Partiview User's Guide

Brian Abbott
Hayden Planetarium
American Museum of Natural History
New York, USA

http://virdir.ncsa.illinois.edu/partiview/

February 11, 2012

Hayden Planetarium
American Museum of Natural History
Central Park West at 79th Street
New York, NY 10024, USA

The *Partiview User's Guide* was typeset using LaTeX.

Contents

Contents 3

List of Tables 5

1 Introduction 6
- 1.1 What is Partiview? . . . 6
- 1.2 About this Guide . . . 7
- 1.3 Partiview License . . . 9
- 1.4 Help and Support . . . 10
- 1.5 Acknowledgments . . . 10

2 Installing and Testing Partiview 11
- 2.1 Program Requirements . . . 11
- 2.2 Installing Partiview . . . 12
- 2.3 Testing Partiview . . . 13

3 Using Partiview 16
- 3.1 Starting and Quitting Partiview . . . 16
- 3.2 Flight Controls . . . 17
- 3.3 Partiview's User Interface . . . 20
- 3.4 Data Formats . . . 26
- 3.5 Data Groups . . . 31
- 3.6 Importing Data . . . 34

4 Changing Global Properties 37
- 4.1 Background Color . . . 37
- 4.2 Changing Your Position . . . 38

4.3 Field of View . . . 40
4.4 Clipping Planes . . . 42
4.5 Point of Interest . . . 44
4.6 Resizing the Viewing Window . . . 45
4.7 Exporting the Display . . . 47

5 Changing Group Properties **49**
5.1 Drawing Points . . . 49
5.2 Placing Polygons on Points . . . 52
5.3 Drawing Textures on Polygons . . . 57
5.4 Setting the Luminosity of Particles . . . 58
5.5 Coloring Particles and Objects . . . 62
5.6 Label Properties . . . 68

6 Drawing Objects **70**
6.1 Boxes . . . 70
6.2 Spheres and Ellipsoids . . . 73
6.3 Meshes . . . 76

7 Creating Data Subsets **78**
7.1 Displaying a Random Subset . . . 78
7.2 Clip Boxes . . . 79
7.3 Thresholding Data . . . 80
7.4 Selecting Data with Specific Values . . . 82
7.5 Selection Expressions . . . 83

8 Statistical Reports **85**
8.1 The Spatial Extent of the Data . . . 85
8.2 Generating a Histogram of the Data . . . 86

A Keyboard Controls **89**

B Partiview Commands **90**

Index **125**

List of Tables

2.1 Test Data Set Files 13

3.1 Log and Linear Flight Scales 18
3.2 Flight Control with the Mouse 18
3.3 Motion Control in Partiview 19
3.4 Additional Mouse Functionality in Partiview 19
3.5 Slider Ranges 23

5.1 Polygon sizes and luminosity 54
5.2 Orientation vectors for polygons 57

A.1 Shortcut keys in Partiview. 89

B.1 Issuing Partiview Commands 90

1

Introduction to Partiview

This chapter will introduce you to Partiview. We will briefly describe the software and discuss the structure of this manual, the Partiview license, as well as support for the software. Subsequent chapters will discuss installation and provide tutorials on how to use Partiview.

1.1 What is Partiview?

Partiview is a program that visualizes and animates three dimensional particle data. With Partiview, you have the ability to navigate through 3-D data as if you're "flying." Partiview is not limited to particle data; it can display 2-D images, multiple 2-D polygons which can be built into 3-D surfaces, and time-evolving data.

Partiview was created by the Virtual Director Group at the National Center for Supercomputing Applications (NCSA) at the University of Illinois, Urbana-Champaign. It is the child of a larger program called Virtual Director which was developed to create and edit flight paths through data sets in the CAVE virtual reality environment. These recorded flight paths are useful when making digital movies for educational programs like NOVA, IMAX films, and even the Space Shows at the Hayden Planetarium.

Partiview possesses many of the same features as Virtual Director, allowing one to view and explore data sets on your desktop or laptop. It cannot record flight paths, though it can play them. You can learn more about Partiview and how to visualize your own data in this manual.

1.2 About this Guide

This guide is meant to provide a hands-on tutorial of Partiview for users of all levels. We have tried to balance the ease of explanation with a level of technical detail. The chapters are organized in the following way.

- "Introduction to Partiview" (the chapter you're currently reading) will introduce you to Partiview, describe its origins, distribution license, and support policies.
- "Installing and Testing Partiview" describes how to install and test Partiview on your system using a sample data sets.
- "Using Partiview" provides the basics of how to navigate, using the graphical user interface, data groups, and working with your own data.
- "Changing Global Properties" describes how to set certain global properties (i.e., those properties that affect all data groups), such as your position, the field of view, and how to resize and detach the Viewing Window.
- Chapters 5, 6, 7, and 8 discuss what you can do to a specific data group, including how to visualize your data with points and polygons, how to set the size of your data, drawing boxes and spheres to illustrate your data, and creating data subsets and statistical reports. These chapters are designed to be read while you're working with the sample data sets provided.
- The Reference Appendix lists the Partiview "Keyboard Controls" and useful "Partiview Commands."

Sample Data Sets We have created the Partiview User's Guide Data which includes several data sets that have been designed to accompany this guide. These include:

Test Data The least complicated data set, only the bare essentials are included here for testing purposes.

Complex Data A complex version of the Test Data where we have added data variables and many additional settings and attributes to the test data.

Sample Data A sampled data set of about 10,000 particles in a spherical distribution. This data set is mainly useful when talking about statistical operations and other commands that act on large numbers of particles.

These data sets are used extensively in this manual for understanding how to use Partiview and its rich command set.

Typographical Conventions

- Blue text: an external link to the Internet.
- Green text: An internal link to another section of this document.
- `Red typewriter font`: Partiview input—commands that you type into Partiview's Command Line or as a keyboard shortcut.
- `Black typewriter font`: a generic face used for console output from Partiview, file names, text in a file, keyboard keys, and text appearing on the user interface.
- **Boldface**: Partiview commands.
- *Italic*: Partiview command arguments which are substituted for a value.
- [ ] (Square brackets): denote optional parts of Partiview commands or optional arguments (the brackets are never typed themselves). For example, the command **cen[ter]** [*x y z*] [*radius*] may be issued by typing `center` or just `cen`, and takes two optional arguments (an *x*, *y*, *z* point, or a length).
- [**+, *,** /]: signifies that the command may be scaled by adding, multiplying, or dividing the current value by a constant. For example, you can change the label size with the **labelsize** command by either setting the size explicitly (`lsize 150`) or by scaling the current value by some factor (`lsize *10`).
- | symbol: represents the word "or," meaning one option *or* another may be used, but not both. For example, the command to draw a clip box in Partiview has the form: **clipbox** [**on** | **off** | **hide**] | [*boxparameters*]. You can either use `cb on`, `cb off`, `cb hide`, or `cb 0,0,0 10,10,10`. You cannot use two of these arguments, like this: `cb hide 0,0,0 10,10,10` (this is wrong).
- ⟨ ⟩ (angle brackets): used to group items together, such as the style arguments in the **ellipsoid** Data Command. The argument has the form **-s** ⟨**solid** | **plane** | **wire** | **point**⟩, which groups the styles available to use with the **-s** option. Only one of the styles may be used with the **-s** option.

