a 2D map of the Earth onto a UV-sphere.

- Increase the number of polygonal faces for the sphere.
- Select Mesh Edit mode by pressing the TAB key
- Right-click on vertices along a meridian going from the north to south pole of the sphere.
- Press CTRL-E on the keyboard and select 'Mark Seam'. The seam will be marked in red and show where the 3D object will be separated (figure 3.5).

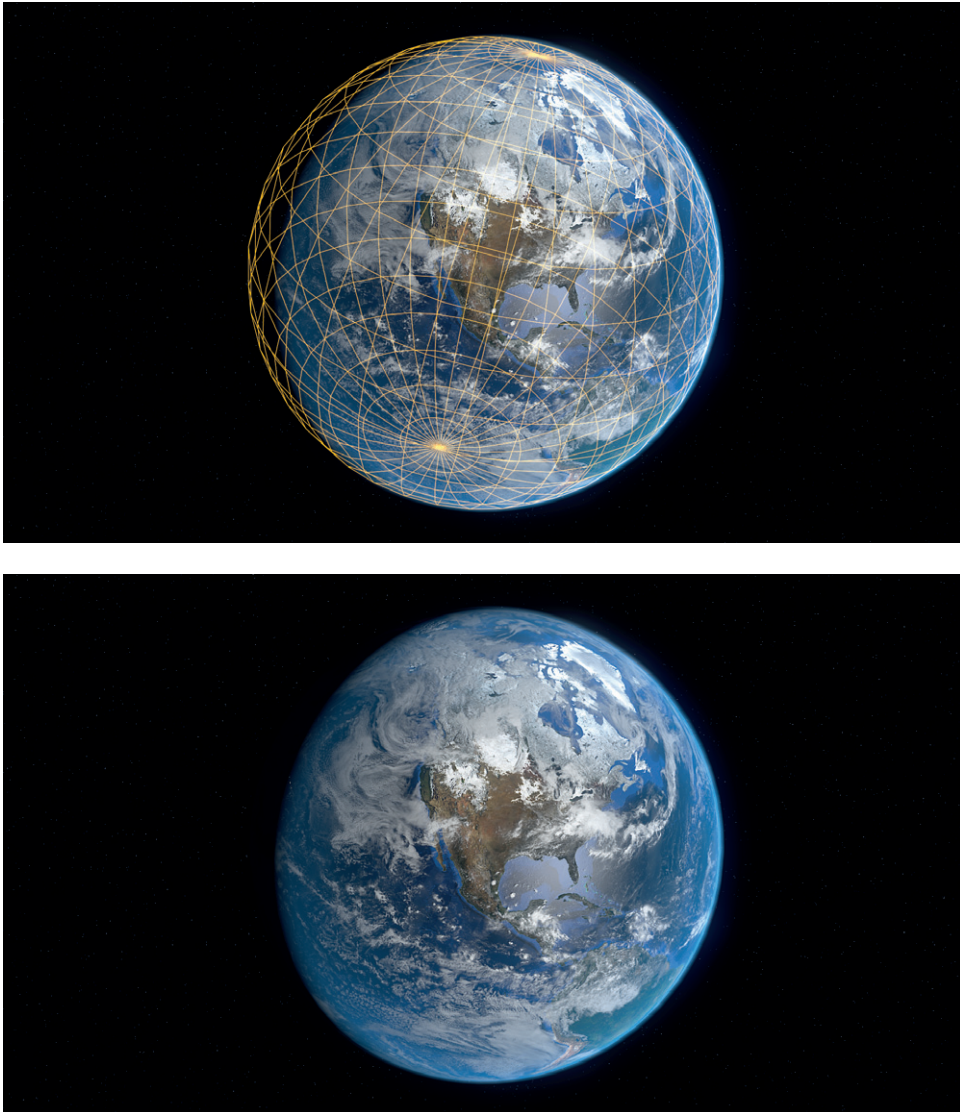


Figure 3.4. Earth map projection with images from <http://earthobservatory.nasa.gov>. Several layers are presented here in a final composite, including a day and night side maps of the Earth and an atmospheric layer.

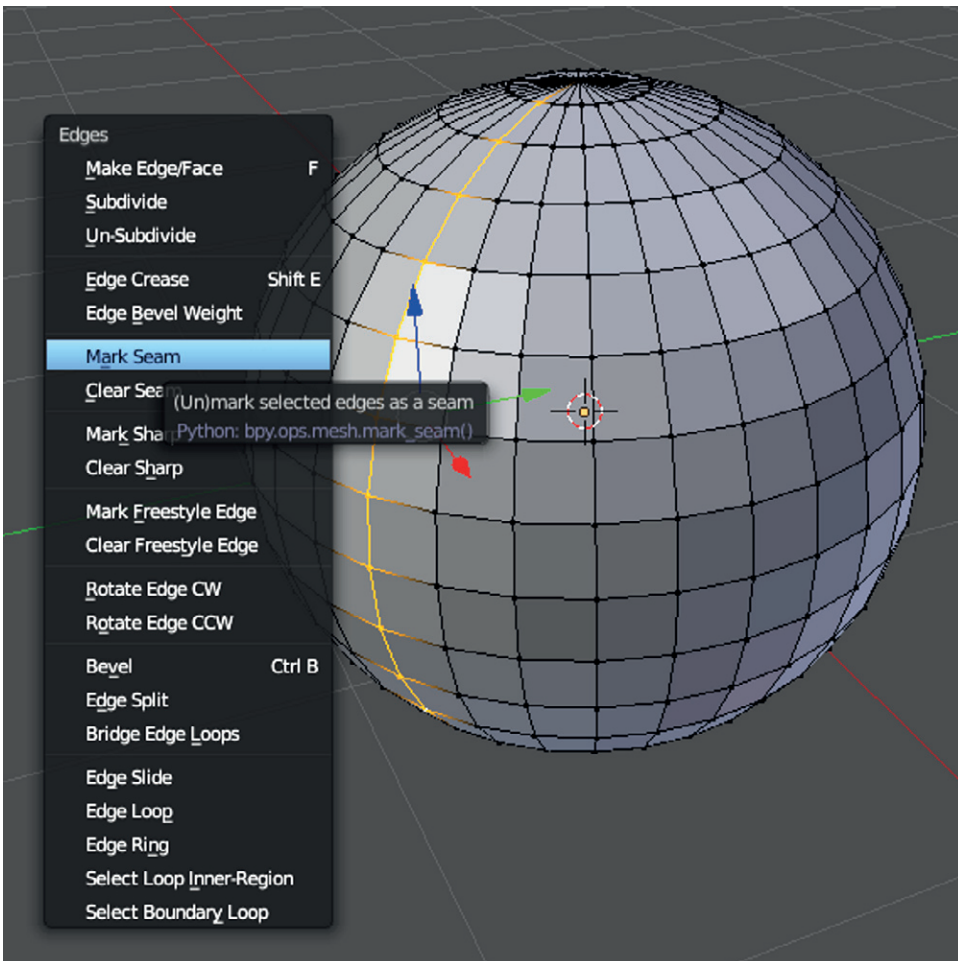


Figure 3.5. In Mesh Edit mode (TAB key), a seam can be added with CTRL-E. In this particular example we are marking the seam along a meridian.

- Choose the UV-editing menu and open a new image file (ALT-O).
- Select all vertices (A key), press the U key for UV Mapping and select 'Sphere Projection' (figure 3.6).
- The results can be seen by selecting 'Texture Mode' under the 'Viewport Shading' drop-down menu.

3.3 3D materials and textures

For 3D materials and textures, Blender can apply halos to data points or render data cubes as transparent volumes. Halo textures can be applied to vertices and are useful for creating 3D scatter plots.

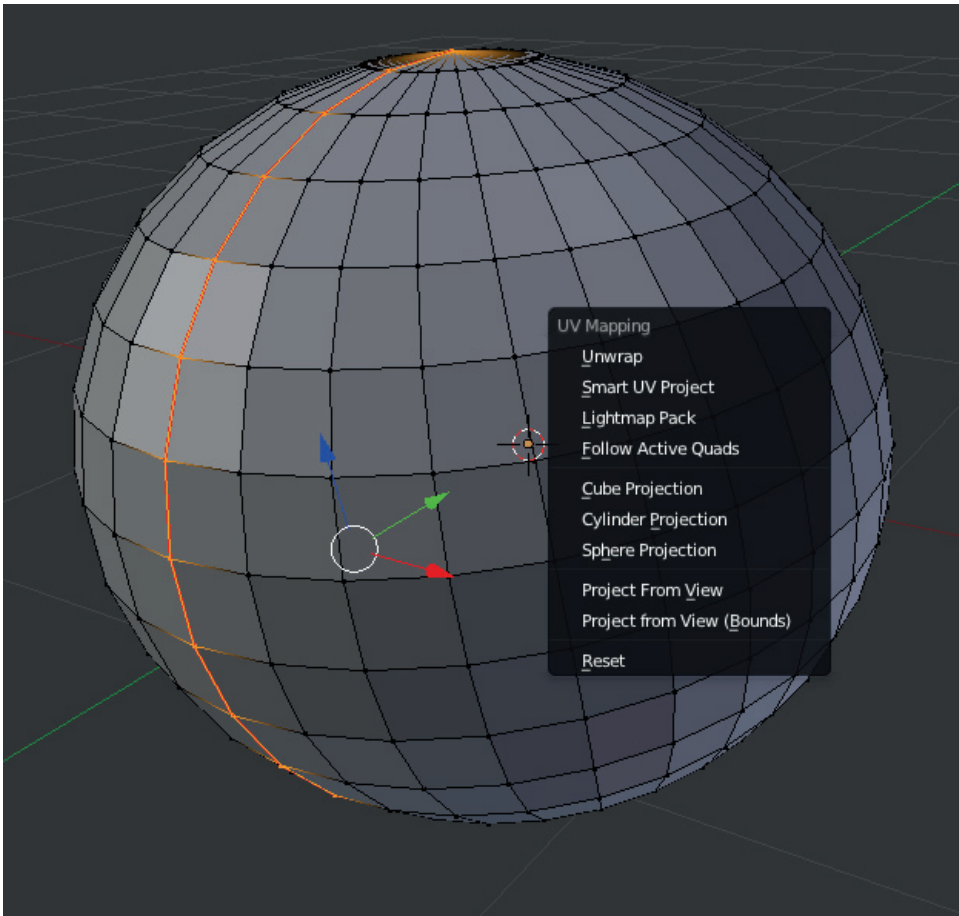


Figure 3.6. With the seam on the meridian properly marked a map projection can now be applied using UV Mapping.

3.3.1 Example: creating points for a 3D scatter plot

We will use the vertices of a 3D mesh as our sample X , Y , Z data and texture those points with a halo.

- Create a UV-icosphere mesh by clicking Add → Mesh → Icosphere. Alternatively the keyboard shortcut SHIFT-A can be used.
- Go to the Materials tab on the Properties panel and click 'New' to add a new material.
- Select the 'Halo' option (figure 3.7).
- Change the halo parameters: size to 0.050 and hardness to 20.

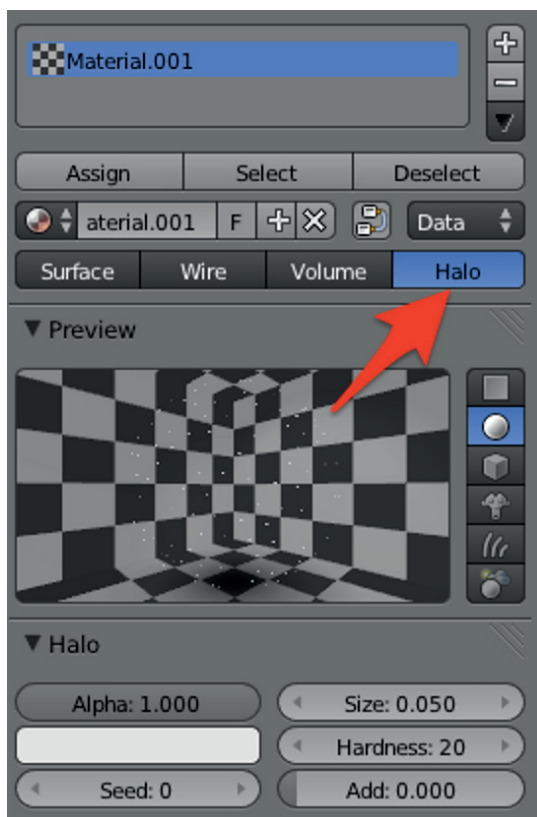


Figure 3.7. The halo material can be applied to illuminate individual vertices in a scatter plot or catalog. These small Gaussian spheres have size, illumination and color properties that can be adjusted.

3.3.2 Example: creating a wireframe mesh

With the same icosphere mesh object (or any mesh), a 3D wireframe object can be created. These are useful for creating background grids in 3D scatter plots. Simply change the material to ‘Wire’ and the shading emission to 1.0 (figure 3.8).

- Create a plane mesh by clicking Add → Mesh → Plane. Alternatively, the keyboard shortcut SHIFT+A can be used.
- Enter Mesh Edit mode by pressing the TAB key on the keyboard.
- On the Mesh Tools panel (left-hand side of the GUI), click ‘Subdivide’ five times to increase the number of grid points in the plane.
- Press the TAB key again to re-enter Object mode. Scale the object to a larger size with the S key.
- Go to the Materials tab on the Properties panel and click ‘New’ to add a new material.

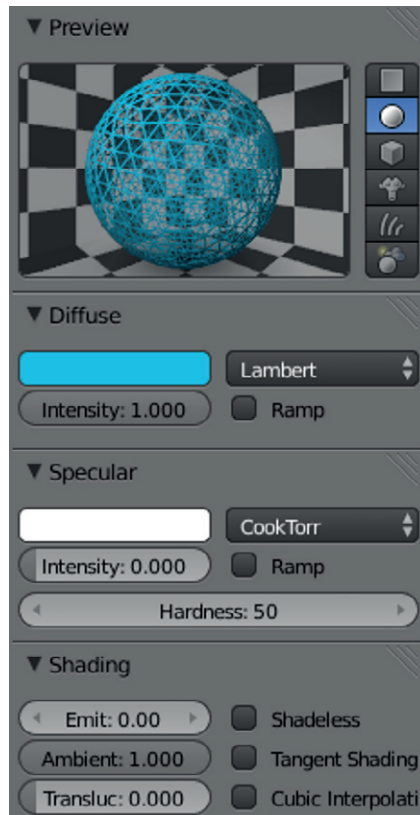


Figure 3.8. A wire material can be applied to a mesh. This is useful in creating grids and bounding boxes for visualizations.

- Select the 'Wire' option (figure 3.8). Change the color if desired.
- Change the wire material parameters: Diffuse Intensity: 1.0, Specular Intensity: 0.0 and Shading Emission: 1.0.

Bibliography

- [1] Blinn J F 1978 Simulation of wrinkled surfaces *SIGGRAPH Comput. Graph.* **12** 286–92

Chapter 4

Lighting

4.1 Lighting and color selection in Blender

Lighting in Blender can be used for aesthetic purposes and during animations to highlight particular parts of a dataset. For scientific visualization there are a number of key lighting elements to consider. *Point lamps* are isotropic radiators with a scalable intensity value, called the ‘Energy’ number. *Sun lamps* are directional radiators at infinity. *Spot*, *point* and *area lamps* are localized directional lamps (figure 4.1).

4.1.1 Example: how to illuminate simulated terrain

- Create a plane mesh by clicking Add → Mesh → Plane. Alternatively, the keyboard shortcut SHIFT+A can be used.
- Enter Mesh Edit mode by pressing the TAB key on the keyboard.
- On the Mesh Tools panel (left-hand side of the GUI), click ‘Subdivide’ six times to increase the number of grid points in the plane.
- Press the TAB key again to re-enter Object mode. Scale the object to a larger size with the S key.
- Go to the Modifiers tab on the Properties panel and choose ‘Displace’. Add a texture and set ‘Strength’ to 0.2.
- Add a second modifier ‘Subdivision Surface’ and set both the ‘View’ and ‘Render’ numbers to 5.
- On the left-hand side Object Tools panel, click ‘Smooth Shading’. This has created a random simulated topographic map.
- Back on the Properties panel, add a material and set the parameters: Diffuse Intensity: 1.0 and Specular Intensity: 0.0.
- Lights can be added by clicking Add → Lamp → Sun. This lamp object can be positioned, rotated and animated like any other Blender object (figure 4.2).

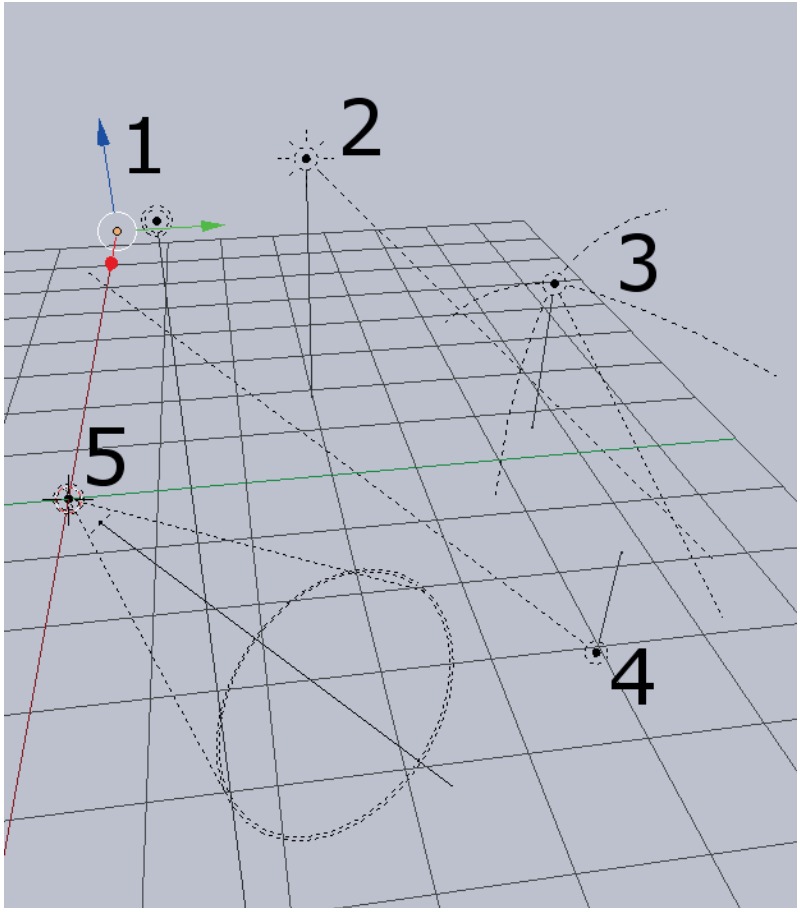


Figure 4.1. The different types of lighting elements include 1 point, 2 sun, 3 hemi, 4 area and 5 spot lamps.