1.3 Partiview License

Developed by:

> Stuart Levy, NCSA Virtual Director Group
> University of Illinois Urbana-Champaign
> http://niri.ncsa.uiuc.edu/partiview/

1.4 Help and Support

The main place for support is the official Partiview website, where you will find information on applications of Partiview, people working with Partiview, and the on-line discussion group for Partiview users.

The Partiview Google group, launched in 2004, houses an email archive and allows you to ask questions to your fellow Partiview users.

If you have questions regarding Hayden Planetarium's Digital Universe, please contact us at the Digital Universe website.

1.5 Acknowledgments

Partiview was created by Stuart Levy. We would like to thank Stuart for his help in creating this document and aiding in the distribution of Partiview at the American Museum of Natural History and the Hayden Planetarium.

Parts of this document are based on the Partiview manual written by Peter Teuben and Stuart Levy. We are grateful for allowing us to borrow liberally from that document.

We also wish to acknowledge support from NASA and the NCSA Alliance for the distribution and development of Partiview and the *Partiview User's Guide*.

2

Installing and Testing Partiview

This chapter describes how to install and test Partiview on your system. Because Partiview demands resources from your GPU, we recommend you test Partiview's performance on your computer using our sample data sets. After you've installed the software, see "Testing Partiview on Your System" to test Partiview using the accompanying data sets.

2.1 Program Requirements

Partiview, itself, is not terribly demanding on computational resources. The size and display characteristics of the data sets are what typically determine the resources required. If in doubt, load the Test Data first to be sure your computer can run Partiview.

Partiview requires the following to run on your computer:

- **Windows**, **Macintosh**, and **Linux**.
- **A 3-D accelerated graphics card** Almost any off-the-shelf computer can run Partiview with simple data sets, but for larger, more complex data sets, you might need a separate graphics card beyond the on-board graphics support found in many low-end laptops.
- Enough resources to load the data into memory.

2.2 Installing Partiview

You can install Partiview using one of two methods:

Downloading the application: If you're just using Partiview to view data, this is probably the best option for you. Downloading a pre-compiled executable application should be sufficient for most users. Visit:

Digital Universe Website Available in Windows, Macintosh, and Linux along with the Digital Universe Atlas.

Partiview Website Available in Windows, Macintosh, and Linux as a stand-alone application without data.

Downloading source code: Use this option if you wish to build Partiview from the source code. Only expert users will need to do this. The Partiview source code exists on a concurrent versions system (CVS) server at the University of Maryland hosted by Peter Teuben. Instructions for downloading and installing the code are on the Partiview web site.

Partiview is completely self-contained; it will not add files or change settings on your computer outside of the directory in which you install the program.

2.3 Testing Partiview on Your System

The test data set was designed for you to see how your computer performs while running Partiview. The data set consists of 27 points that form a cube around the point $(x, y, z) = (0, 0, 0)$.

After downloading and installing the Partiview User's Guide Data, the compressed file will extract the files shown in the table below. These files make up a basic data set.

Table 2.1 – Files in the Test Data Set, part of the Partiview User's Guide Data set. All files other than the start scripts can be found in the `./data/mandata` folder.

File Name	Description
`halo.sgi`	Texture (image) file.
`test.speck`	Formatted data file.
`test.label`	Labels for the data points.
`test.cf`	Pre-loaded configuration commands.
`testdata.bat`	Windows start script.
`testdata.sh`	Linux start script
`testdata.command`	Macintosh start script

The `.speck` file contains the data and is in the format

```
x y z texturenumber
```

while the `.label` file contains the labeling information in the format

```
x y z text label number 1
```

where the word **text** is a Partiview command and `label number 1` is the label you would like to see at the point x, y, z. The `.cf` file is the config file which executes a series of Partiview commands, such as loading data sets, setting colors, point and polygon sizes, the field of view, etc. Finally, the `.sgi` files are images, or textures, that are placed on points or polygons.

The `testdata.bat`, `testdata.sh`, and `testdata.command` files are used to start Partiview and load the appropriate config file (`test.cf`). They take the form (without the square brackets):

```
[path to partiview executable] [path to .cf file]
```

For example, if we have the start script in the same directory as the executable, then for Windows users, the `testdata.bat` file looks like:

```
partiview ./data/mandata/test.cf
```

while the Linux- or Macintosh-based script looks like:

```
./partiview ./data/mandata/test.cf
```

a difference of one "`./`".

Starting Partiview with the Test Data Let's start Partiview with the test data by running the `testdata` start script for your operating system. If you are getting errors and Partiview is not starting, the most likely culprit is that the data files aren't in the correct place for Partiview to find them. The start script must be in the same folder at the Partiview application, and the data must me in the folder `data/mandata` also in the Partiview folder.

Test 1: Flying Around Once you have these data displayed in Partiview, begin by moving these data around a bit. Make sure you're in Orbit Flight Mode. You should see an `[o]rbit` just below the Flight Mode Menu. If the active flight mode is not `[o]rbit`, then change it to the Orbit Flight Mode by selecting it from the menu or by typing an o (lowercase letter O) in the Viewing Window (where the data are displayed). Now, with the left mouse button pressed, drag the mouse to move the data cube.

As you fly, you are orbiting around the central point, which is called the Point of Interest. In this data set, the Point of Interest is located at the center of the data cube and is always indicated by a 3-D Cartesian axis. In Partiview, the red axis points in the direction of positive x, green is in the positive y direction, and blue is in the positive z direction.

How's your flying? Do these data move smoothly across the screen? Do these data move at all? If it takes several seconds for the data cube to move, then your graphics support is not sufficient to run Partiview effectively. If these data move freely without too much computational delay, then proceed to the next test.

Test 2: Turning on the Polygons Our next test involves turning on the polygons for each of these points, thereby increasing the amount of information your computer must redraw. Let's turn on the

polygons by either clicking the Polygon Toggle Button or typing `poly on` in the Command Line. Now, you are displaying a polygon on each point, upon which a texture (or image) is drawn. Do you notice any difference in performance with the polygons on? If so, you may have trouble handling larger data sets. If your performance is still good, move on to the final test.

Test 3: Increasing the slum Value The **slum** command adjusts the luminosity scale factor, thereby increasing the brightness of your data. You can increase this scaling factor by using the Slider. While these data are in motion, click on the slider value selector and *slowly* drag the slider to the right, increasing the value. The polygons should be getting larger and will continue to grow, eventually filling the entire display. The motion might slow down or stop with an increased **slum** value.

Depending on your performance, you may raise the **slum** value to such a level that your computer freezes up. If this occurs, simply click the mouse on the left side of the slider (lower values) and wait for the computer to catch up. It could take a minute or two. If, however, you are able to use the slider pretty easily and your computer doesn't freeze up while redrawing the graphics, then, with the **slum** value down to a reasonable level (i.e., the data cube is not completely blown out), try to increase the window size as you would any other window. If a larger window still performs well, maximize the window to see if you can run Partiview full screen.

If you are still running without significant slowdown, then you are in good shape to display larger, more complex data sets. However, if you experienced some sluggish motion and some delay when changing the Slum Slider, then you may have to experiment to see what happens when larger data sets are loaded. An upgrade to an accelerated graphics system might be necessary to display large data sets. However, an adjustment to the config files for data display is a possible solution as well (for example, displaying the data with a decreased polygon size or even without polygons).

If Partiview runs well on your system, you are now ready to begin learning about how to use the software. In the following chapters, we will explore the software in depth so that you become familiar with using Partiview and visualizing your own data.

3

Using Partiview

This chapter describes the basics on how to use Partiview. We cover how to navigate (Partiview's flight controls), how to use Partiview's user interface, as well as data formats, data groups, and importing data.

3.1 Starting and Quitting Partiview

Start Scripts Partiview can, of course, be run without loading data by either double-clicking on the executable or running it in a shell window. However, we prefer to initialize Partiview with a config file which then loads data and sets their display characteristics. This is accomplished using a start script that takes the form (without the square brackets):

```
[path to partiview] [path to config file]
```

For example, if we were using Windows, the start script for the Test Data would look like

```
partiview ./data/mandata/test.cf
```

This infers that the start script is located in the same directory as the Partiview executable. If this is so, then Partiview should start, then read the `test.cf` file which is located two folders down from the executable (i.e., inside the folder with the executable is a folder called `data`, inside of which is a folder called `mandata` where the `test.cf` file resides).

The .partiviewrc File Once Partiview is executed, it first looks for a file called `.partiviewrc`. This optional file must be in the same directory as the Partiview application and contains configure options for Partiview. Typically, this file contains information about the overall characteristics of Partiview that you would want to apply to all sessions of Partiview, whether you run the Test Data or the Digital Universe. A typical file may have these commands:

```
winsize 1280 1024
eval detach
inertia on
```

telling Partiview to set the size of the Viewing Window to 1280 $\times$ 1024, detaching the GUI from the Viewing Window (where the data appear), and setting the inertia feature on, which allows drift in flight (the inertia command is not really necessary since it is on by default, but it is included here as an example). If you want the same field of view for all sessions, then you could include the **fov** command here too.

Once the `.partiviewrc` file has been read, then the data config file (i.e., `test.cf`) is processed. These files typically read in data files and configure the display. By looking at the config files supplied, you will quickly learn what commands are appropriate and how you can use these files to tailor the display to your liking.

Quitting Partiview There are several ways to quit Partiview. These are:

- Clicking on the top-level window button that your operating system supplies. For Windows this is the "$\times$" button on the upper right, for Macs this is the red button on the upper left side of the window border.
- Hitting the [ESC] key in the Viewing Window. (If you use the escape key in the GUI first, you may have to use it again in the Viewing Window if the two are detached.)
- Typing **exit** in Partiview's Command Line.

3.2 Flight Controls

Partiview has four flight modes, which each mode capable of two types of motion (forward/backward, orbit, etc.), one for each mouse button.

Flight Scale Moving forward or backward presents difficulties when you are traversing the large scales of some data sets. If your speed were constant everywhere, it would take forever to get from one end of the data to another. For this reason, Partiview has two flight scales in the forward and backward direction, *linear* and *logarithmic*, described in Table 3.1.

Table 3.1 – Flight scales in Partiview. The distance, d, is measured from the Point of Interest.

Scale	If $d = 0$...	If $d = \infty$...	Description
Linear	v = constant	v = constant	Constant scale everywhere; useful for exploring an object near the Point of Interest.
Logarithmic	$v = 0$	$v = \infty$	The scale increases radially away from the Point of Interest, resulting in higher speeds as your distance increases from the Point of Interest.

Flight Modes Partiview has four flight modes, listed in Table 3.2. Each mode is capable of two types of motion. These are discussed in Table 3.3.

Table 3.2 – Flight Control with the Mouse in Partiview. Shortcut keys are shown in square brackets and are used in the Viewing Window.

	Button			
Flight Mode	Left (1)	Middle (2)	Right (3)	Scale
Fly [f]	pan	select [p]	forward	linear
Orbit [o]	orbit	select [p]	forward	log
Rotate [r]	orbit	select [p]	rotate	not applicable
Translate [t]	translate	select [p]	forward	linear

Changing the Flight Mode Change the flight mode by selecting a mode from the Flight Mode Menu or by typing one of the shortcut keys (f, o, r, or t) in the Viewing Window. The active flight mode appears below the Flight Mode Menu.

Table 3.3 – Motion Control in Partiview

Motion	Description
pan	Shift your view without moving from your current position, equivalent to looking around by moving your head but not your feet.
forward, reverse	With eyes forward (looking at the center of the display), moving toward or away along your line of sight.
orbit	Orbit about the Point of Interest. If the Point of Interest is in view, this is a simple orbit around that point. If the Point of Interest is out of view, then you will orbit the Point of Interest but look forward, akin to looking away from the center of a carousel as you revolve about its center.
rotate	Rotate the view about the Point of Interest. When the Point of Interest is in view, this produces a twisting motion parallel to your screen. If the Point of Interest is out of view, then the data will appear to approach from an angle.
translate	Move in a direction parallel to the display, thereby moving the data across the display in the direction of mouse motion. This is equivalent to moving your feet sideways while keeping your eyes looking straight ahead.

Table 3.4 – Additional Mouse Functionality in Partiview. Use the mouse buttons or designated keyboard shortcuts in the Viewing Window.

Function	Mouse Button	Keyboard	Description
select	middle	p	With the cursor over an object, click the middle mouse button to produce a report on that object in the Console Window. If you don't have a middle mouse button, place the cursor on an object and use the p key.
Changing the Point of Interest	[Shift]+middle	[Shift]+p	Select a data point near the mouse and move the Point of Interest to that location.

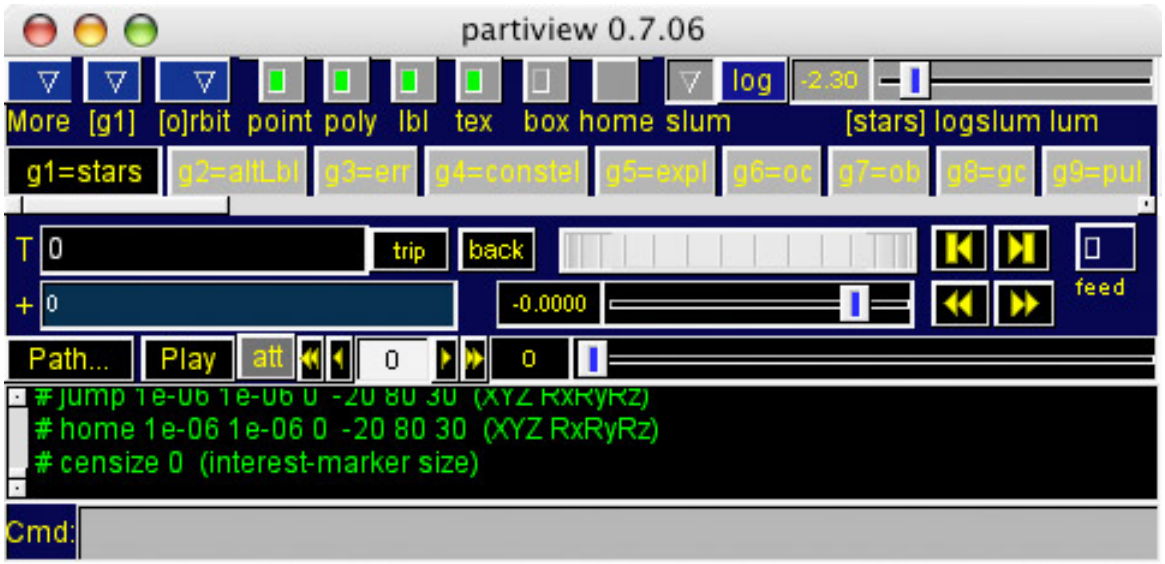
3.3 Partiview's User Interface

Partiview's graphical user interface (GUI) was written to be simple and compact. However, for a new user, parts of the interface may seem cryptic. The buttons, sliders, and menus are designed to make navigation, data manipulation, and data group toggling effortless, but they result in an interface that looks foreign to most experienced computer users.