4.1.2 Material emission

Emission for a mesh object is controlled on the Properties panel Materials tab with the Emit slider. By increasing the value, an object will become visible without external lighting sources (or *lamp objects*). Examples of using this include creating backgrounds grids for data or having self-illuminating data points for a 3D scatter plot. Typically a combination of lamps and mesh emission values are used to illuminate a rendered scene.

In addition to controlling the color of an object or lighting element, the transparency and translucency can be controlled under the ‘Shading’ and ‘Material’ transparency options. The default options will usually result in a faster render time; using ray tracing will often increase the amount of time for the final render [1].

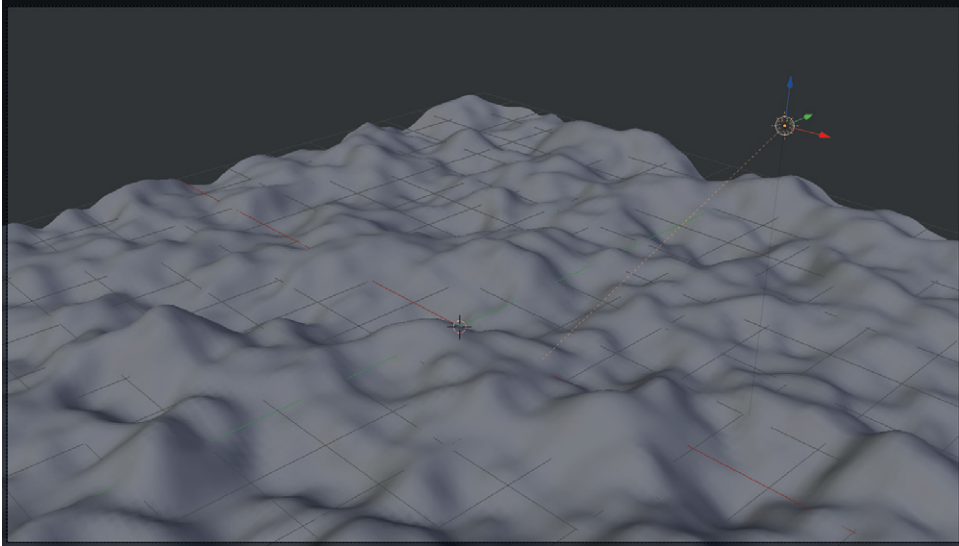


Figure 4.2. Simulated terrain generated with a displacement modifier and showing a lighting element in the 3D view port.

Bibliography

- [1] Kent B R 2013 Visualizing astronomical data with blender *Publ. Astron. Soc. Pac.* [125 731–48](#)

Chapter 5

Animation

The output of a scientific visualization created with Blender can include 3D scatter plots, static images of a 3D scene, or 3D animations with or without supporting camera movements and tracking.

5.1 Keyframes

To facilitate the animation of a mesh object, the X , Y , Z position, rotation and scaling must be recorded in a *keyframe*. Keyframes are noted in the animation playback bar at the bottom of the GUI via yellow lines pertaining to the selected object. Keyframes are inserted via the I key—possible selections are shown and described in figure 5.1. Keyframes are deleted via the key combination ALT-I. An advantage of using keyframes is that Blender will automatically interpolate between frames—keyframes do not need to be set for every frame in the animation of a mesh object.

5.1.1 Example: how to rotate and animate an object

The example below shows how to insert rotational keyframes to rotate a torus about the X -axis (colored red in the default 3D view space) and edit them with the Graph Editor so smooth motion can be created. The animation will be 300 frames at 30 frames per second for a total duration of 10 s. In those 10 s, the torus mesh will rotate 360 degrees (figure 5.2).

- Set the number of end frames to 300 on the Animation toolbar at the bottom of the screen. Set the current frame to 1.
- Create a torus mesh by clicking Add → Mesh → Torus. Alternatively, the keyboard shortcut SHIFT-A can be used.
- Open the Transform toolbar (small plus sign +).
- Keyframe the torus at 0° rotation (I key followed by ‘Rotation’ on the pop-up menu). Alternatively, the user can right-click on the rotation coordinates in

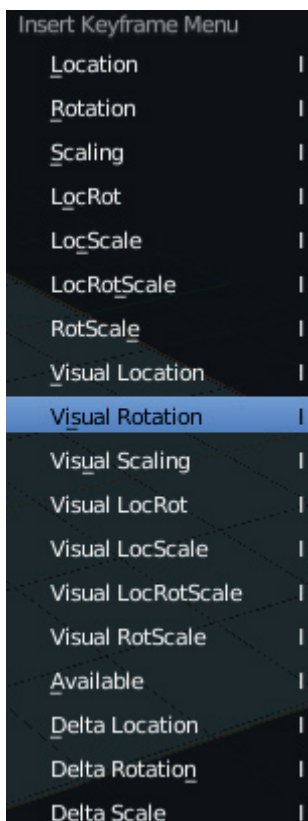


Figure 5.1. Pressing the I key allows the user to key an object, which inserts a keyframe recording a positional aspect of the object during a visualization.

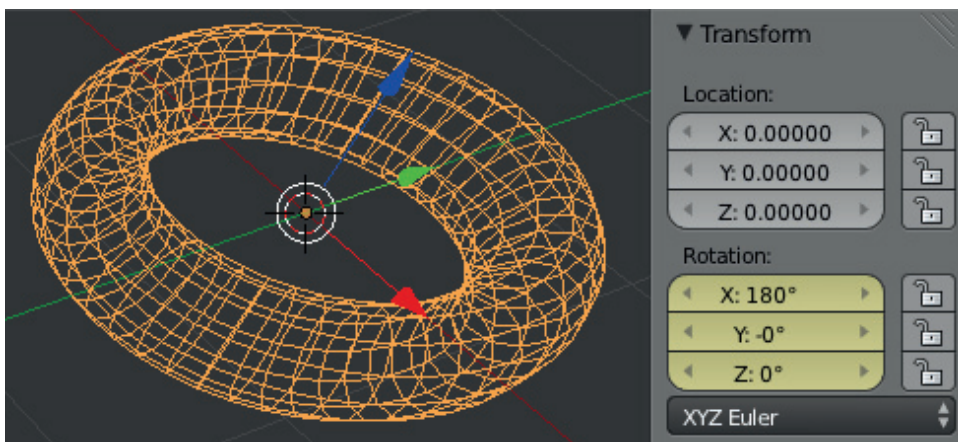


Figure 5.2. The rotation is keyframed (in yellow) during the animation of a torus mesh object.

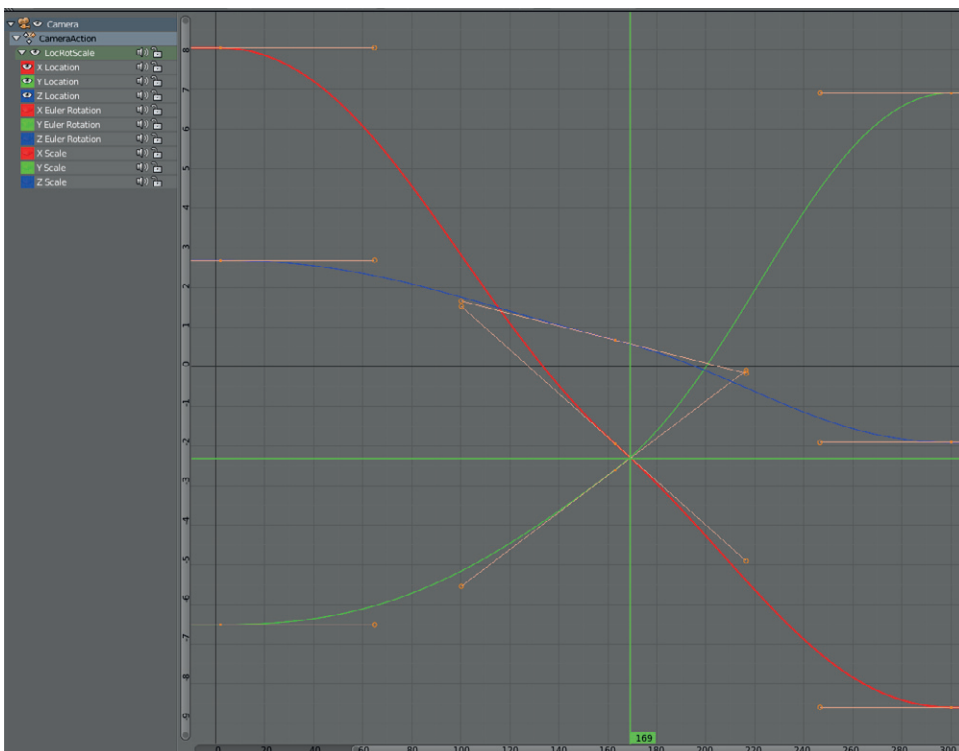


Figure 5.3. The Graph Editor allows the user to visually control the animation in Blender. By changing the shape of these curves a smooth animation during the visualization can be created.

the Transform toolbar and choose ‘Insert Keyframe’. A yellow vertical line will be placed on the Animation toolbar at the bottom of the window.

- Move the green animation slider to frame 150.
- Change the *X*-rotation value to 180 degrees in the Transform toolbar. Right-click on the value and insert another rotation keyframe.
- Finally, move the green animation slider to frame 300, change the *X*-rotation value to 360 degrees and right-click and insert another keyframe.
- Play the animation with the right pointing arrow play button on the Animation toolbar.

5.1.2 Example: using the graph editor

Note that the rotation of the torus ramps up in the early frames and then slows down at the end. If we want to cycle this animation to move smoothly, a constant speed is required. The Graph Editor can be used to adjust the rotation.

- Open the Graph Editor (figure 5.3) in the main window.
- At the bottom of the Graph Editor window, select View → View All.

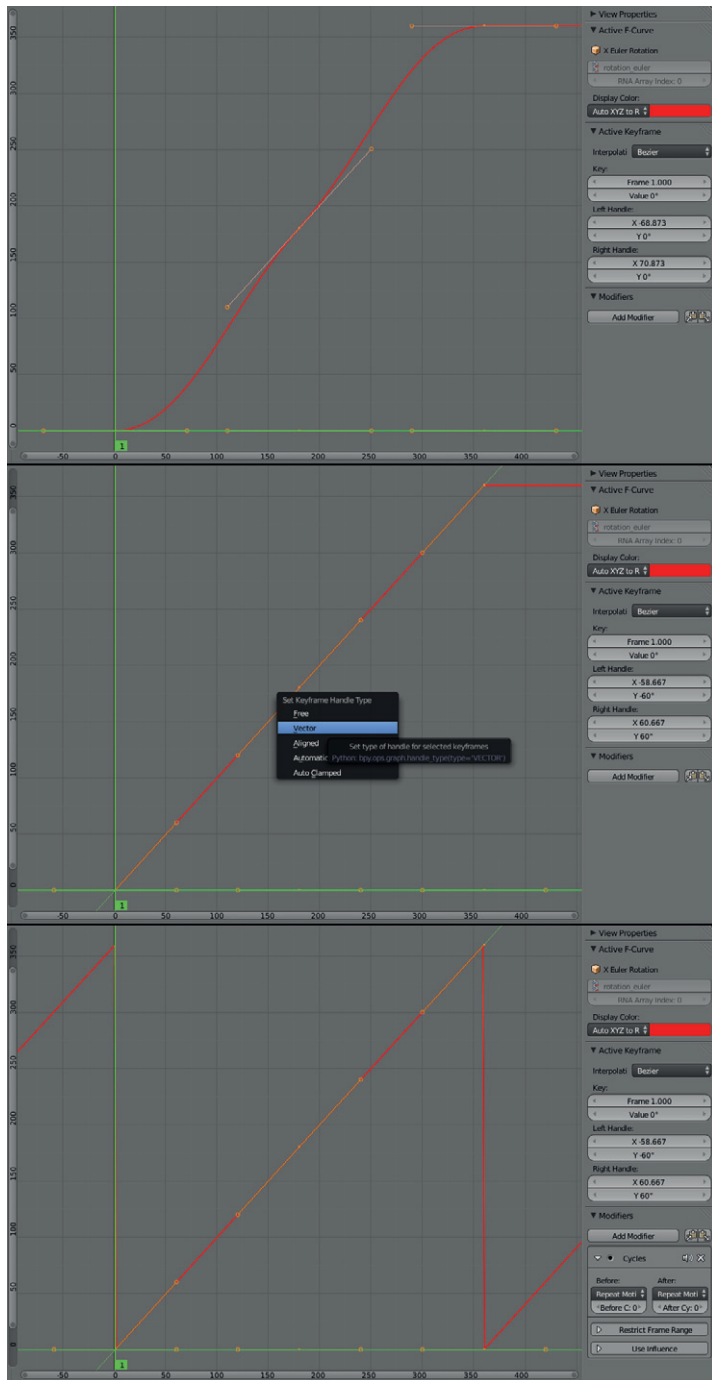


Figure 5.4. These three panels show vectorizing the rotation about the X-axis (red curve), to which a cyclic modifier is added to repeat the animation.

- Press the V key and set the ‘Keyframe Handle Type’ to ‘Vector’.
- Return to the 3D view port in the default view and replay the animation, which should run at a constant speed throughout the number of frame specified on the animation time line.

5.1.3 Example: adding a cyclic modifier

Building on the previous example, we can have Blender automatically repeat the visualization by adding a cyclic modifier to the Graph Editor.

- Click ‘Add Modifier’ on the right-hand side of the screen.
- Choose ‘Cycles’. A saw tooth waveform will appear in the graph editor.
- Figure 5.4 shows the progression of how to add a cyclic modifier and what the final graph should look like.

The torus will now repeat keyframed motion for the duration of the video rendering, independent of the length.

5.2 Frame rates and rendering output

Rendering video will depend on hardware resources and the goals of the visualization. For broadcast quality HD output, the highest resolution and video codec should be used. For trial renderings during visualization development, lower resolutions without all the rendering elements can be utilized.

The stamp feature in the Render tab is useful for recording metadata about the visualization—frame rates, render times, authors and what scene components were used in the final composite.

5.2.1 Output formats

Single frame images can be exported in a number of formats¹. For high quality video output, AVI JPEG or AVI RAW can be used as an initial step. The video can then be converted if needed. FFMPEG² is a useful video conversion utility for any operating system. The example below converts an AVI file to Quicktime Prores:

```
ffmpeg -i file.avi -vcodec prores -profile 3 -an file.mov
```

5.3 Node compositing

The Node Editor is a powerful Blender feature that allows a user to graphically access elements of the API, as well as combine different visualization layers into one final composite before feeding it to the rendering engine. The Node Editor can be accessed via the Editor drop-down selector in the lower left of the GUI.

¹ wiki.blender.org/index.php/Doc:2.6/Manual/Render/Output

² Download available at www.ffmpeg.org/.

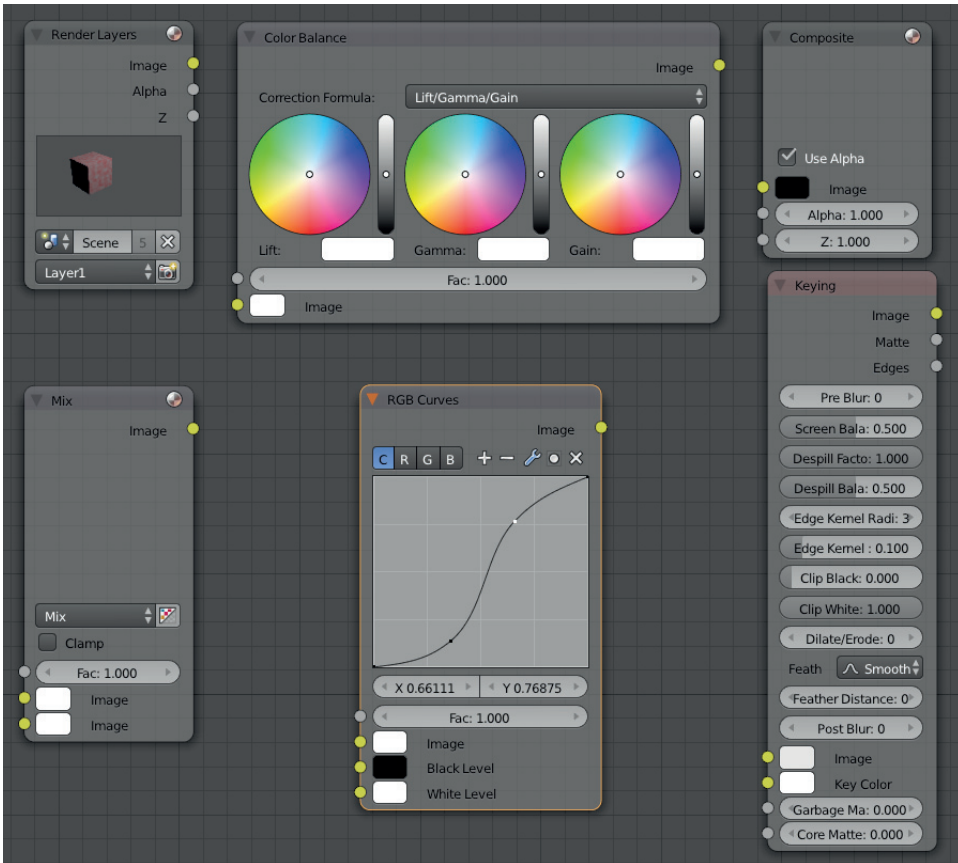


Figure 5.5. Example nodes used for compositing. Each node has imaging inputs and outputs which can lead to connecting nodes. Example nodes here give starting rendering layers, RGB color corrections, mixture nodes to composite layers and output nodes that connect to the rendering engine.