Most of Partiview's rich command set is not represented in the GUI. We recommend examining the Reference Sections for a list of commands and keyboard shortcuts.

Using the screen shot below, let's investigate the GUI elements one by one.

Figure 3.1 – Partiview's graphical user interface (GUI). Consult the tables and images below to learn how to use the GUI buttons and sliders.



Partiview Menus 		
	**More Menu**	`More` contains two items that are rarely used. `Inertia` toggles the "drift" feature on and off when you are flying. `H-R Diagram` invokes a separate window where an H-R Diagram will be displayed (this is not implemented in the Digital Universe).
	**Groups Menu**	Choose the active data group. The active data group is displayed below the menu (here it is group 1, indicated by `[g1]`).
	**Flight Mode Menu**	Choose between the Orbit, Fly, Rotate, Translate flight modes. Orbit is the default flight mode and is indicated below the menu by `[o]rbit`. (See "Flight Controls" for information on Partiview's four flight modes.)

Toggle Buttons 		
	**Point**	Turn the points on and off for the active group.
	**Polygon**	Turn the polygons on and off for the active group.
	**Label**	Turn the labels on and off for the active group.
	**Texture**	Turn the textures (images) on and off for the active group.
	**Box**	Turn the boxes on and off for the active group.
	**Home Button**	Return to the 'home' position set by the **home** command in the config file.

The Partiview Slider 		
	**Slider Scale Button**	A toggle button between the logarithmic or linear scale (if available) for the active slider. See the table below for the range on each slider in the log and linear modes.
	**Slider**	Use the blue value adjuster to alter the value of the active slider. The logarithmic or linear value is indicated to the left of the Slider. Below the slider you'll see the active data group and the linear value of the slider (except for the Slum Slider, which has no value shown).
	**Slider Menu**	A drop-down menu to select the slider function. The active slider appears below the menu (the Slum Slider is shown). Slider Menu functions are described in the table below.

Slider Menu 		
Alpha	Sets the opaqueness of an object or image.	**alpha**
FOV	Adjusts the field of view. We typically use values near 60°; "telescopic" views may be achieved with small values for the field of view.	**fov**
Censize	The size of the Cartesian Point of Interest marker. Values are in the units of the particular data you are viewing.	**censize**
Labelmin	Set the minimum pixel height before a label will be drawn. This is useful if you want only the nearby labels displayed. Set this value to 0 pixels for all labels to be drawn. Setting **labelmin** to 20 will draw labels only when they are more than 20 pixels high.	**labelminpixels**
Labelsize	Set the height of the labels in pixels.	**labelsize**
Polysides	Adjust the number of sides of the polygons in the active group.	**polysides**
Polysize	Set the size of the polygons.	**polysize**
Slum	Scale the luminosity of the particles, increasing or decreasing their brightness.	**slum**

Table 3.5 – Range on the sliders in the linear and logarithmic scales (if available).

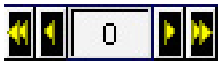
Slider	Linear Range	Logarithmic Range	Default Range
Alpha	0–1	not available	linear
FOV	0–180	not available	linear
Censize	0–10,000	0.001–10,000	log
Labelmin	0–20	not available	linear
Labelsize	0.01–1,000	0.001–1,000	log
Polysides	3–16	not available	linear
Polysize	0–10	0.001–10	log
Slum	not available	0.001–31,623	log

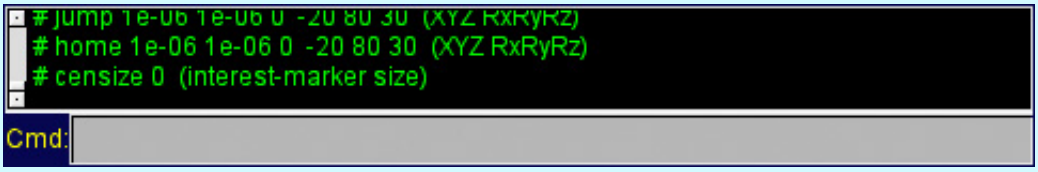
Group Buttons **Note: Group Buttons appear only if two or more data groups are defined.** 	
Left Mouse Button	Turn the active group on or off.
Right Mouse Button	Activate a data group (a data group must be active for you to change its display properties).

Time Controls

NOTE: The time controls appear in the GUI only when time-evolving data are loaded.

	**Time Display**	Black text box displays the current time. If an offset has been set using the Trip Button, this shows the offset from the tripmeter. The absolute time is the sum of the T and + text boxes.
	**Reference Time Display**	If the Trip Button is pressed, this blue text box shows the reference time.
	**Trip Button**	Marks a reference in time. Sets the Time to zero and the Reference Time to the current time.
	**Back Button**	Sets Time to zero. If the Trip Button has been set, this will return the time to the Reference Time.
	**Time Dial**	Fine-control time adjuster.
	**Time Control Buttons**	Adjust time by (0.1 $\times$ *speed*) data time units.
	**Speed Slider**	Logarithmic control of the speed.
	**Speed Toggle Buttons**	Toggle time forward or backward.
	**Feed Button**	This button has no effect and was built into the GUI for future use.

Flight Path Controls **NOTE: Partiview is not capable of recording a flight path.** 		
	**Path Button**	Opens a file explorer to choose a flight path file to load.
	**Play [Stop] Button**	Toggle the path animation on and off. Right-click on this button to adjust the play speed. `Play 0.5` will play the path at half speed; `play 5` will play the path 5 times as fast. `Play 5f` increases the frame rate 5 times.
	**att Button**	This button appears to have no function.
	**Frame Controls**	The frame number is displayed in the white text box.  and  move your position on the path by ± 1 frame.  and  move your position on the path by ± 10 frames.
	**Flight Path Slider**	Manually adjust time and position on the path. Wall-clock time is shown in the black text box to the left of the slider.

Console Window and Command Line 	
Console Window	Shows the input and output to and from Partiview. Some commands issued by the user are echoed here in yellow along with Partiview's response (if any) to them in green.
Command Line (`Cmd:`)	Use the Command Line to enter Partiview commands interactively. To type in this line, focus must be given to this narrow window. You can do this by either placing the mouse in this small space (and hope it stays there) or use the `[Tab]` key to move the cursor to the Command Line. Use the up and down arrow keys to scroll through the history of commands issued.

3.4 Data Formats

In this section we will discuss how Partiview data is formatted and take you through the steps necessary to display data in Partiview. Note that we give a step-by-step recipe of this information in "Importing Data."

Test Data Files Now that we have seen the Test Data set in Partiview, let's look at the actual files that make up the data set to understand how the files are formatted. A listing of files can be found in Table 2.1. Data files in Partiview can have any extension; however, we recommend using `.speck` for data files and `.cf` for config files. There can be a separate label file as well with the extension `.label`, but these data can also appear in the `.speck` file, streamlining the data and labels into one file.

test.speck If you open this file in a simple text editor, you will find some commands followed by the data listing. The `test.speck` file should be in the `mandata` folder, which is in the `data` folder located where you are running Partiview.

Looking at the file, you will see a number of lines throughout the file that begin with the '#' symbol. This indicates a comment in the file, a line that will be ignored by Partiview and is useful for inserting comments and notes in the file.

Data in Partiview take the form:

```
x y z datavar0 datavar1 ...  datavarN
```

where x, y, and z are the 3-D coordinates of the point and the data variables (set with the **datavar** command) are pre-defined data variables that are used to describe each point, such as luminosity, color, etc. These data variables are not necessary though; Partiview will take files with just x, y, z data.

In `test.speck` we define one data variable called `txnum`, which is the texture number assigned to that point. While they are all the same for each point, we need to specify a texture number to load an image (or texture) on a polygon. It should be noted, however, that the first three columns of any data file are reserved for x, y, and z. Therefore, column 4 will be data variable 0, if it exists.

The first line of this file (after the commented lines) reads

```
datavar 0 txnum
```

By issuing this command in the data file, we are setting a data variable called `txnum` to data variable 0 which is column 4 in the data file. The **datavar** command is used to define any data that we want associated with each x, y, z point. For example, in a stellar data set one may want to include information on each star's brightness, its color, and its luminosity. This is possible by writing those attributes to your data file and defining them using the **datavar** command.

While we have defined column 0 to be this variable called `txnum`, we need to explicitly tell Partiview which column contains the texturing information. This is done in the next line with the **texturevar** command, which sets the value to `txnum`, the data variable assigned to represent the texture number for each particle. Finally, we need to associate the value of the `txnum` to an image. Using the **texture** Data Command, we assign a texture number of 1 (corresponding to the values in column 4) to a file called `halo.sgi`.

test.label The label file is just what you might imagine it to be, a file containing labeling information. As we stated earlier in this section, labeling information can be placed in a `.speck` file. However, for large data sets, it makes sense to have a separate `.label` file to keep your data better organized. In Partiview, the **text** command is used to place text at a specified point. A label takes the form

```
x y z text yourlabel
```

where x, y, and z are the location of the label and `yourlabel` is the label for that point. The label can have spaces, commas, and other non-alphanumeric characters.

test.cf The `test.cf` file is the config file for the Test Data. Config files are composed of Partiview commands that are read and executed at start-up. These commands load the data, configure the data display, and set various Partiview preferences.

If you inspect the `test.cf` file, you will notice the first two lines (that are not comments) are the commands that read in the data using the **include** data command. Note that many of these commands begin with the word `eval`. The **eval** command is the necessary prefix for a Control Command if you wish to issue it within a data file, rather than interactively at the Command Line. We discuss the

difference between Control Commands and Data Commands in the next few paragraphs, and we will discuss the commands you see in this file throughout this chapter.

Adding Data in Partiview While we touched on the topic of loading data, we will offer a more through discussion here. Let's start from scratch with an empty canvas, if you will. Start Partiview without any data by either double-clicking on the Partiview executable file or running it in a shell.

Adding Data Manually In Partiview, data can be manually added at the Command Line via the command:

```
add x y z
```

Of course, when data are read in via a file, which is the primary way in which you will want to import data, you need not preface it with the **add** command. However, because we are executing a Data Command at the Partiview Command Line, the **add** preface is required (as it is for all Data Commands issued from the Command Line).

Try adding a few data points in Partiview by entering these commands.

```
add 1 1 1
add -1 -1 -1
add 2 2 2
add -2 -2 -2
```

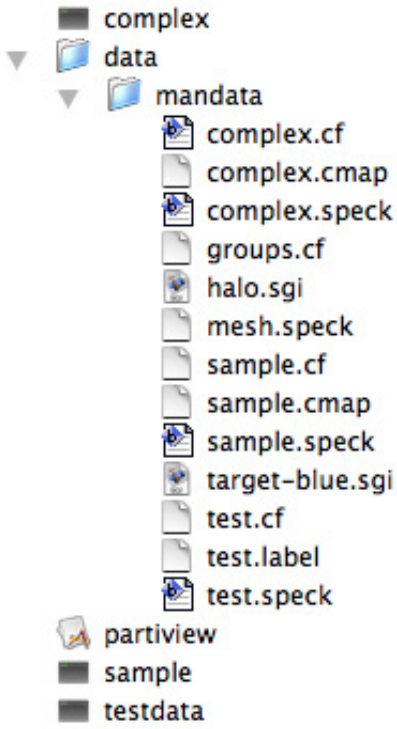
You will notice that nothing happened. This is because you need to tell Partiview about the display characteristics of these points, specifically, how bright they are. All display parameters have defaults and the default for a particle's luminosity is zero. You can confirm this by typing `lum` at the Command Line. This will report the current luminosity, which should read `lum-by 0() [0..0 mean 0 over 0]`—a zero value for the luminosity. You may set the luminosity to a constant value by typing the command

```
lum const 500
```

You may need to fly away from the Point of Interest a little to see all the points, which should form a line of four blue points.

Reading Data from a File While instructive, it is inefficient to type in all your points at the Command Line. So, let's read some data from a prepared data file. We will read in the file `complex.speck`, the complex version of the Test Data. First, we need to specify a file path, telling Partiview where the data files are.

Figure 3.2 – The hierarchy of files and folders in the Partiview User's Guide data set.



The Complex Data can be found in the `mandata` folder, which is located one directory level down from the directory that contains both the `data` folder and the Partiview executable.

To load the data, you must tell Partiview where the data files are located. This is accomplished using the **filepath** command. Because this is a Data Command, you must preface this command with **add** if you're going to use it interactively at the Partiview Command Line. Now that we know where the files are, we can issue the command

```
add filepath ./data/mandata
```

(NOTE: You may need to type the entire path to the folder, and spaces will need to be entered as a backslash-space.) This command will set the file path to include all files in the `mandata` directory.

Rather than read the data into the same data group that contains those points we just entered, let's read the data into a new group. A new data group may be created using the **g***N*[**=***alias*]. Let's call this data group "complex" and define it in Partiview this way:

```
g2=complex
```

You may have noticed the Group Buttons pop on, displaying buttons for both the original data you entered manually, group 1, and your new group, `complex`. Assuming the `complex` group is the active group (which you can confirm under the Groups Menu where you should see `[g2]`), we can then read in the data. Let's use the **read** command to import the data file `complex.speck`

```
read complex.speck
```

Upon reading this file (you may need to move toward the Point of Interest to see the data set), you will see that the labels pop on, but there are no data represented in the view (no points or polygons). If you look at the data file, you will see that we have combined the labels into the `.speck` file, rather than have two separate files. So you've just seen the labels pop on in the display, but there are no points drawn yet. Let's first adjust the look of the labels, then work on displaying the data points.