Nodes represent a particular feature or action in post-processing with inputs and output as well as keyword parameters that can be modified. A node is essentially a graphical representation of a Python function. Sample nodes are shown in figure 5.5. Once they are in place they are connected with lines from one output port to another node's input port (figure 5.6).

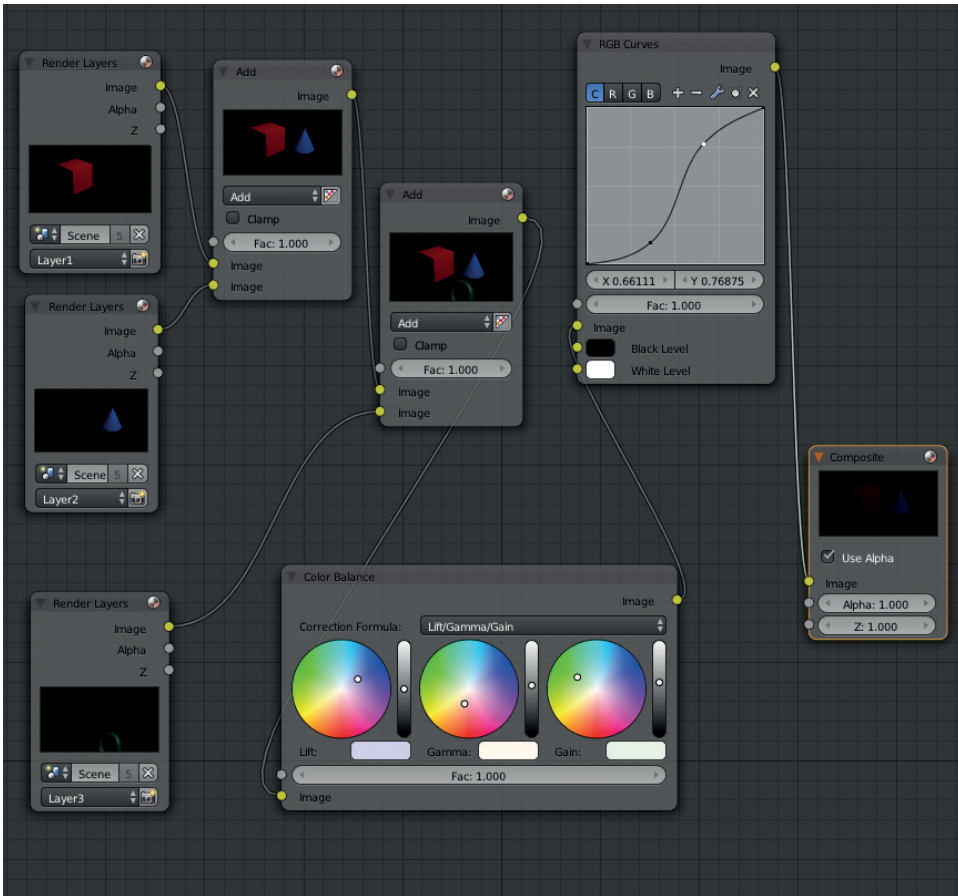


Figure 5.6. This window shows three render layers being combined and passed through a color balance and RGB node before being passed to the compositor. Nodes can be connected by drawing lines with the left mouse button from one output to another image input. Lines can be destroyed with the CTRL key and nodes can be deleted with the X key.

Chapter 6

Point of view: camera control

We will now discuss how to record our visualizations with the movement and tracking of a camera. In Blender the currently selected camera object presents the field of view that will be in the final render. Camera objects have a number of critical parameters: the X , Y , Z location and rotation of the camera, the vector pointing normal to the camera's 'film' plane, the field of view, the viewing projection and the aspect ratio. For example, in figure 6.1, the camera is located at position (1,2,3), rotated at 45° along the X -axis, with a 35° field of view in a perspective projection and an aspect ratio of 16:9 (1080p HD quality).

6.1 Projection differences: a matter of perspective

Consider the differences in projection in figure 6.2. The perspective projection shows all parallel lines along an axis converging to a single point on the horizon line [1]. This is what the human eye would see. The orthographic projection shows all lines along an axis to be parallel—what is conventionally used in a scientific plot. The use of either depends on how the user wishes to present their data in the visualization.

6.2 Camera keyframes

A camera object's location and rotation can be keyframed like any Blender object. In addition, a camera can be attached to a moving track and constrained to point toward a particular object, regardless of the camera's location. These properties are useful for tracking shots that can add a very dynamic view to an otherwise plain looking visualization.

6.2.1 Example: tracking to an object

At this point we introduce the idea of a constraint. A camera object can be constrained to track to another object—the normal of the 'film plane' of the camera will

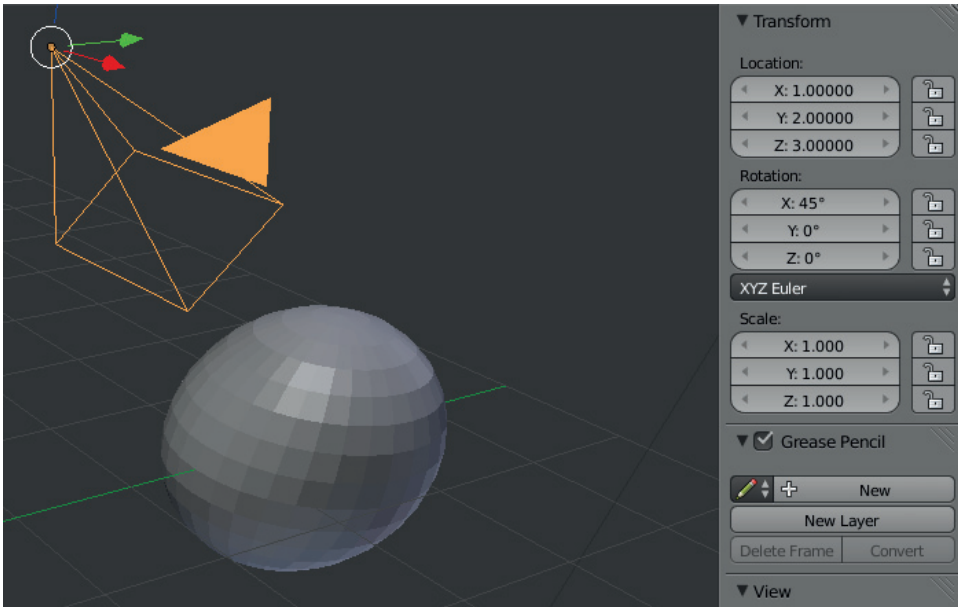


Figure 6.1. The 3D view port shows a camera pointing to a spherical mesh, with the Transform toolbar detailing the camera's exact position and rotation configuration.

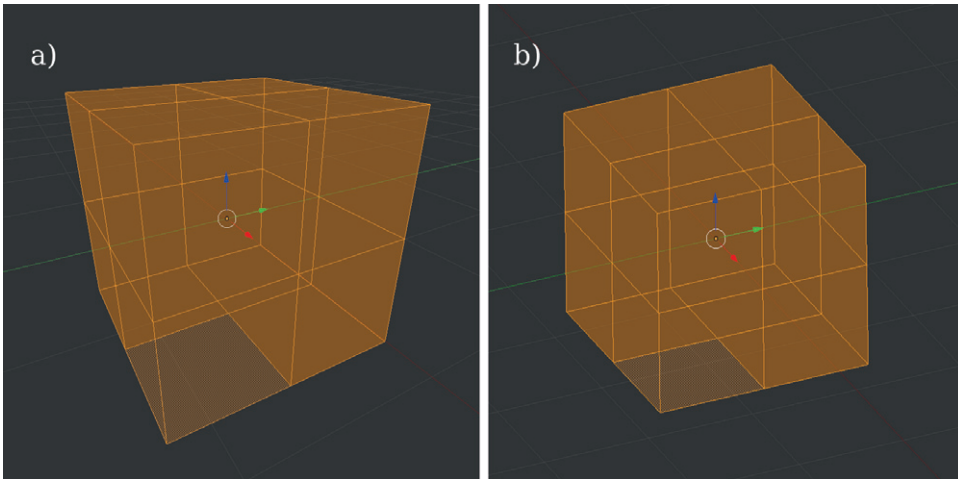


Figure 6.2. Two examples showing (a) a perspective view of a cube mesh and (b) the same mesh with an orthographic projection [2]. Copyright 2013 Brian R Kent, publications of the Astronomical Society of the Pacific.

point directly at the defined center of the other object. For many scenarios, it is useful to add an empty object to the visualization scene. The empty object does not render in the visualization and is represented by an orthogonal set of axes in the GUI (figure 6.3). In essence, the camera object will point toward wherever the empty

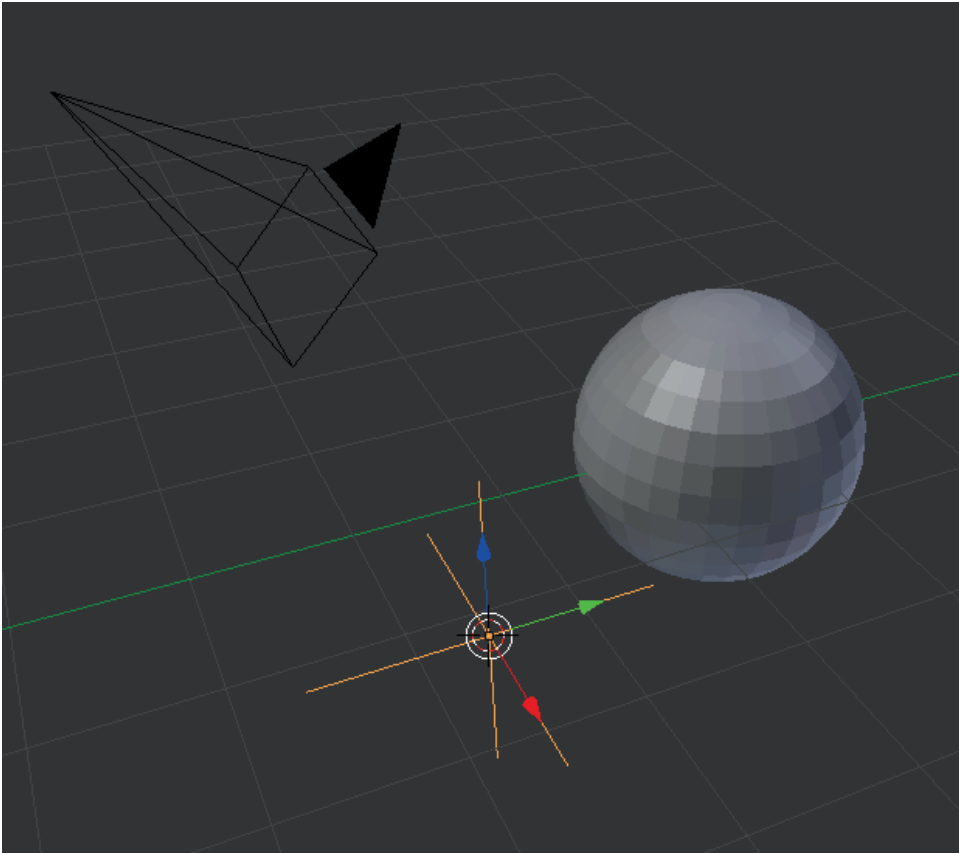


Figure 6.3. This view shows the location of an empty axis object. This type of object does not render and can be attached to other meshes so that the camera can follow and track it during a visualization animation.

object is located. An empty object can be attached to another object to aid in tracking a moving object during a visualization.

- Begin with the default scene upon starting Blender—a cube, camera and lamp.
- Add an empty object with Add → Empty → Plain Axes.
- Right-click to select the cube object *first* and then SHIFT–right-click again to select the empty object.
- Press CTRL–P to set the cube to the *parent* empty object.
- Right-click and choose the camera object.
- Click the Constraints tab on the right-hand side Properties panel.
- Choose ‘Track To’ and select the target as ‘Empty’.
- Select ‘To’ as $-Z$ and ‘Up’ as Y . This will correctly orient the upward and normal directions when looking through the camera field of view.
- A dashed blue line will now point directly from the camera object to the empty object, showing that no matter where the camera is moved, the tracking between the two objects will always hold during the visualization (figure 6.4).

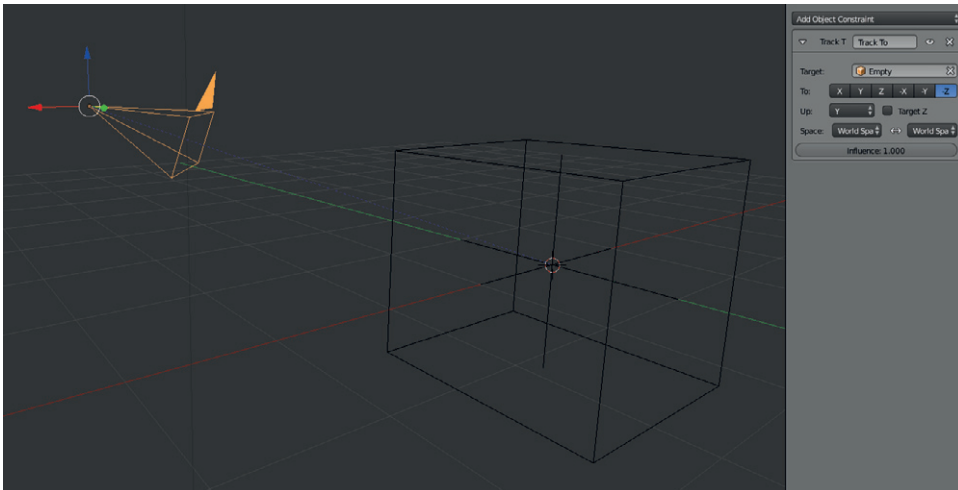


Figure 6.4. A camera object with a ‘Track To’ constraint applied such that the camera will always center its field of view on the empty object that is a parent of the rendered cube. The blue dashed line normal to the camera plane points from the camera object tracking to the object that it follows.

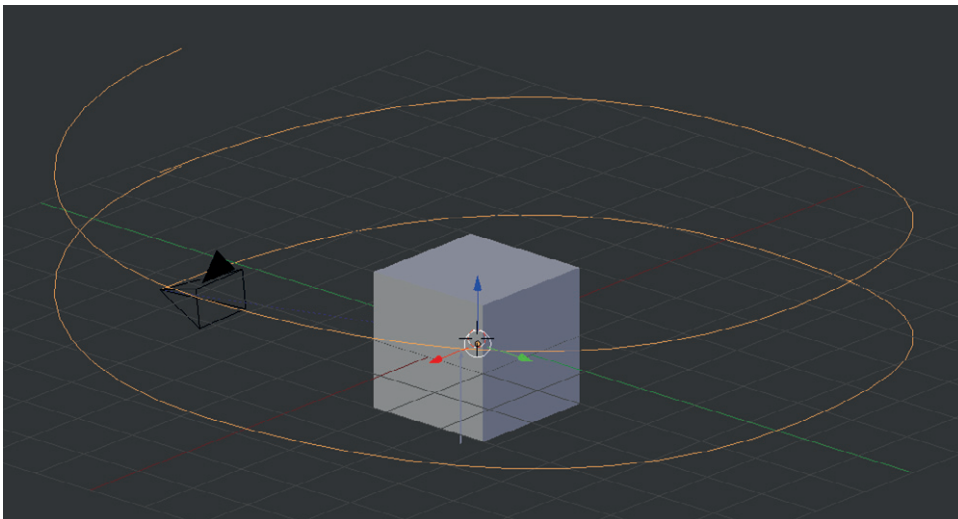


Figure 6.5. This panel shows a Bézier curve with a camera attached. The camera is locked to the curve and will follow it through the animation, while pointing to the cube mesh object.

6.2.2 Example: an object following a path

A camera object can be keyframed to move between two points (as described in section 6.2). However, it can also be set to follow a continuous smooth pre-determined path. The Bézier curve object (figure 6.5) can be used to construct this path. The camera object can then be locked to follow the path. This is extremely

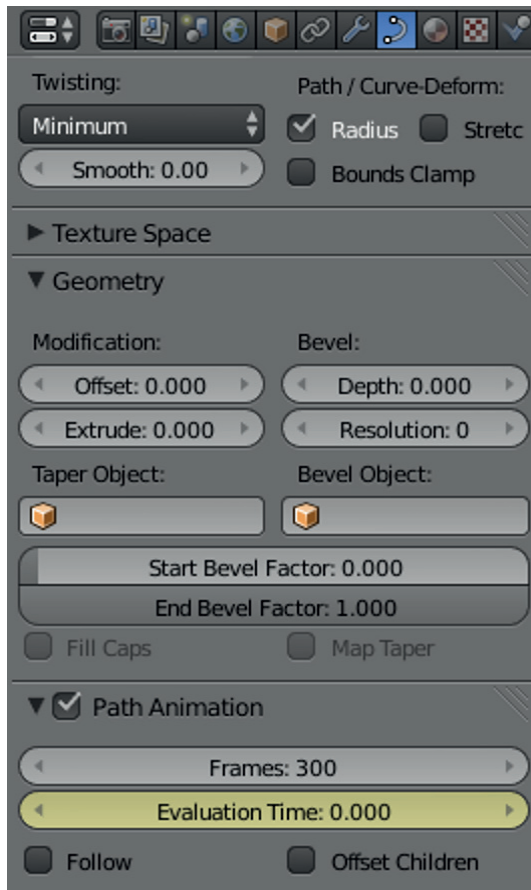


Figure 6.6. This window is found on the Properties panel and is used to keyframe the path animation for a Bézier curve. Once the path is keyframed, an object such as a camera can be locked to the path.

useful when one wishes to move the camera through a scene while tracking at the same time. The following scenario continues from the previous camera tracking example.

- Right-click to select the scene camera object—it will highlight orange.
- In the Transform panel, zero out any offsets with the position and rotation to be 0.0 and 0.0 degrees, respectively, for X , Y and Z .
- Click Add → Curve → Circle.
- Select the Object Data tab (figure 6.6) in the Properties panel. Click the check box for ‘Path Animation’, change the frames to 300.
- On the Animation toolbar, set the frame to zero and then, under path animation, right-click ‘Evaluation Time’ and ‘Insert Keyframe’.
- On the Animation toolbar, set the frame to zero and then under ‘Path Animation’, set the evaluation time to 300.0, right-click ‘Evaluation Time’ and ‘Insert Keyframe’. The parameterized unit of the evaluation time in this example is the frame number.

- Right-click to select the camera object again. Click on the ‘Constraints’ icon and add the object constraint called ‘Follow Path’. Change the forward axis to $-Z$ and the ‘Up’ vector to Y .
- Play the animation and observe that the camera object follows the path with the camera normal vector tangent to the circle.
- Add another object constraint for the camera—this time a ‘Track To’ constraint. Change the ‘To’ vector to $-Z$ and the ‘Up’ vector to Y .

The camera will now follow the Bézier circle path during the animation, while pointing at the cube. If the path is moved or scaled, the camera will follow.

Bibliography

- [1] Carlbom I and Paciorek J 1978 Planar geometric projections and viewing transformations *ACM comput. Surv.* **10** 465–502
- [2] Kent B R 2013 Visualizing astronomical data with Blender *Publ. Astron. Soc. Pac.* **125** 731–48

Chapter 7

Python scripting

The Blender API can be used in Python scripts to automate tasks and read in scientific data. The API allows the user to create and manipulate objects and data, edit properties and groupings, and control the animation sequence of a scientific visualization.

7.1 Scripting in blender

The Blender scripting interface is shown in figure 7.1. When a new script is started, a ‘Run Script’ button appears in the GUI—pressing this button will run the script. In addition, Blender scripts can be run from the command line via:

```
blender --python myscript.py
```

7.1.1 Blender and Python libraries

The following libraries will be commonly used with Blender scripts for scientific visualization. These typically will appear in the script preamble with ‘import’.

bpy. Application modules that allow control over the context, data types, operators and GUI applications.

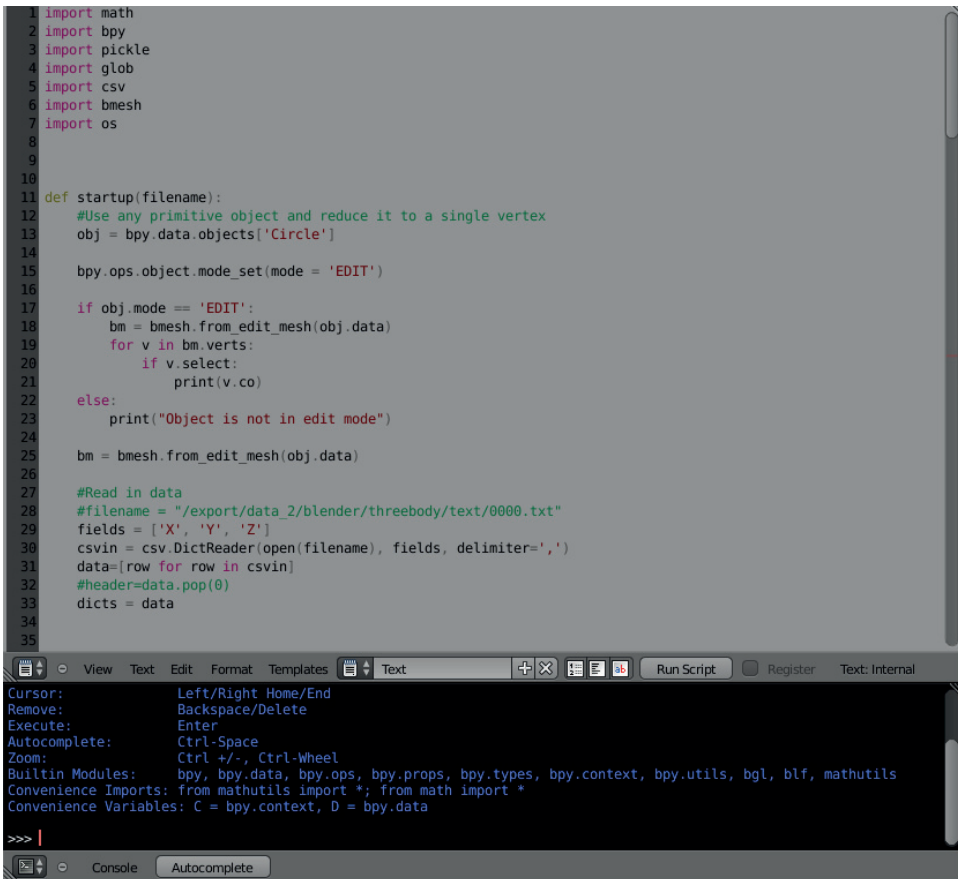
bmesh. A standalone module giving access to mesh objects, vertices and connecting segments.

csv. A useful Python module for reading formatted data.

glob. A standard Python module for traversing directory structure and obtaining lists of files to load.

os. An operating system interface useful for loading/finding paths where required libraries might be needed.

math. Basic mathematical operations and functions.



```

1 import math
2 import bpy
3 import pickle
4 import glob
5 import csv
6 import bmesh
7 import os
8
9
10
11 def startup(filename):
12     #Use any primitive object and reduce it to a single vertex
13     obj = bpy.data.objects['Circle']
14
15     bpy.ops.object.mode_set(mode = 'EDIT')
16
17     if obj.mode == 'EDIT':
18         bm = bmesh.from_edit_mesh(obj.data)
19         for v in bm.verts:
20             if v.select:
21                 print(v.co)
22     else:
23         print("Object is not in edit mode")
24
25     bm = bmesh.from_edit_mesh(obj.data)
26
27     #Read in data
28     #filename = "/export/data_2/blender/threebody/text/0000.txt"
29     fields = ['X', 'Y', 'Z']
30     csvin = csv.DictReader(open(filename), fields, delimiter=',')
31     data=[row for row in csvin]
32     #header=data.pop(0)
33     dicts = data
34
35

```

```

Cursor: Left/Right Home/End
Remove: Backspace/Delete
Execute: Enter
Autocomplete: Ctrl-Space
Zoom: Ctrl +/-, Ctrl-Wheel
Builtin Modules: bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, blf, mathutils
Convenience Imports: from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>> |

```

Figure 7.1. The editor and command line allow the user to control the Blender interface and functions via the Blender Python API.

7.1.2 Example: reading formatted ASCII data into blender

There are many different methods for reading formatted text into a Python dictionary. We employ dictionaries here for their clarity and convenience in terms of defining a Python data construct. The data for this short example will be in the following format:

```

#Name X Y Z
Point001 4.2 2.3 5.6
Point002 1.2 9.3 6.4
Point003 4.3 2.3 7.4
...

```

We can then read in the specified data with the following short Python script.

```
import bpy
import bmesh
import csv

'''Read in the data according to the defined list of fields
   Remove the final header if needed.
   Define the table as an empty list
'''
fields = ['Name', 'X', 'Y', 'Z']
csvin = csv.reader(open(f))
data=[row for row in csvin]
header=data.pop(0)
table = []

'''For each row in the data list, map the fields and
   data elements to a dictionary
   Append the dictionary to the table list.
'''
for row in data:
    datarow = map(float,row[0].split())
    rowdict=dict(zip(fields, datarow))
    table.append(rowdict)
```

7.1.3 Example: adding data points to an object as vertices

For this example we create a cube object mesh (actually, any mesh will work) and remove all but one vertex. That vertex will then be duplicated at all the positions of the imported ASCII text file.

- Create a UV-cube mesh by clicking Add → Mesh → Cube. Alternatively, the keyboard shortcut SHIFT+A can be used.
- Press TAB to enter Mesh Edit mode. CTRL+right-click on all vertices but one and press X to delete them.
- Run the following script, which will switch the GUI to Edit mode and duplicate the vertex at each X, Y, Z point loaded from the ASCII table:

```
import bpy
import bmesh
import csv

#Switch to edit mode
bpy.ops.object.mode_set(mode = 'EDIT')
```



```

obj = bpy.data.objects['Cube']
mesh = obj.data
bm = bmesh.from_edit_mesh(obj.data)
rowcount = 0
for i in range(0, len(bm.verts)-1):
    try:
        bm.verts[i].co.x = table[rowcount]['X']
        bm.verts[i].co.y = table[rowcount]['Y']
        bm.verts[i].co.z = table[rowcount]['Z']

        rowcount += 1
        bmesh.update_edit_mesh(obj.data)
    except:
        pass

```

7.1.4 Example: animating multiple time steps

For simulations where the data may be a changing 3D time-series, inserting keyframes between each time step can animate the data. Keyframe insertion and the animation playback toolbar can also be controlled via the Blender API. This example reads in a series of formatted ASCII text files, putting together the previous two code blocks.

```

import bpy
import bmesh
import csv
import glob

'''This line will change depending on whether Linux,
Mac OS, or Windows is used
'''
files = glob.glob(mydirectory + '/*')

for f in files:
    #Switch to edit mode
    bpy.ops.object.mode_set(mode = 'EDIT')

    print(Frame: + str(f))
    bpy.context.scene.frame_set(framecount)
    fields = ['Name', 'X', 'Y', 'Z']
    csvin = csv.reader(open(f))
    data=[row for row in csvin]
    header=data.pop(0)
    table = []

```

```
for row in data:
    datarow = map(float,row[0].split())
    rowdict=dict(zip(fields, datarow))
    table.append(rowdict)

obj = bpy.data.objects['Cube']
mesh = obj.data
bm = bmesh.from_edit_mesh(obj.data)
rowcount = 0
for i in range(0,len(bm.verts)-1):
    try:
        bm.verts[i].co.x = table[rowcount]['X']
        bm.verts[i].co.y = table[rowcount]['Y']
        bm.verts[i].co.z = table[rowcount]['Z']

        rowcount += 1
        bmesh.update_edit_mesh(obj.data)
    except:
        pass

bpy.ops.anim.keyframe_insert(type='Location')
```

Chapter 8

Projects and 3D examples

It is best to expand upon the overview of information presented in this book with a collection of examples. Each example will use different parts of the Blender interface to implement a different style of scientific visualization. Example blend files, data files and videos will be provided to better illustrate some of these features.

8.1 3D scatter plot

A 3D scatter plot can be useful in showing trends in multiple parameters or the locations of objects in 3D space. In this project stars from the *Hipparcos* project will be displayed in 3D [1]. The concepts used will be:

- Reading in formatted data with the Blender Python API.
- Setting up a Cartesian grid.
- Moving the camera around the dataset.

The following steps will set up this visualization.

- Add a plane with Add → Mesh → Plane. Scale the plane with the S key and press TAB to enter Mesh Edit mode. Subdivide the plane five times and press TAB one more time to again return to Object mode.
- Add a material to the plane mesh on the Properties panel and set the type to ‘Wire’. Choose a color that contrasts well with the background—blue on black usually works well.
- Set the World tab background horizon color on the Properties panel to black.
- Add a simple mesh with Add → Mesh → Circle. Press TAB to enter Mesh Edit mode, SHIFT select all but one of the vertices and press X to remove them. Press TAB one more time to go back to Object mode.

We then import the *Hipparcos* catalog with the following Python script:

```
import bpy
import math
import bmesh
import csv

obj = bpy.data.objects['Circle']

if obj.mode == 'EDIT':
    bm = bmesh.from_edit_mesh(obj.data)
    for v in bm.verts:
        if v.select:
            print(v.co)
else:
    print("Object is not in edit mode")

bm = bmesh.from_edit_mesh(obj.data)

#Read in Hipparcos data
filename = 'hygxyz.csv'

fields = ['StarID', 'HIP', 'HD', 'HR', 'Gliese', 'BayerFlamsteed',
          'ProperName', 'RA', 'Dec', 'Distance', 'PMRA', 'PMDec', 'RV', 'Mag',
          'AbsMag', 'Spectrum', 'ColorIndex', 'X', 'Y', 'Z', 'VX', 'VY', 'VZ']

reader = csv.DictReader(open(filename), fields, delimiter=',')

dicts = []

#Skip first header line
next(reader)
for row in reader:
    dicts.append(row)

#Add in vertex elements with XYZ coordinates at each row

for row in dicts:
    xpos = float(row['X'])/100.0
    ypos = float(row['Y'])/100.0
    zpos = float(row['Z'])/100.0
    bm.verts.new((xpos,ypos,zpos))

bmesh.update_edit_mesh(obj.data)
```

- The data should load quickly. We can now add a material to the *Hipparcos* data points.
- Select the data points and click the Materials tab on the Properties panel.
- Select ‘Halo’ and change the size value to 0.005 and hardness value to 100. A white or light yellow color can be used to color the points.

We can now use the default camera object to point at an empty object as in section 6.2.1.

- Add an empty object with Add → Empty → Plain Axes for the camera to track (figure 8.1).
- Right-click to choose the camera object and set the position and rotation values on the Transform toolbar to zero.
- Click the Constraints tab on the right-hand side Properties panel.
- Choose ‘Track To’ and select the target as ‘Empty’.
- Select ‘To’ as $-Z$ and ‘Up’ as Y . This will correctly orient the upward and normal directions when looking through the camera field of view.

Animate the visualization by keyframing the camera.

- This animation will be 20 s long at 30 frames per second. Therefore set the number of frames to 600 and set the current frame to 1.
- Right-click to select the camera and press the I key to keyframe the position and rotation of the camera.
- On the Animation toolbar, set the current frame to 600. Move the camera in the 3D view port to a different location and orientation.
- Keyframe the camera position and rotation one final time with the I key.

The visualization can now be exported in a 1080p HD video.

- On the Render tab in the Properties panel select *HDTV 1080p* and set the frame rate to 30 frames per second.
- Set the output to AVI JPEG, quality 100 percent, and specify a unique filename. Click the ‘Animation’ button at the top of the tab to render the visualization. A frame from the final animation is shown in figure 8.2.

8.2 *N*-body simulation

For this example we will provide the reader with *N*-body data from a galaxy collision simulation. The idea is to expand upon the initial scatter plot example by putting the *N*-body points into shape keys to improve performance and then iterate over each snapshot to complete the animation. The concepts used will be:

- Reading in multiple files of formatted data with the Blender Python API.
- Setting up a Cartesian grid.
- Associating each temporal snapshot with a shape key.
- Adding halo materials to the data points.
- Moving the camera to track to the centroid of a galaxy until a specified time during the galaxy collision.

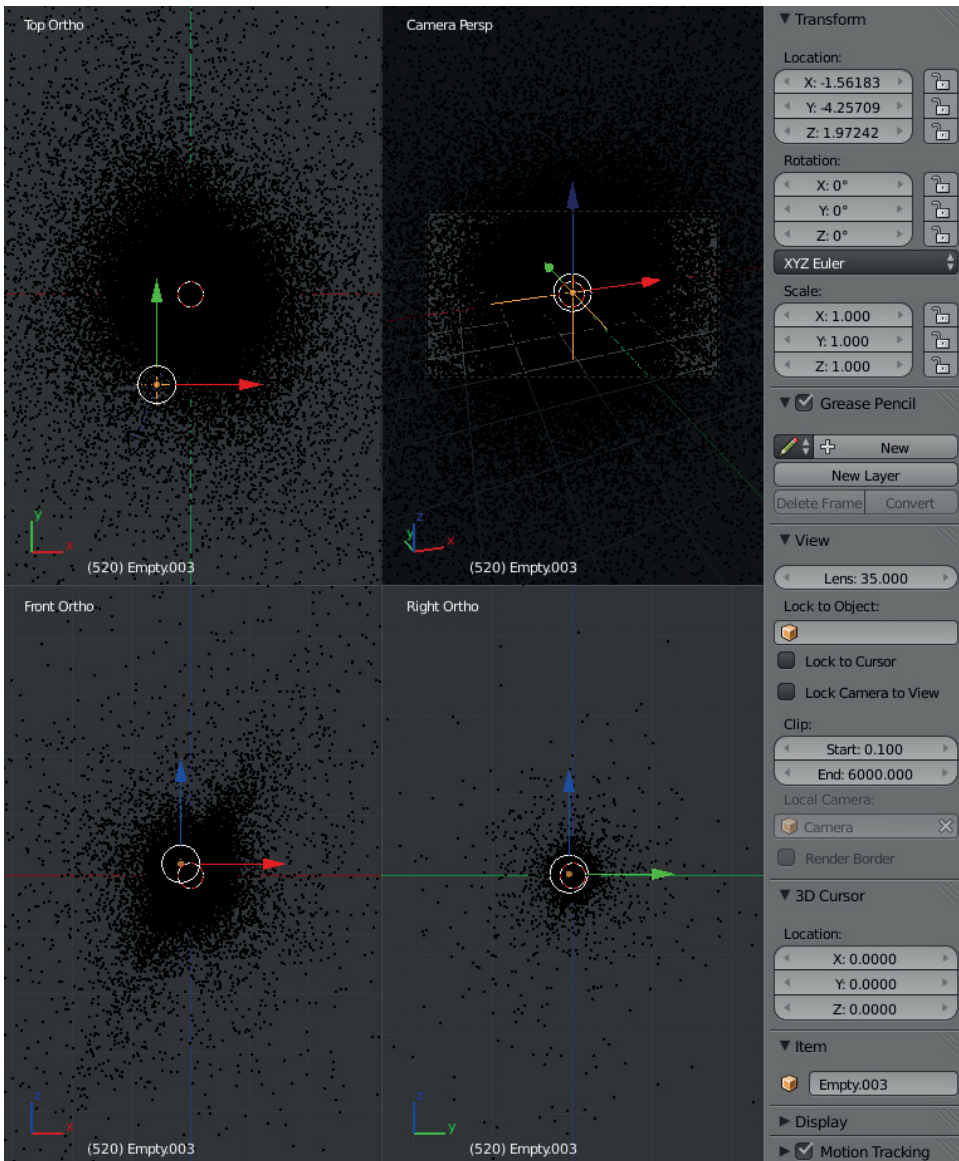


Figure 8.1. The set-up for an empty tracking object. The camera will point to this object during the fly through of the *Hipparcos* catalog.