Once you fly away from the data cube, you will notice that the labels are so small that you can't read them. Let's see what their current size is and adjust them using the **labelsize** (or **lsize**) command. First, see what the current size is by issuing the **lsize** command without any arguments. Partiview should report `0.05` (the default size). This size is in the units of your coordinate system (if our data set is in meters, then the label height will be 0.05 meters). Change the size by typing the command

```
lsize *1.1
```

Once you enter this command, keep hitting the Enter (or Return) key until you reach the desired size; you are multiplying the size each time by 1.1. You can turn off the label axes (the individual axes on each label) by typing

```
laxes off
```

To clarify things, turn the group 1 data off. You can do this by left-clicking on the `g1` button, or typing `g1 off` in the Command Line. However you choose to turn the group off, be sure the `complex` data

group (group 2) remains the active data group. (If it isn't the active group, just right-click on its group button.) As a first step toward displaying points, let's give these data a constant luminosity of 1000 using the command

```
lum const 1000
```

The points appear, but seem to be randomly colored; let's see how Partiview is coloring the points by typing `color`, which invokes the **color** command without any arguments. It should report the following: `coloring-by 1(coloridx) 0.. 6 mean 2.19`, meaning that Partiview is using data variable 1 called `coloridx` for the color information. In Partiview, the default is to use the 5th column (or data variable 1) for coloring the particles and the 4th column (data variable 0) for luminosity information.

Let's see what data variables are available to us by typing

```
datavar
```

This will report all the data variables defined in the data file, which you can see in the `complex.speck` file. You may want to increase the size of the Console Window or use the scroll bar to see the full report. We see that the color information is in datavar 1 and is called `coloridx`. Even though Partiview has set that column to be the default color column, let's explicitly tell Partiview that we want to use the data variable `coloridx` to color our particles. We can use the **color** *datavar* command for this

```
color coloridx
```

Upon issuing that command, the colors did not change, so we were right in assuming Partiview was using the `coloridx` column to color the points. For more information on coloring objects, see "Coloring Particles and Objects."

We have now read in a data set and loaded it into a new data group. Let's now discuss what data groups are, how to create them, and how to manipulate data within them.

3.5 Data Groups

This section will review the methods for defining, activating, and manipulating data groups within Partiview. To discuss these concepts, we have created a config file called `groups.cf`. Let's load this file into an empty Partiview session using the **read** command:

```
read ./data/mandata/groups.cf
```

(You may need to type the complete path, or if you're continuing from the last section, type `read groups.cf`.)

Once you have the file loaded, you will see the familiar Test Data appear. Along with the test data button, you will also notice three other data group buttons, `complex1`, `complex2`, `complex3`. Using these data groups, we will run through some of the more common operations on data groups.

Defining a Data Group As we have shown in the previous section, a data group is defined using the **g***N* command; however, there are several variations on this command. Typically, it is wise to assign a name, or alias, to each of your data groups using the **g***N***=***alias* command. If you look at the `groups.cf` file, you will notice that we define each data group with the **object** command, such as, `object g1=test`.

How to Refer to Data Groups Data groups can be referenced using either the group number or the group name. We encourage the use of names simply because data often get shuffled around which then changes the group number for a particular data set. However, the name of that group can remain the same. Therefore, when using the Command Line, we propose referring to data groups by using the **object** *alias* command. For example, type this command in the Command Line:

```
object complex1
```

This changes the active data group to group 2 since `complex1` is defined as group two (`g2`). The **object** *alias* command is more powerful than simply selecting a group, you may also append group control commands to control how the group is displayed. For example, you can turn on the `complex2` group and change its color and luminosity using these commands:

```
object complex2 on
color const 1 .5 0
object test color const 0 0 1
label off
```

This gives you the tools to switch between data groups and execute commands from the Command line. Behind the scenes, the **object** *alias* command is changing the active data group, which is why the

following commands do not need the **object** *alias* command to preface them. However, there's more than one way to specify the active data group.

Setting the Active Data Group The active data group is the data group on which all group Control Commands will act. For example, if you want to change the color, luminosity, label size, polygon attributes, or any other data-group-specific parameter, you need to be sure you are acting on the appropriate group by activating that group.

There are several ways to set the active data group, these include:

- Selecting the group from the Groups Menu
- Right-clicking on the group button which will also display the group
- Using the command **object** *alias*
- Issuing the **g***N* command.

The quickest method is to use the group buttons. If you're using the Command Line, we discourage the use of the **g***N* command since group numbers often change. Rather, we prefer using **object** *alias* if you're using the Command Line.

More often than not, when you are trying to change the display of a data group and nothing is happening, it is because you are acting on the wrong data group and need to change to the appropriate group. Be sure to inspect the active data group indicated below the Groups Menu.

Operating on all Data Groups There is a command to operate on all data groups at once. The command **gall** will allow you to perform any group Control Command on all defined data groups. For example, if you turn on all data groups and fly away from the data so you can see each of the groups, you can brighten them all up by a factor of 3 using the **slum** command by typing:

```
gall slum *3
```

Or, you could turn on all the labels, resize them, color all data white, and then turn off all data groups using these commands:

```
gall label on
gall lsize *2
gall color const 1 1 1
gall off
```

If you use the **-v** option, `gall -v`, you will get a report of all data groups and their display status.

3.6 Importing Data

If you have your own data and would like to display it in Partiview, follow the steps in this section to visualize it in Partiview. Much of the description in this section is a consolidation of the previous two sections, presented in a step-by-step recipe.

To display your data in Partiview follow these basic steps:

1. Prepare your data in the Partiview data format and write that to a `.speck` file.
2. If you want labels on the points, include these in your `.speck` file or create a `.label` file.
3. Create a config (`.cf`) file with the desired Partiview commands to initialize the display of your data as well as the Partiview session.
4. If you wish to have automatic data loading upon starting Partiview, create a start-up file (`.bat` file for Windows, `.command` for Macintosh, `.sh` for Linux).
5. Run the startup file and your data should be visible in Partiview.

Now, let's look at each of these steps in detail.

Prepare Your Data The Partiview data format can be as simple as a file of x, y, z data. However, there are more complex data formats that result in more powerful data display and manipulation. Generally, a data point takes the form:

```
x y z datavar0 datavar1 datavar2 ...  datavarN
```

If you have data variables in addition to the position data, they must be defined at the top of the file. For example, if you have color data associated with each point, you can define that as `datavar` 0 using a **datavar** command at the top of the file, like

```
datavar 0 color
```

Or, if you have 20 different textures that you want to associate with your data and column 6 in your data file indicates which texture to use for that data point, then put the commands

```
datavar 2 texnum
texturevar 2
```

The first command tells Partiview to associate the data variable called `texnum` with column 6 (remember, the first three columns are always x, y, z, and column 4 is the zeroth data variable, so column 6 is data variable 2). The second command explicitly associates column 6 as the texture variable via the **texturevar** command. The best way to explore the file formats is to explore the supplied data sets and those of the Digital Universe.

Create a Label File (optional) The `.label` file takes the form:

```
x y z text yourlabel
```

where the **text** command is used to tell Partiview that anything after it should be printed at that specified point. The `yourlabel` label can have spaces and punctuation. For example, a typical line may look like the following:

```
20 30 40 text Alpha Ori
```

Then, Partiview would place the label "Alpha Ori" at the location $(x, y, z) = (20, 30, 40)$. Note that you can place label data in the `.speck` file, but it will increase the number of data points that Partiview thinks exists since each line is counted as a data point.