This example of a galaxy simulation was generated with GADGET-2 [2], an extremely useful N -body and smoothed particle hydrodynamics package for astrophysics. Its capabilities range from large scale structure cosmology simulations to galaxy collisions. The simulation in this example is run for approximately 1100 time steps for a total run time of two billion years. Each large spiral disk galaxy has

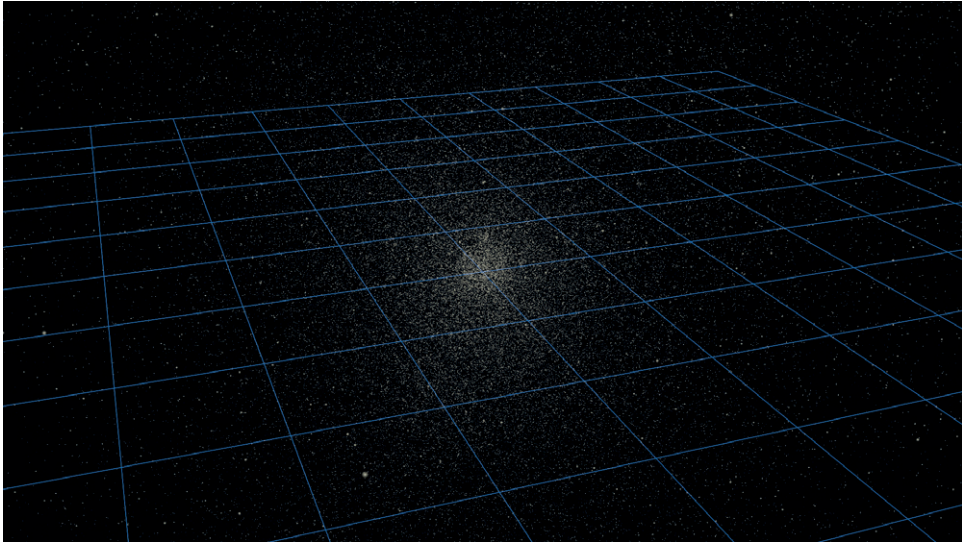


Figure 8.2. A 3D rendering of the *Hipparcos* catalog, utilizing a halo material, a subdivided grid and camera tracking.

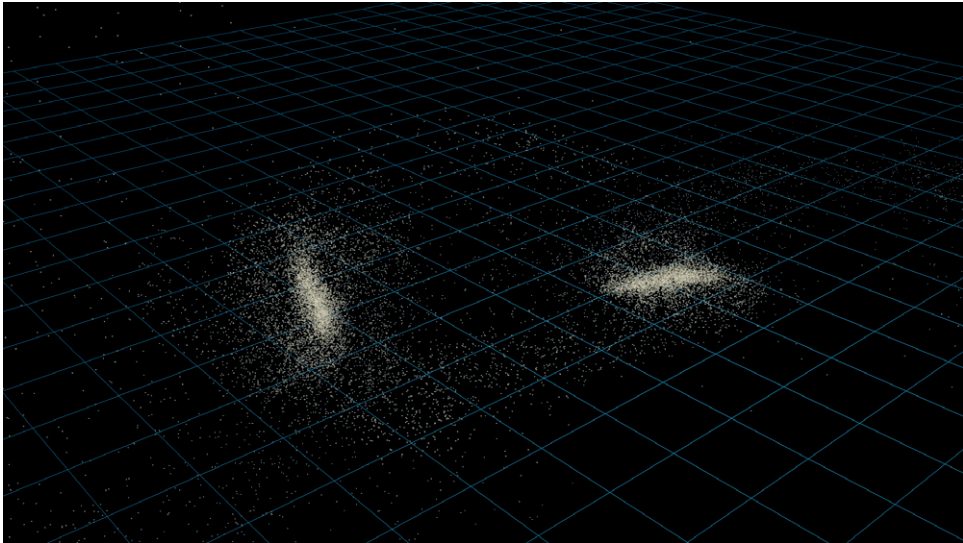


Figure 8.3. A 3D rendering of an *N*-body simulation showing a collision between two galaxies.

10 000 disk particles and 20 000 halo particles with Milky Way scale lengths and masses (2.1 kiloparsecs and 10^9 solar masses). The particle position data are shown as a single vertex with a halo material. Each snapshot is keyframed as a camera object is flown along a Bézier curve. A frame from the animation is shown in figure 8.3.

The ASCII snapshot files produced by the simulation are in the following simple X, Y, Z comma-separated-value format:

```
-44.4367,-44.1049,-0.539139
-94.6014,-74.8028,-0.5839
-70.9394,-77.0528,-0.357556
-58.1572,-57.4876,-0.917166
-63.3899,-61.3828,-0.793917
...
```

First a shape key needs to be created in the script with (for this simulation) 20 000 vertex points:

```
import bpy
import bmesh
#Switch to object mode
bpy.ops.object.mode_set(mode = 'OBJECT')

#Create initial Basis Shape Key
bpy.ops.object.shape_key_add()
#Create initial Shape Key
bpy.ops.object.mode_set(mode = 'OBJECT')
bpy.ops.object.shape_key_add()
bpy.data.shape_keys[0].key_blocks["Key 1"].name = 'Key 1'

bpy.data.shape_keys[0].key_blocks["Key 1"].value = 1

bpy.data.shape_keys[0].key_blocks["Key 1"].
    keyframe_insert("value", frame=0)
```

The snapshot data files are then read in using the `glob` and `csv` Python module. The vertices are added to a single vertex circle object just like the mesh created in the 3D scatter plot example.

```
import glob
import csv
import bpy
import bmesh

files = glob.glob('*.txt')
files.sort()
```



```

for f in files:
    fields = ['X', 'Y', 'Z']
    csvin = csv.DictReader(open(f), fields, delimiter = ',')
    data = [row for row in csvin]

dicts = data
obj = bpy.data.objects['Circle']
mesh = obj.data
bm = bmesh.from_edit_mesh(obj.data)
rowcount = 0
for i in range(0, len(bm.verts)-1):
    try:
        bm.verts[i].co.x = float(dicts[rowcount]['X'])/10.0
        bm.verts[i].co.y = float(dicts[rowcount]['Y'])/10.0
        bm.verts[i].co.z = float(dicts[rowcount]['Z'])/10.0

        rowcount += 1

        bmesh.update_edit_mesh(obj.data)
    except:
        pass

```

Add in the keyframes for each shape key:

```

#Key a single frame per data file
for j in range(1, frames+1):
    for keyname in bpy.data.shape_keys[0].key_blocks.keys():
        if keyname != os.path.basename(files[framecount-1]):
            bpy.data.shape_keys[0].key_blocks[keyname].value = 0
            bpy.data.shape_keys[0].key_blocks[keyname].
                keyframe_insert("value", frame = framecount)
    framecount += 1

```

We now change the vertices to have a material texture of a small Gaussian sphere.

1. With the circle object selected, click on the Materials tab in the Properties panel to create a new material with the 'Halo' option.
2. Change the halo size to 0.05.
3. Change the hardness to 100.
4. Modify the desired halo color to your liking.

Since both galaxies are moving, we will keyframe an empty object to track the centroid of one galaxy before the collision. As the collision evolves the empty object will slow to a stationary position (figure 8.4). The camera object will also move and track the empty object during the entire animation.

1. Add an empty object with Add → Empty → Plain Axes for the camera to point toward.

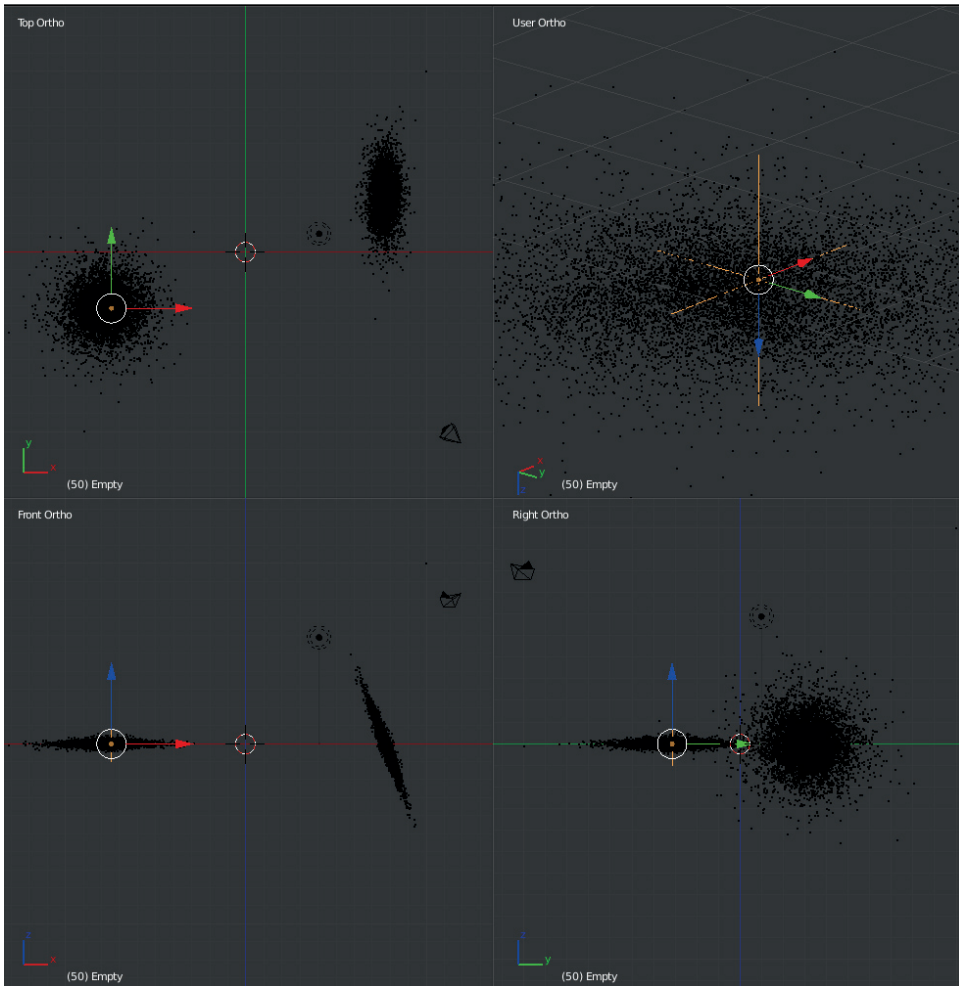


Figure 8.4. An empty object is placed at the centroid of one of the galaxies and keyframed to move during the camera tracking.

2. With the current frame set to zero, place the empty object near the center of a galaxy and keyframe the position with the I key.
3. Play the animation through the first part of the collision and then stop. Move the empty object to a new position near the collision centroid and add another keyframe.
4. Right-click to choose the camera object and set the position and rotation values on the Transform toolbar to zero.
5. Click the Constraints tab on the right-hand side Properties panel.
6. Choose 'Track To' and select the target as 'Empty'.
7. Select 'To' as $-Z$ and 'Up' as Y . This will correctly orient the upward and normal directions when looking through the camera field of view.

Snapshots and different camera views from the final animations are shown in figure 8.5.

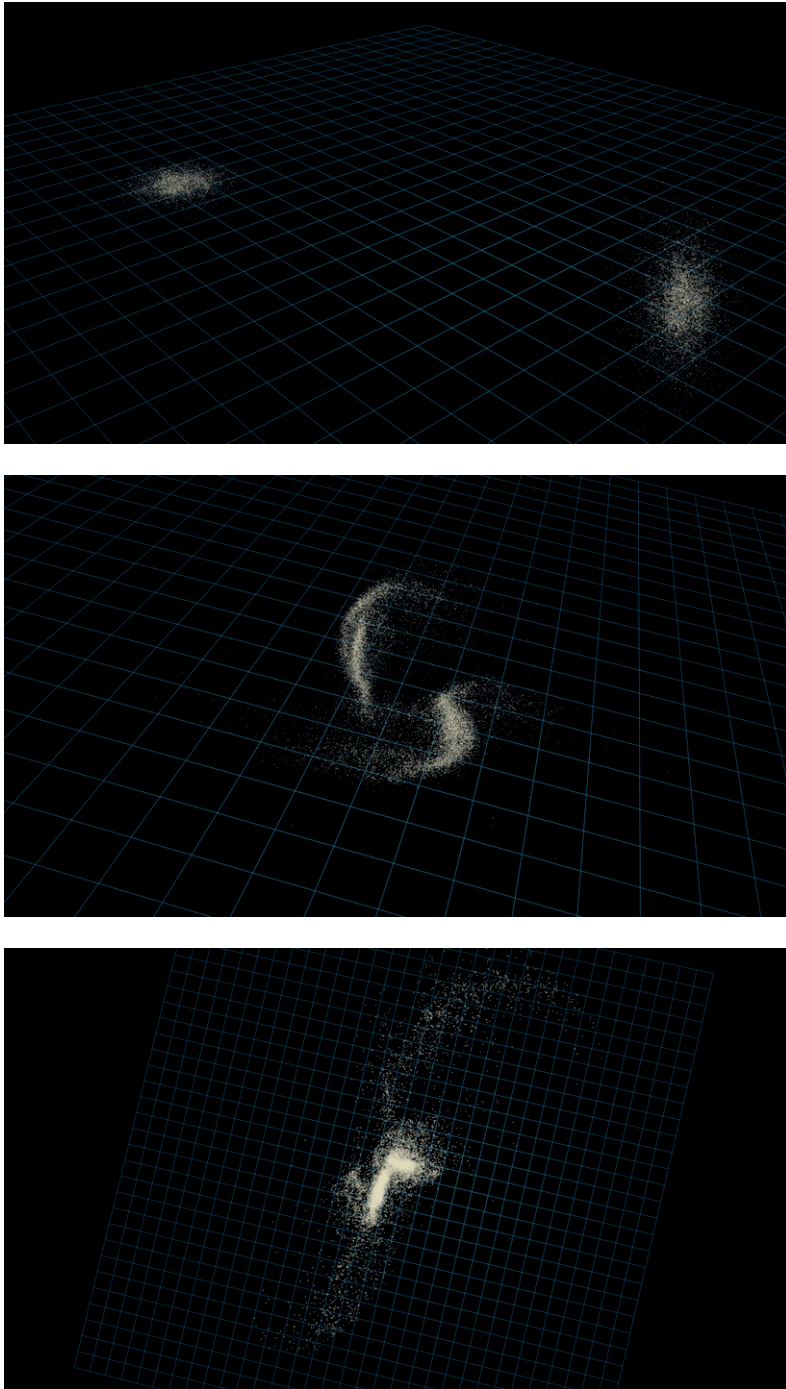


Figure 8.5. Frames from an N -body galaxy collision simulation.

8.3 Magnetic fields

In this project we will produce a 3D visualization of the magnetic potential for a loop of electric current. The concepts used will be:

- Evaluating the appropriate equations in Python for the magnetic potential.
- Add a model for the current loop using extrusion.
- Plot the contours using vertex editing mode.
- Rotate the contours due the azimuthal symmetry of the equation.
- Add a colored wire material for the potential contours.

From classical electrodynamics we know that the current density \mathbf{J} has an azimuthal component in a ring of radius a . Reviewing these basic equations we have [3]:

$$\mathbf{J} = -J_\phi \sin \phi' \mathbf{i} + J_\phi \cos \phi' \mathbf{j}. \quad (8.1)$$

The solution for the vector potential is written as:

$$\mathbf{A}(\mathbf{x}) = \frac{\mu_0}{4\pi} \int \frac{\mathbf{J}(\mathbf{x}')}{|\mathbf{x} - \mathbf{x}'|} d^3x'. \quad (8.2)$$

Due to the symmetry of the configuration, we can write the azimuthal component A_ϕ as:

$$A_\phi(r, \theta) = \frac{\mu_0}{4\pi} \frac{4Ia}{\sqrt{a^2 + r^2 + 2ar \sin \theta}} \left[\frac{(2 - k^2)K(k) - 2E(k)}{k^2} \right] \quad (8.3)$$

where $K(k)$ and $E(k)$ are complete elliptic integrals. The solution can be expanded in the following approximation:

$$A_\phi(r, \theta) = \frac{\mu_0 I a^2 r \sin \theta}{4(a^2 + r^2)^{3/2}} \left[1 + \frac{15a^2 r^2 \sin^2 \theta}{8(a^2 + r^2)^2} + \dots \right]. \quad (8.4)$$

We will evaluate this expression, compute the contours and use vertex editing mode to draw the contours. Because the scenario is azimuthally symmetric, we can use the Blender spin tool to rotate the magnetic potential contours about the Z -axis. Finally, we will add the current loop and animate the visualization. The data from 8.4 are generated and read in via the following Blender Python script:

```
import pickle
import numpy
import bpy
import bmesh
import csv
```

```

obj = bpy.data.objects['MagneticField']

bpy.ops.object.mode_set(mode = 'EDIT')

bm = bmesh.from_edit_mesh(obj.data)

#Read in contour data
filename = 'magnetic.txt'
fields = ['X', 'Y']
csvin = csv.reader(open(filename))
data = [row for row in csvin]
dicts = []

for row in data:
    datarow = map(float,row[0].split())
    rowdict = dict(zip(fields, datarow))
    dicts.append(rowdict)

for row in dicts:
    try:
        xpos = row['X']
        ypos = row['Y']
        bm.verts.new((xpos,ypos,0.0))
        bmesh.update_edit_mesh(obj.data)
        #Note that the minus 2 is the number of points to connect
        bm.edges.new((bm.verts[i] for i in range(-2,0)))
    except:
        print("Skipping")

bmesh.update_edit_mesh(obj.data)

bpy.ops.object.mode_set(mode='OBJECT')

```

The ring of current is created using the extrude tool.

- Add a mesh with Add → Mesh → Circle.
- Enter Mesh Edit mode with the TAB key.
- Extrude the vertices with the E key followed by a single mouse click.
- Scale the new set of vertices with the S key and then return to Object mode with the TAB key.
- On the Transform toolbar set the dimensions to 1.5 units in size to match the values used in the model.
- Color the current loop with a blue material.

Due to azimuthal symmetry, the contours can be extended via the spin tool (figure 8.6).

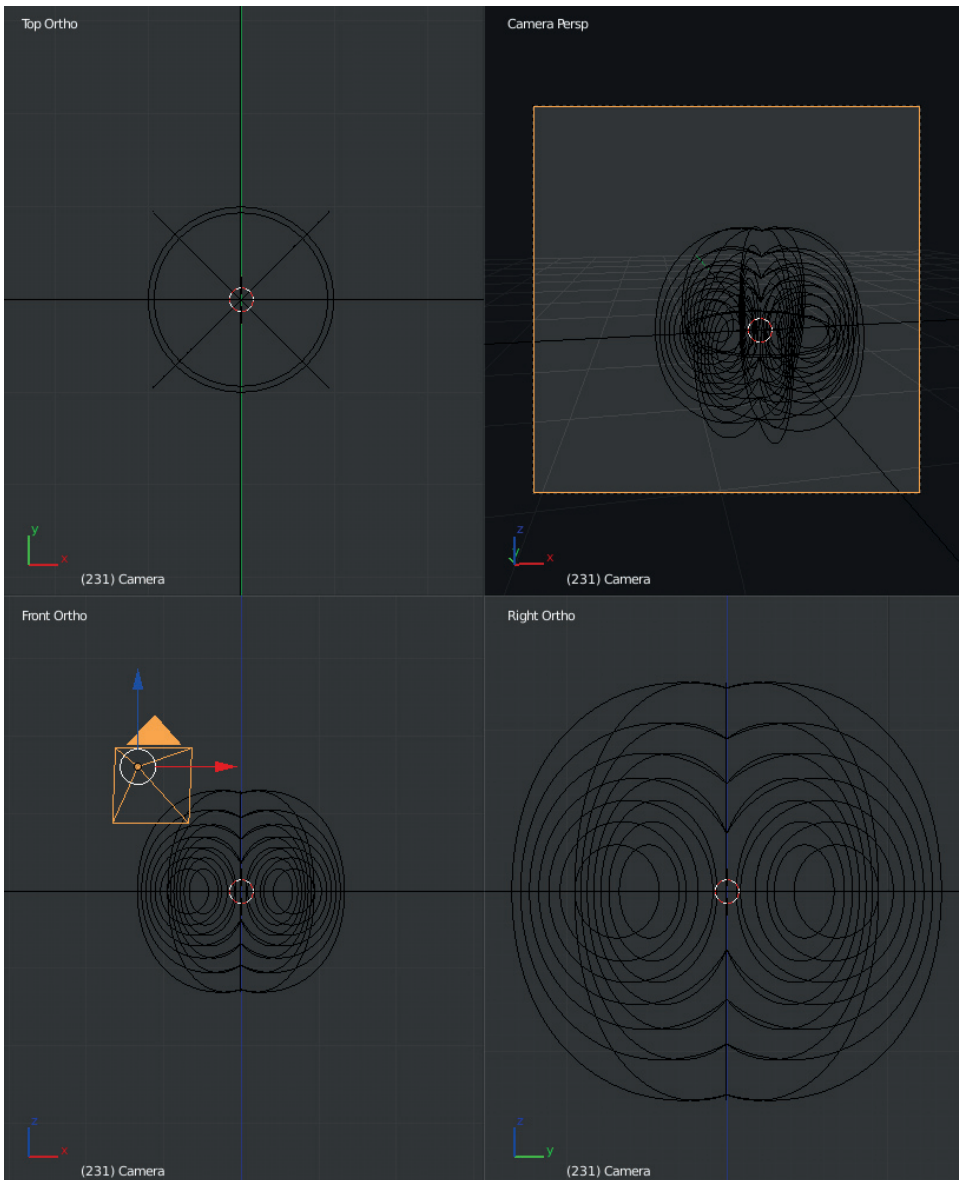


Figure 8.6. Quad view along each axis along with the camera view for the magnetic potential.

- Select the magnetic potential contour mesh.
- Enter Mesh Edit mode with the TAB key.
- Choose ‘Spin’ from the Mesh toolbar on the left-hand side.
- Set the steps to 6, angle to 360, center to zero and Z-axis to 1.0. Check the duplicate box and this will create a set of azimuthally symmetric contours.
- Exit Mesh Edit mode with the TAB key and color the potential contours loop with a red material (figure 8.7).

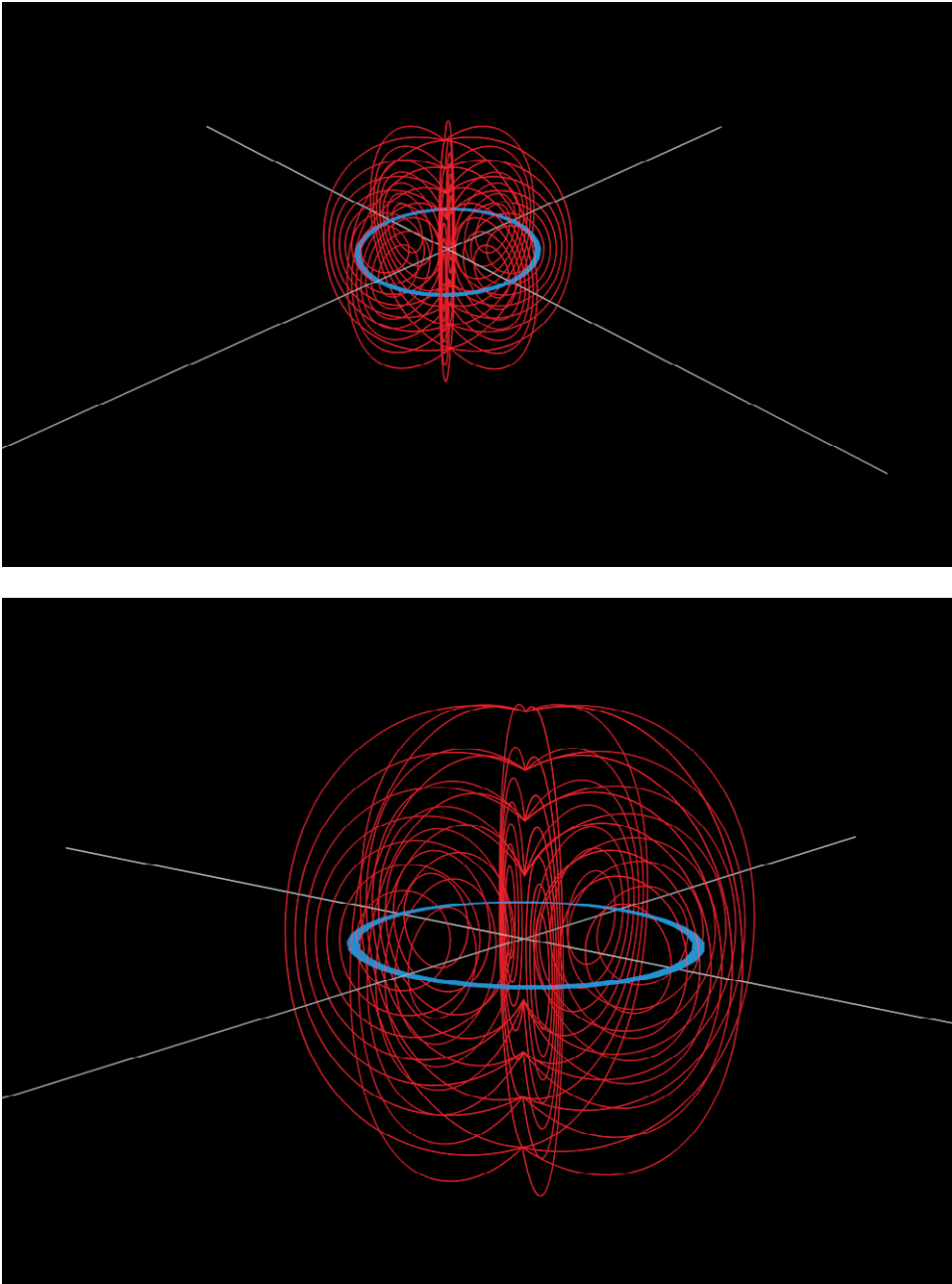


Figure 8.7. A 3D rendering of the magnetic potential for a current loop.

8.4 Lagrangian equilibrium and zero-velocity curves

We can compute and map the Lagrangian equilibrium points and zero-velocity curves for a two mass system¹. This project will use the following concepts:

- Render the curves as 3D surfaces of revolution.
- Introduce the transparency properties for Blender materials.
- Duplicate mesh models to calculated positions.

The formulation considered here is outlined in chapter 3 of *Solar System Dynamics* [4]. We consider two masses in circular orbits of masses m_1 and m_2 about a common center of mass at the origin O . The sum of the masses and the constant separation between them are both unity. q is the ratio of the masses and we define:

$$q = \text{ratio of masses } m_1/m_2 \quad (8.5)$$

$$\bar{\mu} = q/(1 + q) \quad (8.6)$$

$$\mu_1 = 1 - \bar{\mu} \quad (8.7)$$

$$\mu_2 = \bar{\mu}. \quad (8.8)$$

The location of the triangular Lagrangian points L_4 and L_5 can be solved for:

$$L_4 = \left(\frac{1}{2} - \mu_2\right)\hat{x} + \sqrt{\frac{3}{2}}\hat{y} \quad (8.9)$$

$$L_5 = \left(\frac{1}{2} - \mu_2\right)\hat{x} - \sqrt{\frac{3}{2}}\hat{y}. \quad (8.10)$$

The collinear Lagrangian equilibrium points are derived as:

$$L_{1r2} = \alpha - \frac{1}{3}\alpha^2 - \frac{1}{9}\alpha^3 - \frac{23}{81}\alpha^4 \quad (8.11)$$

$$L_{2r2} = \alpha + \frac{1}{3}\alpha^2 - \frac{1}{9}\alpha^3 - \frac{31}{81}\alpha^4 \quad (8.12)$$

$$L_{3r1} = 1 - \frac{7}{12} \frac{\mu_2}{\mu_1} + \frac{7}{12} \left(\frac{\mu_2}{\mu_1}\right)^2 - \frac{13223}{20736} \left(\frac{\mu_2}{\mu_1}\right)^3. \quad (8.13)$$

We can further refine these collinear Lagrangian points by using the Newton–Rhapson method to solve for the vectors $\mathbf{r1}$ and $\mathbf{r2}$. These examples are included at the book website². Next we provide code that solves for these points.

¹ For excellent research on this topic using simulations of cataclysmic variable stars, visit <http://astro.tamuc.edu/wood/>.

² www.cv.nrao.edu/~bkent/blender/


```

import math
import os
import csv
import numpy as np
from scipy.optimize import fsolve
import numpy as np
from scipy.optimize import newton
import matplotlib.pyplot as plt

def dPotential1(r, *mus):

    errflg = 0
    #Unpack tuple
    mu1, mu2 = mus

    T1 = 1. - r
    T2 = T1 - 1.0/T1**2
    T3 = r - 1.0/r**2

    dU = mu1*T2 - mu2*T3 # Equation 3.73

    return dU

def dPotential2(r, *mus):

    errflg = 0

    #Unpack tuple
    mu1, mu2 = mus

    T1 = 1. + r
    T2 = T1 - 1.0/T1**2;
    T3 = r - 1.0/r**2;

    dU = mu1*T2 + mu2*T3 #Equation 3.85

    return dU

def dPotential3(r, *mus):

    errflg = 0

    #Unpack tuple
    mu1, mu2 = mus

```

```

T1 = 1. + r
T2 = r - 1.0/r**2
T3 = T1 - 1.0/T1**2

dU = mu1*T2 + mu2*T3 # Equation 3.90

return dU

def Potential(x, *data):

    #Unpack tuple
    y, z, mu1, mu2, Vref = data

    r1 = math.sqrt((x+mu2)**2 + y**2 + z**2)
    r2 = math.sqrt((x-mu1)**2 + y**2 + z**2)

    T1 = 1.0/r1 + 0.5*r1**2
    T2 = 1.0/r2 + 0.5*r2**2

    # Equation 3.64 in Murray and Dermott
    U = mu1*T1 + mu2*T2 - 0.5*mu1*mu2 - Vref

    return U

def Potential2(y, *data):

    #Unpack tuple
    x, z, mu1, mu2, Vref = data

    r1 = math.sqrt((x+mu2)**2 + y**2 + z**2)
    r2 = math.sqrt((x-mu1)**2 + y**2 + z**2)

    T1 = 1.0/r1 + 0.5*r1**2
    T2 = 1.0/r2 + 0.5*r2**2

    # Equation 3.64 in Murray and Dermott
    U = mu1*T1 + mu2*T2 - 0.5*mu1*mu2 - Vref

    return U

#-----
# secondary / primary mass: M2/M1 (M1 > M2)
q = 0.2

# Number of points used in contour plot and in Roche Lobe
npts = 251

```

```

# Number of contours in contour plot
nc = 75

#Equations are from Solar System Dynamics by Murray and Dermott

# Equation 3.1
mu_bar = q / (1.0 + q)
# Equation 3.2, Location of Mass 1 (primary) is (-mu2, 0, 0)
mu1 = 1.0 - mu_bar
# Equation 3.2, Location of Mass 2 (secondary) is (mu1, 0, 0)
mu2 = mu_bar

#Define the Lagrangian Points
ratio = mu2 / mu1
alpha = (ratio/3.0)**(1.0/3.0) # Equation 3.75
L1_r2 = alpha - alpha**2/3.0 - alpha**3/9.0 - 23.0*alpha**4/81.0
L1_y = 0.0 # Equation 3.83
L2_r2 = alpha + alpha**2/3.0 - alpha**3/9.0 - 31.0*alpha**4/81.0
L2_y = 0.0 # Equation 3.88

# Equation 3.93
beta = -7.0*ratio/12.0 + 7.0*ratio**2/12.0 - 13223.0*ratio**3/20736.0

L3_r1 = 1.0 + beta
L3_y = 0.0 # Equation above 3.92
L4_x = 0.5 - mu2
L4_y = +sqrt(3.0)/2.0 # Equation 3.71
L5_x = 0.5 - mu2
L5_y = -sqrt(3.0)/2.0 # Equation 3.71
# L4 and L5 are done.

# Corrections to L1, L2, and L3...
L1_r2 = newton(dPotential1, L1_r2, args=(mu1,mu2))
L1_r1 = 1.0 - L1_r2 # Equation 3.72
L1_x = L1_r1 - mu2 # Equation 3.72

L2_r2 = newton(dPotential2, L2_r2, args=(mu1, mu2))
L2_r1 = 1.0 + L2_r2 # Equation 3.84
L2_x = L2_r1 - mu2 # Equation 3.84

L3_r1 = newton(dPotential3, L3_r1, args=(mu1, mu2))
L3_r2 = 1.0 + L3_r1 # Equation 3.89
L3_x = -L3_r1 - mu2 # Equation 3.89

# Now, calculate the Roche Lobe around both stars
# Potentials at L1, L2, and L3
L1_U = Potential(L1_x, 0.0, 0.0, mu1, mu2, 0.0)
L2_U = Potential(L2_x, 0.0, 0.0, mu1, mu2, 0.0)
L3_U = Potential(L3_x, 0.0, 0.0, mu1, mu2, 0.0)

```

```

# Find x limits of the Roche Lobe
L1_left = newton(Potential, L3_x, args=(0.0, 0.0, mu1, mu2, L1_U))
L1_right = newton(Potential, L2_x, args=(0.0, 0.0, mu1, mu2, L1_U))

xx = np.linspace(L1_left, L1_right, npts)
zz = np.linspace(0.0,0.0, npts)
yc = np.linspace(0.0,0.0, npts)

for n in range(1,npts-1):
    try:
        yguess = newton(Potential2, L4_y/10.0, \
            args=(xx[n], zz[n], mu1, mu2, L1_U), maxiter = 10000)
    except:
        yguess = 0.0

    if (yguess < 0.0): yguess = -yguess
    if (yguess > L4_y): yguess = 0.0

    yc[n] = yguess

yc[1] = 0.0
yc[npts-1] = 0.0

```

We can then read the X and Y pairs into a Blender mesh to draw the contours. First we create a single vertex mesh in the GUI called 'RocheOutline'.

```

import csv
import glob
import os
import bpy
import bmesh

obj = bpy.data.objects['RocheOutline']
bpy.ops.object.mode_set(mode = 'EDIT')
bm = bmesh.from_edit_mesh(obj.data)
dicts = [{'X': xx[i], 'Y': yc[i]} for i in range(0,len(xx))]

#Add in vertex elements with XYZ coordinates at each row
for row in dicts:
    xpos = row['X']
    ypos = row['Y']
    bm.verts.new((xpos,ypos,0.0))
    bmesh.update_edit_mesh(obj.data)
    #Note that the minus 2 is the number of points to connect
    bm.edges.new((bm.verts[i] for i in range(-2,0)))

bmesh.update_edit_mesh(obj.data)

bpy.ops.object.mode_set(mode='OBJECT')

```

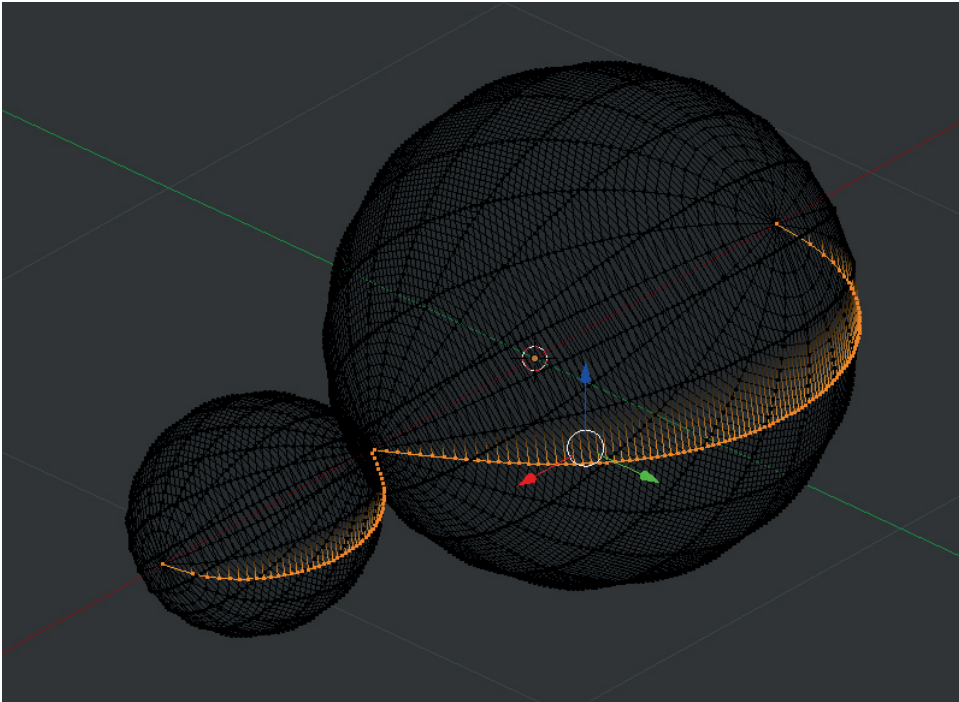


Figure 8.8. The zero-velocity curves can be mapped and then rotated in 3D using the spin tool in Mesh Edit mode.