Create a Config File Create a configuration file to save the properties of the Partiview session as well as the properties of the data display. The properties of the Partiview session are set using the global display commands outlined in "Changing Global Properties" and include settings made with commands like **fov**, **interest**, **censize**, and **jump**, to name a few.

The data display properties are set using the group display commands and include commands to set the particle luminosity, the color of the data set, as well as starting with the points or polygons on or off, among many others. In addition, some other commands will be needed to read the files and set the correct file path. Please see the config files in the `mandata` directory for examples.

Create a Startup File The startup file will launch Partiview and load data sets as well as your configuration commands. This is done by a one-line file which contains the name of the Partiview executable followed by the name of the config file. The startup file must be in the same directory as the Partiview executable. For Windows, this file must have the extension `.bat`. So, the `yourdata.bat` file might look like this:

```
partiview ./data/yourdatafolder/yourdata.cf
```

For Macintosh users, the file must end with the `.command` extension and be executable. The format looks like:

```
./partiview ./data/yourdatafolder/yourdata.cf
```

For Linux users, the file may have any name but must be made into an executable via the `chmod` UNIX command. The file might look something like this:

```
./partiview ./data/yourdatadirectory/yourdata.cf
```

Launch Partiview Run the startup file you just created to launch Partiview and your data. If your data do not appear in the display, a number of things could be wrong. Most likely, you need to set a **lum** and **slum** value. Check the Console Window for errors.

4

Changing Global Properties

Global view attributes are those settings that affect the entire view—functions that span all data groups. These are not typically used for changing how particles are displayed, but are used to describe the overall view in which your data groups are drawn. Among these settings are the field of view, the background color, and setting the camera position.

4.1 Background Color

Changing the background color in Partiview is simple. Using the **bgcolor** command, you may set the background to any color you wish by specifying a red, green, blue (R, G, B) color. The default value is black [(R, G, B) = $(0, 0, 0)$], but let's say you want to change the background to red, this can be accomplished via the command:

```
bgcolor 1 0 0
```

Try this in Partiview. Also, try changing to varying levels of gray by issuing the command **bgcolor** with only one argument. When only one argument is specified, all three arguments, R, G, and B are set to that value. All colors range from 0 to 1, so `bgcolor 0` gives you a black background, `bgcolor .5` yields a gray, and `bgcolor 1` is white.

4.2 Changing Your Position

In Partiview, there are two ways to change your position, either fly to a specified location, or use the **jump** command to instantly go to a point (x, y, z).

Determining your Position In order to see information on your current position, the **where** command generates a report to the Console Window. By typing

```
where
```

Partiview generates four lines of information describing your position. These are

```
camera at x_w y_w z_w (w) x_g y_g z_g (gN)
looking to x̂_w ŷ_w ẑ_w (w) x̂_g ŷ_g ẑ_g (gN)
jump x_w y_w z_w R_x R_y R_z scale
c2w:  a_xx a_xy a_xz 0 a_yx a_yy a_yz 0 a_zx a_zy a_zz 0 x_w y_w z_w 1
c2obj:  b_xx b_xy b_xz 0 b_yx b_yy b_yz 0 b_zx b_zy b_zz 0 x_g y_g z_g 1
```

where x, y, z are the values of your current position, $\hat{x}, \hat{y}, \hat{z}$ are unit vectors that describe the forward direction vector, R_x, R_y, R_z are the rotation angles about the x, y, and z axes, and the `c2w` a_{ij} and `c2obj` b_{ij} coefficients describe a transformation matrix from camera (your view) to world coordinates. The `w` subscript signifies a number that is relative to the world coordinates, while a g subscript is relative to the active data group `g`N.

If we look at Partiview without any data and only the Point of Interest marker in our view, we can explain some of these numbers. The red axis points in the $+x$ direction, the green axis points in the $+y$ direction, and the blue axis points in the $+z$ direction. The $\hat{x}, \hat{y}, \hat{z}$ values are unit vectors (they add to 1) that describe your forward vector. For example, if you line up squarely on the red axis, so that you're looking down the red axis and the blue and green axes form a plane parallel to your viewing plane, then the forward direction is just about all in the negative x direction, so $\hat{x}$ should be very close to -1, with very small components in the y and z directions. Similarly, flip 180° and $\hat{x}$ would be +1. Position yourself at a 45° angle in all directions, and $\hat{x} \sim \hat{y} \sim \hat{z}$.

Resetting the Camera Position You can reset the camera position using the keyboard shortcut keys `cw` in the Viewing Window (type the letter `c`, then `w` in the Viewing Window, not the Command Line). This transports you to this location (as reported by **where**):

$(x, y, z) = (0, 0, 3)$
$(\hat{x}, \hat{y}, \hat{z}) = (0, 0, -1)$
$(R_x, R_y, R_z) = (0, 0, 0)$

looking down the $+z$ axis from 3 distance units away with the $+x$ axis pointing to the right and the $+y$ axis pointing up.

Jumping to a new position If you'd like to go instantaneously to some point in the data, you may use the **jump** command to specify an x, y, z and R_x, R_y, R_z of your choosing. Let's define these rotation angles R_x, R_y, R_z more throughly. If you reset the camera position (with the `cw` keys), then the rotation angles are all zero. In Rotate Flight Mode, rotate with the right mouse button so that the x (and y) axis is about 45° from where it was after resetting. Now, issue the **jump** command with no arguments and you will see that the value of R_z is around 45°. You have rotated about the z axis (which is pointing directly at you) by about 45°.

Reset the camera position again. Now, in the Orbit Flight Mode, rotate about the x axis by placing the mouse at the top of the y (green) axis and pulling down along this axis with the left button pressed. This should keep the x axis stationary. Now, we expect the value of R_x to change since we're rotating about the x axis. Does **jump** prove us right?

As another example, if you'd like to go 10 units out the x axis and look back on the Point of Interest, you will need to set $(x, y, z) = (10, 0, 0)$ and $(R_x, R_y, R_z) = (0, 90, 0)$. Why must $R_y = 90°$ and not 0°? Well, reset the camera position using the `cw` keys. Now, imagine increasing your x position by 1 unit (i.e., moving in a direction parallel to the $+x$ axis by one unit). What will happen? The Point of Interest will move to the left 1 unit since we've moved the camera to $x = 1$. Using **jump** first inspect your current position: $(x, y, z) = (0, 0, 3)$. Now, move one unit out the x axis (maintaining your z position) using this command:

```
jump 1 0 3
```

The rotational angles will remain the same if they are unspecified. As you see, the Point of Interest has moved off to the left and we have not changed the direction we're looking. We only moved a few steps to the right, without moving our head.

4.3 Field of View

In any graphics package, there exists a space inside which your data is displayed. In Partiview, this space is shaped like a pyramid. You are looking from the tip of the pyramid and everything you see in the Viewing Window is inside its walls. This pyramid can have a small height from base to apex, rendering it relatively flat, or its height can be large, rendering a tall, slender pyramid. The width of the pyramid is set by the field of view command (**fov**).