Once this contour is created, we can use a mirror modifier to add in the second part of the symmetric contour and then spin the contour about the X -axis.

- Choose the Roche outline and add a mirror modifier about the X -axis.
- Enter Mesh Edit mode with the TAB key.
- Choose the spin tool on the left-hand Mesh Tools panel, set the degrees to 360 and number of steps to 40. Set the X , Y and Z -axes to 1.0, 0.0 and 0.0, respectively, making X the axis of rotation. Uncheck the ‘Dupli’ box so that a surface is created.

Finally the surfaces can be made transparent. This is useful in visualization scenarios where objects need to be seen within another mesh.

- Add a red surface material on the Properties panel.
- Change the diffuse intensity to 1.0.
- Set the shading emission and translucency to 0.0.
- Check the ‘Transparency’ box, click ‘Z Transparency’ (for faster rendering) and set the Alpha value to 0.35 (figure 8.8).
- In the Object tools section select ‘Smooth Shading’. This will allow the surface to be rendered without hard edges.

The primary and secondary masses can be added at their respective positions with simple colored UV-spheres, as can the Lagrangian points, designated as small tetrahedrons (figure 8.9).

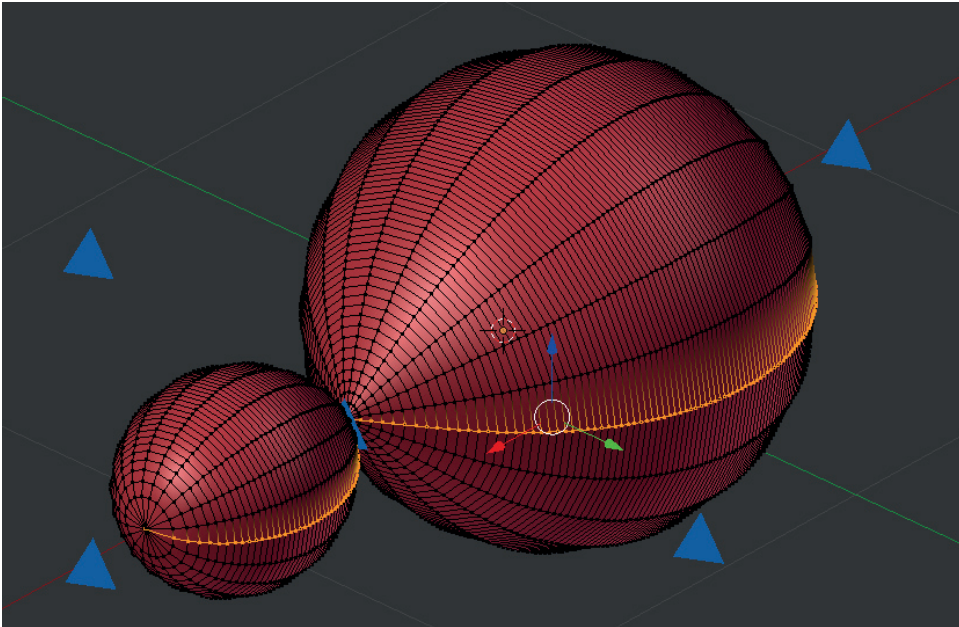


Figure 8.9. The Roche lobes can be rendered transparent via the Properties panel Materials tab.

8.5 Geophysics: planetary surface mapping

Blender is well suited for creating 3D surfaces from mapping data. In this project we will use topographic maps from the Mars MOLA Digital Terrain Topographic Maps to generate a 3D surface map [5]. This project introduces the following concepts:

- Importing an image as a plane.
- Using a displacement modifier.
- Mapping an image to a UV-plane.
- Using both a topographic map and color basemap imaging to layer textures.

The data are obtained from the USGS PDS Imaging Node³. The two images each have a size and resolution of 1000×1000 pixels at 1.32 kilometers per pixel. The topographic map will be used for the relative heights between points on the surface for the Blender mesh object. The Mars basecolor map will be used as a texture layer.

First we will activate a new feature under File → User Preferences → AddOns. Click the check box for ‘Import Images as Planes’ and then ‘Save User Settings’. This will allow us to to image a topographic map directly as a mesh plane object.

The images need to be grayscale topographic maps and *not* shaded relief maps.

- Choose File → Import → Images as planes and select the topographic map file.
- Press the TAB key to enter Edit mode.

³ Images and data available from www.mapaplanet.org.

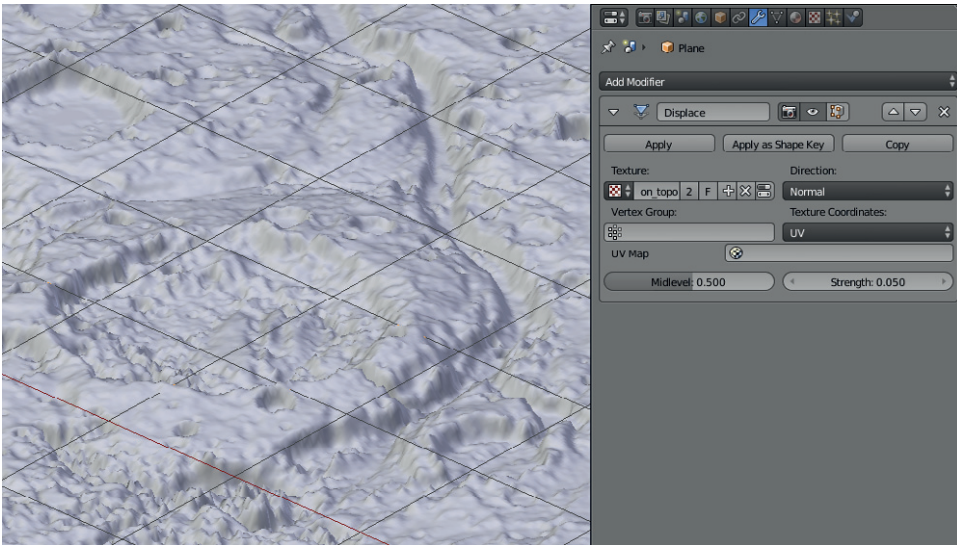


Figure 8.10. Setting up the displacement modifier to load the Martian topographic map into a mesh object.

- Press the W key and select ‘Subdivide’. If a mesh division is less than the number of pixels in the image, the resulting mesh will be averaged. This is acceptable for quick look visualizations, but can be performed at full resolution in the final rendering.
- In the Properties panel select Modifiers → Displace. Select the texture to be the topographic map file. Set the strength (a convenience scaling factor) to 0.05. Set the texture coordinates to ‘UV’ (figure 8.10).
- In the Properties panel select the Materials tab. Set the diffuse and spectral intensity to 0.0 and the shading emission to 1.0.
- In the Properties panel select the Textures tab. Add two textures—one for the displacement from the topographic map and one for the surface imaging. Their types need to be set to ‘Image or Movie’ and the coordinates to ‘UV’. For the surface imaging the diffuse influence should be set to ‘Color: 1.0’ and for the topographic map ‘Intensity: 1.0’. These settings are described in figure 8.11.

The final composite as viewed normal to the plane is shown in figure 8.12.

8.6 Volumetric rendering and data cubes

Data cubes are datasets composed of cross sections in a given phase space that are best viewed as rendered transparent volumes [6]. These kinds of data have applications in biology, medicine and astrophysics. CAT scan data provide slices of a scanned object with each element being a volumetric pixel (voxel). Data from radio telescopes that consist of right ascension and declination (positions on the celestial sphere) and frequency can be used to show the rotation of a galaxy or chemical species at millimeter wavelengths [7].



Figure 8.11. Terrain texture setting for Martian mapping.

In this application we utilize the following Blender functions:

- Loading cross section slices into Blender.
- Adding the data as a voxel ‘texture’.
- Setting the material and texture properties to render a volume.

For this example we will use CAT scan data of a human head that is easy to download and illustrates a simple example of volumetric rendering⁴. Other excellent examples from radio astronomy and the NRAO radio telescope facilities can be found on-line as well⁵.

- Begin with the default file that loads on Blender start-up.
- Right-click to select the default cube object. This will act as the data container.

⁴<http://graphics.stanford.edu/data/voldata/>

⁵<https://archive.nrao.edu/>

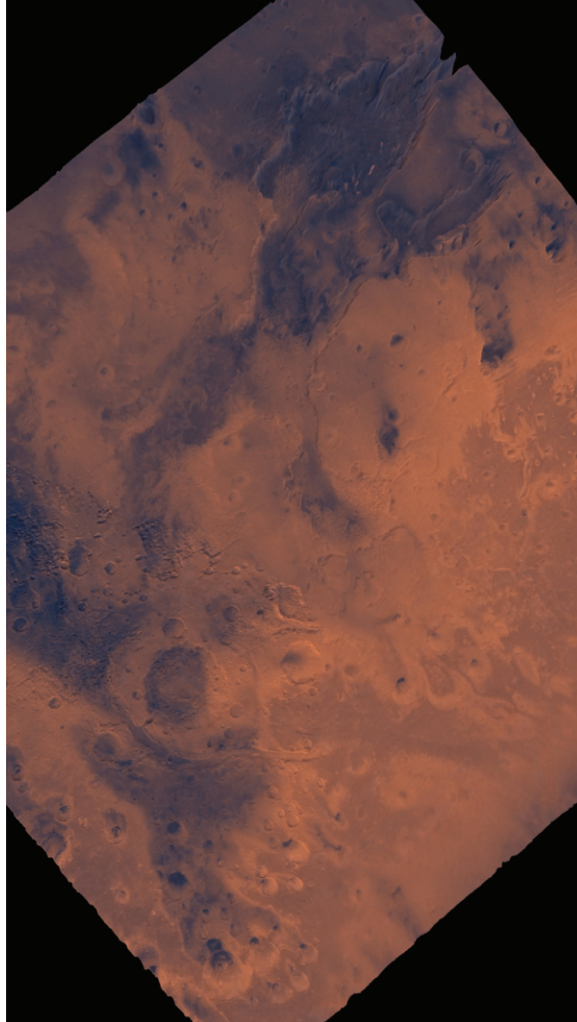


Figure 8.12. Mars terrain composite generated from a map displacement modifier and texture layer.

- Scale the data container to match the data cube dimensions in the Transform dialog on the right-hand side of the GUI.
- Click on the Materials tab on the far right-hand side of the Properties panel.
- Click the ‘+’ button to add a new material. Select ‘Volume’ as the material type.
- The graphic densities have been set in the accompanying blend file. For different data these will have to be scaled appropriately by the user.
- Change the graphic density to 0.0 and density scale to 2.0.
- Under ‘Shading’ set the emission to 0.0 and scattering to 1.4.

The material type has been set and now the image slices can be applied to the texture for the mesh.

- Click the Textures tab on the Properties panel.
- Click the ‘+ New’ button to create a new texture.
- From the Type drop-down box choose ‘Voxel Data’.
- Check the ‘Ramp’ box and set the far right ramp color to that of your choosing. We will use a greyscale color for this example.
- Under the ‘Voxel Data’ section, open the first file in the image sequence.
- Change the ‘File Format’ and ‘Source’ to ‘Image Sequence’.
- Under ‘Mapping’ change the projection to ‘Cube’.
- Under ‘Influence’ select ‘Density’, ‘Emission’ and ‘Emission Color’ and set their values to 1.0 (figure 8.13).

Several views of the data cube are shown in figure 8.14. The procedure outlined here can be applied to any kind of 3D dataset that requires volume rendering for viewing [8]. A data cube from radio telescope observations is shown in figure 8.15 [9].

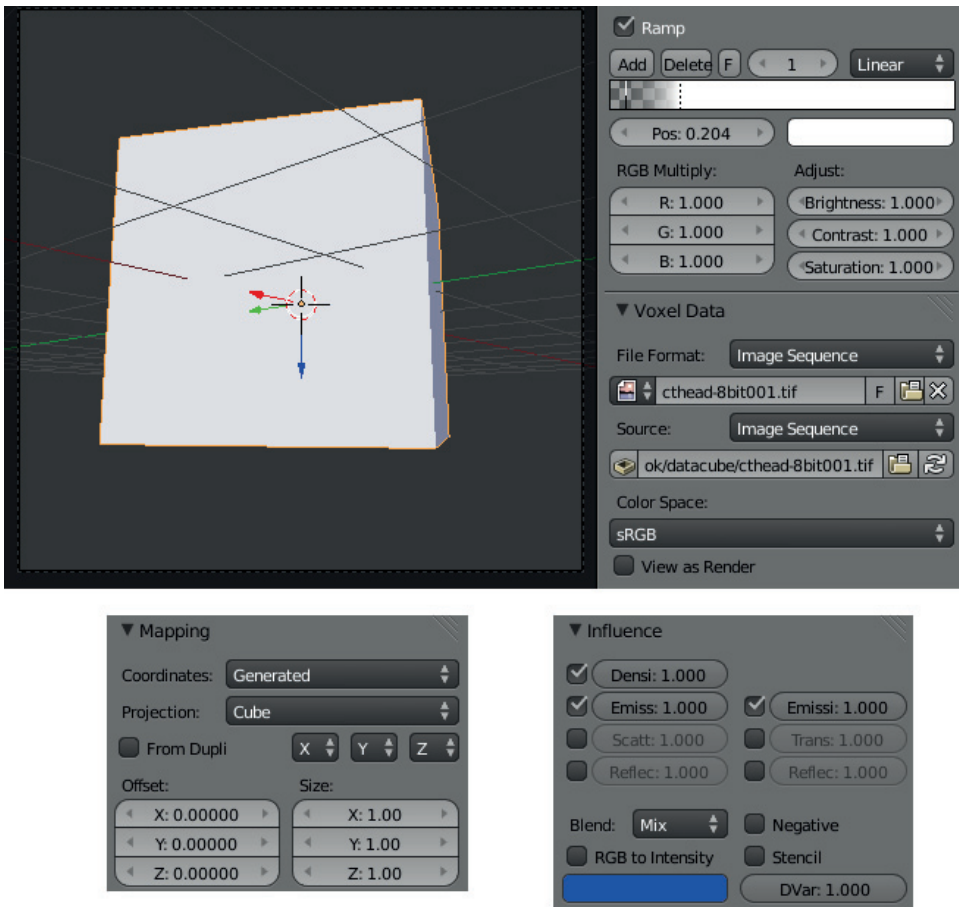


Figure 8.13. Set-up configuration for rendering a data cube.

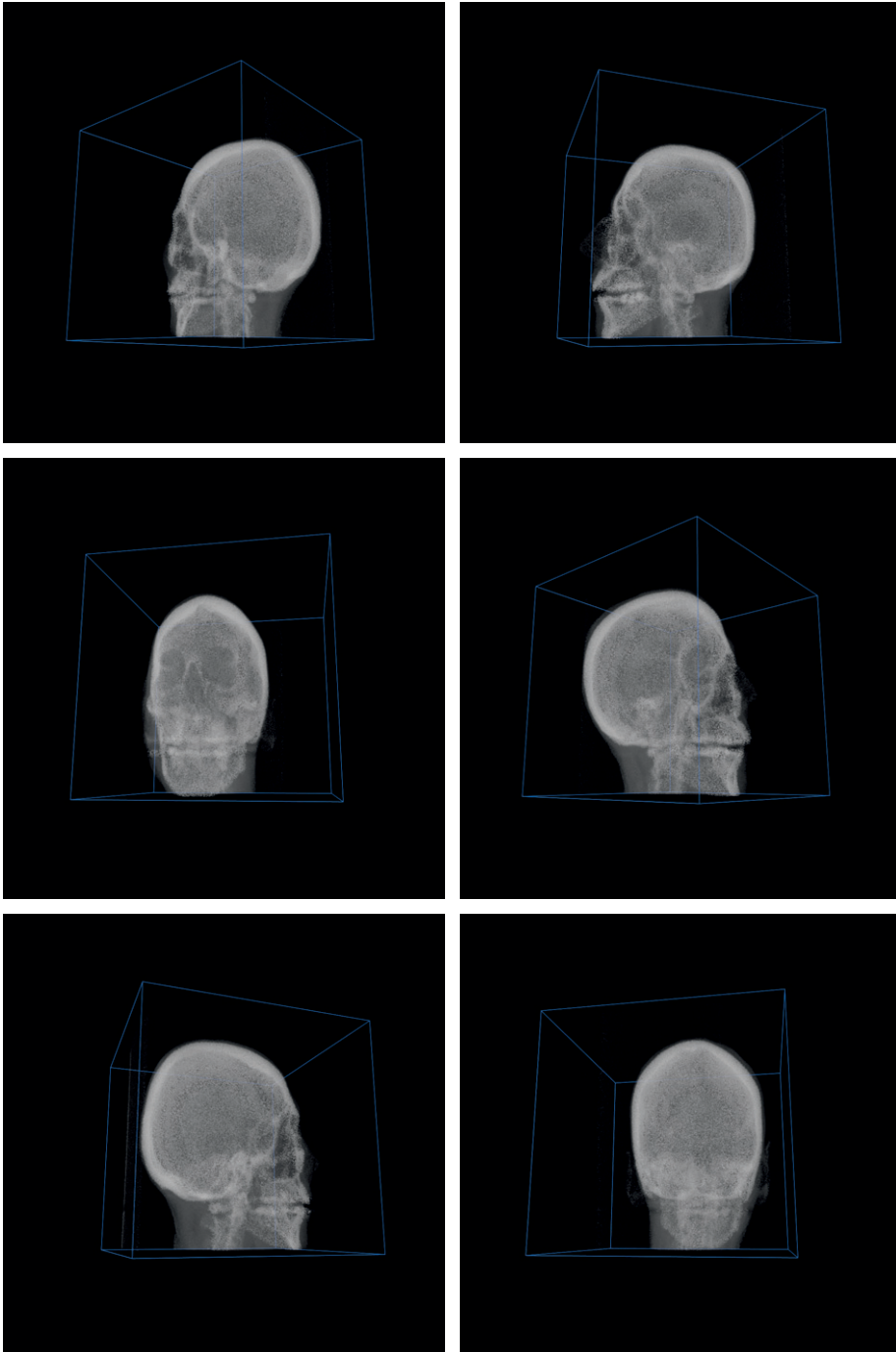


Figure 8.14. Rotating the volume rendering of a data cube.

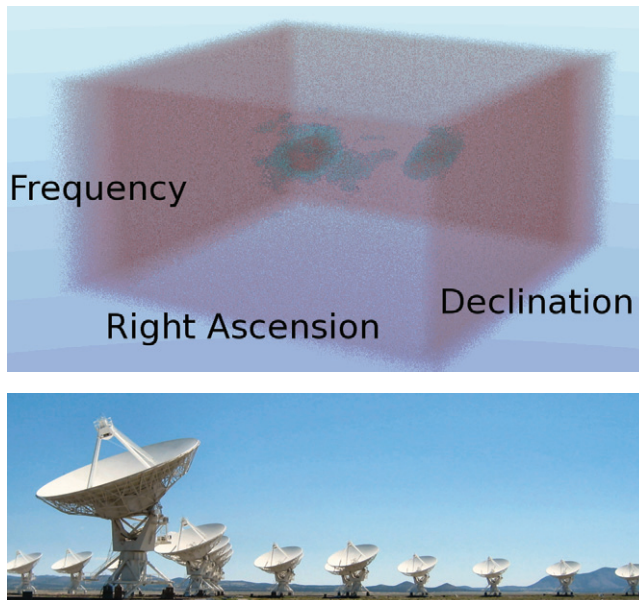


Figure 8.15. 3D rendering of a galaxy data cube obtained with observations from the National Radio Astronomy Observatory Very Large Array in New Mexico, USA.

8.7 Physics module and rigid body dynamics

The Blender physics module allows a user to use the built-in differential equation solver to add fields to a visualization scene and create physics demonstrations. Simple projectile motion, charged particles interacting with magnetic fields and simple harmonic motion are all possible with this module. The fields available to the user are shown in figure 8.16. In this project we will use the following concepts.

- Activate the Physics property tab.
- Set plane meshes as passive rigid bodies.
- Scale the simulation units to SI.
- Set time steps and compute the mesh positions (known as a *bake*) before running the animation to improve efficiency.

This example will combine a number of Blender physics module items and demonstrate their properties.

- Begin by Add → Mesh → Plane and scale the object with the S key.
- Choose the Physics module on the Properties panel (figure 8.17).
- Select ‘Rigid Body’ and set the type to ‘Passive’.
- A ball (Add → Mesh → UV Sphere) should be added at a location $Z = 1.0$ above the plane.
- Choose the Physics module, select ‘Rigid Body’, and set the type to ‘Active’.

Hitting the play button on the animation time line will compute the position of the ball with $a = 9.8 \text{ m s}^{-2}$ and no drag. The plane is passive and fixed, and therefore



Figure 8.16. Fields available to the user with the Blender physics module. Mesh objects can be set to react to the physical constraints that occur because of a given field.

the ball simply stops. The ‘Surface Response’ for both the ball and the surface can be modified from a range of zero to one. With a maximum value of 1.0 for both objects the collision will be elastic with no energy dissipation—the ball will simply keep bouncing throughout the animation repeatedly. We will now create projectile motion by giving the ball an X -component to its initial velocity.

- Set the ‘Bounciness’ level to 0.5 for the sphere and 1.0 for the fixed plane.
- Add a second plane and rotate it with keys R–Y–90. Set the Physics module to ‘Rigid Body’ and ‘Passive’ just like the first plane. In addition, check the ‘Animation’ box.
- Keyframe the X -location for the first and tenth frame. Due to the rigid body dynamics we have now set up, the plane will give the ball an X velocity.
- Set the ‘Friction’ value for the fixed ground plane to 0.0.

Pressing the play button one more time from the beginning will now show the ball exhibiting projectile motion.

We will use the mesh screw tool and multiple spheres to finish the rigid body dynamics example.

- Add → Mesh → Circle to create the start of a rigid body tube.
- Enter Mesh Edit mode with the TAB key and add a rotation vector⁶ as shown in figure 8.18.

⁶wiki.blender.org/index.php/Doc:2.6/Manual/Modeling/Meshes/Editing/Duplicating/Screw

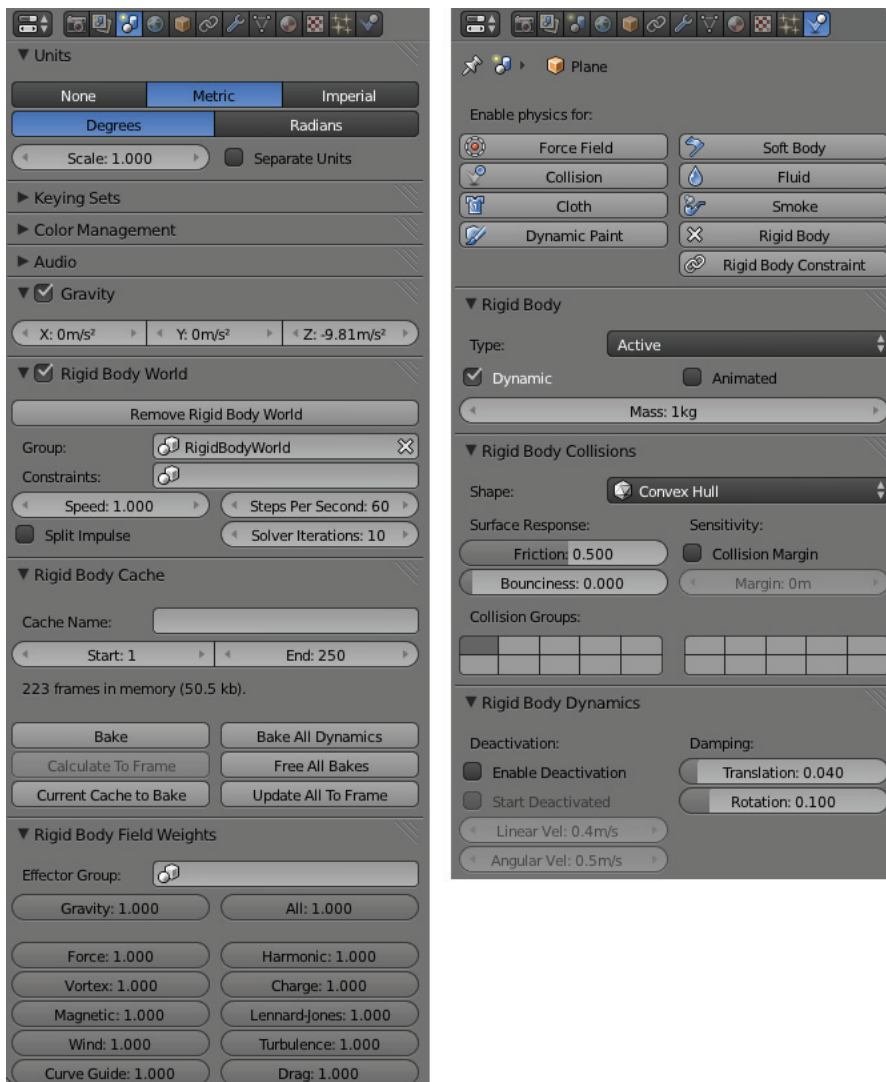


Figure 8.17. The Physics menu allows for the addition of fields to a visualization scene. These particular menus show the configuration for a fixed passive surface with rigidity and elasticity values. The acceleration due to gravity on Earth is set to the SI value of 9.8 m s^{-2} , and the differential equation solver is set to 60 steps per second under the Scene menu.

- Activate the screw tool on the left-hand side of the GUI and set the number of steps to 48.
- A corkscrew tube will be created. Press the TAB key again to enter Object mode and select the Physics Module tab. Set the 'Rigid Body' object to be 'Passive' with a 'Mesh' collision shape instead of the default 'Convex Hull'. This will allow the ball to roll *inside* the tube.

- Move the corkscrew relative to the plane mesh object such that the lower opening will allow the ball to roll out onto the plane.
- Change the sphere object's collision shape to 'Sphere' and place it at the top of the corkscrew tube.
- Duplicate the sphere with SHIFT-D and place the new objects on the flat plane (figure 8.19).

The simple elements for our rigid body dynamics simulation are in place. The last step will be to *bake* the simulation elements and compute all the positions. As depicted in figure 8.17, press the 'Free All Bakes' button followed by 'Bake'. Press the play button on the animation time line to view the simulation (figure 8.20).

For longer simulations, *baking* will take significantly longer depending on factors like the number of vertices and fields in the simulation. However, by baking in the dynamics and performing the computation *a priori*, the rendering time will be significantly decreased.

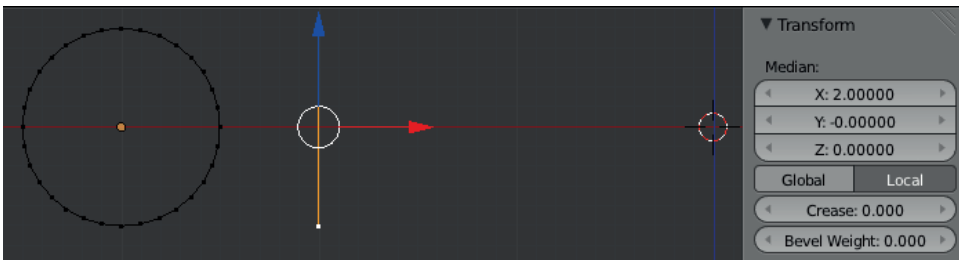


Figure 8.18. The initial set-up for a rigid body simulation. This example from the Mesh Edit window shows the set-up for a circle and rotation vector. The screw tool will be used to rotate the vertices about an incline to create a corkscrew tube shape. This shape will have a mesh collision shape in the Physics module such that another sphere can roll inside the tube.

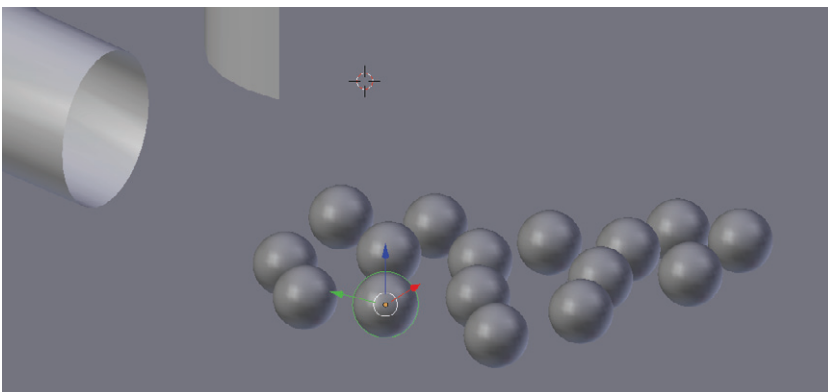


Figure 8.19. UV-sphere mesh objects with a spherical collision shape set in the Physics module are duplicated with SHIFT-D. The objects and properties are duplicated and set up for a collision.

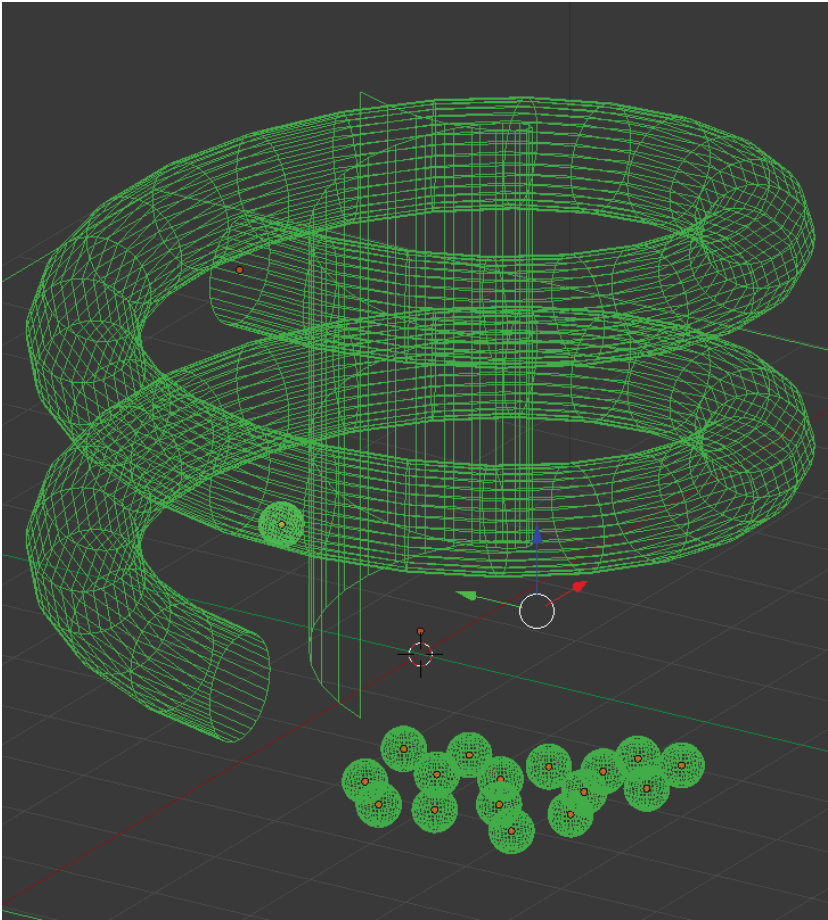


Figure 8.20. The physics menu allows for the addition of fields to a visualization scene. These particular menus show the configuration for a fixed passive surface with rigidity and elasticity values. The acceleration due to gravity on is set to the SI value of 9.8 m s^{-2} and the differential equation solver is set to 60 steps per second under the Scene menu.

Bibliography

- [1] Perryman M A C *et al* 1997 The HIPPARCOS catalogue *Astron. Astrophys.* **323** L49–52
- [2] Springel V 2005 The cosmological simulation code GADGET-2 *Mon. Not. R. Astron. Soc* **364** 1105–34
- [3] Jackson J D 1999 *Classical Electrodynamics* 3rd edn (New York: Wiley)
- [4] Murray C D and Dermott S F 1998 *Solar System Dynamics* (Cambridge: Cambridge University Press)
- [5] Christensen P R *et al* Mars global surveyor thermal emission spectrometer experiment: investigation description and surface science results *J. Geophys. Res.* **106** 23823–72

- [6] Levoy M 1988 Volume rendering: display of surfaces from volume data *Comput. Graphics Appl.*
- [7] Hassan A H, Fluke C J and Barnes D G 2011 Interactive visualization of the largest radio-astronomy cubes *New Astron.* **16** 100–9
- [8] Kent B R 2013 Visualizing astronomical data with Blender *Publ. Astron. Soc. Pac.* **125** 731–48
- [9] Walter F *et al* 2008 THINGS: The H I Nearby Galaxy Survey *Astron. J.* **136** 2563–647

Appendix A

Blender keyboard shortcuts

The listing below gives important keyboard shortcuts that will be useful in using Blender for scientific visualization.

A	Select/deselect all
TAB	Toggle between Mesh Edit and Object modes
X	Delete object
S	Scale object
R	Rotate object
G	Grab object
I	Insert keyframe
E	Extrude selected vertices, lines, or faces
.	Period key—center on selected object
CTRL-Z	Undo
SHIFT-S	Snap cursor
U	UV Mapping
F12	Still frame render
T	Object tools toggle
CTRL-E	Mark edges and seams on meshes
→	Advance animation frame
←	Reverse animation frame