In order to examine the field of view in Partiview, we are going to draw a sphere that is made up of latitude and longitude lines at 10° intervals, then look out from the center of that sphere. This is somewhat complicated, but provides a visual understanding of the field of view, as well as an introduction to drawing ellipsoids.

The **ellipsoid** data command is one of the more complex commands. Rather than get bogged down in that command now, we will draw a simple ellipsoid and explain the command in more detail later in this guide.

First, let's start Partiview without any data (or start with a data set pre-loaded and turn off all the data). We will draw a sphere around our Point of Interest that has a radius of 1,000 units and vertices every 10° in both latitude and longitude. This is achieved via the command

```
add 0 0 0 ellipsoid -r 1000 -c 18 -s wire -n 36,19
```

where we have chosen a color index of 18 (referring to the default color map), a wire-frame drawing style, the number of longitude lines to be 36, and the number of latitude lines to be 19 (this number includes the poles). This gives us a sphere with lines every 10°.

Now, change the clipping planes (we discuss this in the next section) to accommodate the sphere using the **clip** command

```
clip .1 100000
```

and pull away so that you see the entire sphere.

Next, let's draw an object in the *xy* plane so we can easily see which line of latitude is the equator. The object we draw will be a flat, disk-shaped object with the *z* radius equal to zero. Draw this via the command

```
add 0 0 0 ellipsoid -r 1000,1000,0 -c 18 -s wire -n 36
```

This places a round plane (R_x = 1000, R_y = 1000, R_z = 0) with 36 vertexes in the x and y directions, drawn with the wire style.

Now jump back to the Point of Interest using

```
jump 0 0 0
```

(or use the Home Toggle Button) and notice the bright white line along the equator (you may have to pan around a little). It's brighter because the two ellipsoids are drawn on top of one another. In the Fly Flight Mode, move the equator down so that it is at the bottom of your display. You may need to rotate it level (switch to Rotate Flight Mode and use the right mouse button) and switch back to Fly to move it down again.

Because these lines of latitude are in 10° intervals, you can count how many degrees the field of view is from the bottom of the Viewing Window to the top. The number of degrees from top to bottom should equal the value for the field of view. Let's be sure we know what the field of view is by setting it to 40° using the command

```
fov 40
```

Now you should count 40° from top to bottom (you may need to pan a little to see all the lines). Note, you can also change the field of view using the Field of View Slider.

Allowable values for the field of view range between (but are not equal to) 0° and 180°. If you change the **fov** to 90°, you should be able to see the pole and the equator (provided the pole is at the top of the Viewing Window and the equator is at the bottom). Obviously, this begins to distort the view pretty severely.

Let's load the Test Data by reading the `test.cf` file to see the effects on the data display. This will reset the field of view, but that's okay—we can check the current value, then turn the ellipsoids off using these commands:

```
add filepath [path to partiview]/data/mandata
read test.cf
fov
ellipsoid off
```

We see that the field of view is 50°. Also, you might have noticed that we didn't use the **add** command in front of the **ellipsoid off** command. This is actually a control command version of the **ellipsoid**

command. We discuss the difference between these in "Spheres and Ellipsoids." Briefly, **ellipsoid off** turns the ellipsoids in the active data group off. Use the **ellipsoid on** command if you wish to re-display them.

If you set the field of view back to 90° and pan around the sky in Fly Flight Mode, you will notice the distortion on the edges of the display. Change the field of view to something impractical, like 160°, then see the resulting effect. Finally, turn the ellipsoids back on and fly around to see the effects on the entire display.

4.4 Clipping Planes

We mentioned in the previous section that Partiview draws data inside a pyramid and the location of the triangular sides of that pyramid (the limit of your peripheral vision) are determined by the field of view. In this section we will discuss the apex and the base of the pyramid. These are called clipping planes in computer graphics and are set using the **clip** command.

When we said the view is specified by a pyramid, we were not 100% correct. In reality, the tip of the pyramid has been removed, leaving the pyramid with a flat top. This flat plane, parallel to your screen, is called the near clipping plane, while the base of the pyramid is called the far clipping plane.

The **clip** command takes the floating point arguments *nearplane* and *farplane*. The *nearplane* argument is the distance from your screen to the near clipping plane, the top plane of the pyramid. Nothing will be drawn between your screen and this near clipping plane, so you will want this to be a small value. *farplane* is the distance from your screen to the base of the pyramid. Beyond this plane, nothing is drawn. You want this value to be large enough to include your data. The units of these arguments are in the units of your data set. For example, if we're looking at stars, the units might be in light-years.

Experimenting with the clipping planes To experiment with setting the clipping planes, let's start the Sample Data. If we assume the units of these data are in meters, then typing

```
clip
```

will report the current values of 0.1 meters and 100,000 meters for the near and far clipping planes, respectively. This means the height of our pyramid is 100,000 meters minus 0.1 meters. These values

are set in the `sample.cf` file and are reasonable values based on the scale of these data. However, let's change these values and see the effects.

Watch the data as you change the near clipping plane to 10 meters by typing:

```
clip 10
```

(Notice that we did not specify the *farplane* argument—Partiview assumes the current value if the argument is not specified.) Upon typing this command, the Point of Interest and a few nearby points disappear. They are within 10 meters from your vantage point.

Now fly forward or backward from your current location using the right mouse button. Notice how data appear if you're flying backward, or disappear if you're flying forward. If you increase the near clip to some large value, you would expect more and more data to disappear as the near clipping plane approaches the far clipping plane.

Now perform a similar exercise for the far clipping plane. Let's change the clipping planes back to the original values of 0.1 and 100,000 so that we recall the original view, then change the far clip plane to 100 using these commands:

```
clip .1 100000
clip .1 100
```

A significant number of data points have disappeared since we have brought the far clipping plane closer to us. Here, our pyramid is very flat, with a height of only 100 meters.

Note, we can also change the far plane without having to know the near plane. If you substitute a non-numeric character for the *nearplane* argument, it will remain the same. For example, change the far clipping plane to 10,000 using this command:

```
clip - 10000
```

This leaves the near clipping plane at 0.1 meters and changes the far clipping plane to 10,000 meters.

Clipping and your computer OpenGL behaves differently for different graphics cards and operating systems. Without being experts on this subject, there are a few issues that have been noted with Partiview. One is that on some computers, any *nearplane* value under 1 produces unpredictable results. If data or polygons aren't being drawn as you expect, try setting the near clip to 1. Also, if the ratio of

farplane to *nearplane* exceeds 10,000, this may produce unwanted effects, such as the blinking of distant particles. However, most modern hardware has no problem with a high clip plane ratio.

4.5 Point of Interest

Let's start with a fresh version of the Test Data by restarting Partiview. The Point of Interest is the point about which orbiting and rotating take place. It is marked with the red, green, and blue, Cartesian axes (assuming **censize** > 0), where red is the $+x$ axis, green is the $+y$ axis, and blue is the $+z$ axis. The location of the Point of Interest sets the flight scale in the logarithmic flight modes (zero speed on the point and increasing speed as one moves away from the point), as discussed in "Flight Controls."

Adjusting the Size of the Point of Interest If you wish to adjust the size of the Point of Interest, this is accomplished using the **censize** command. Typing `censize` without any arguments reports the current size of the Point of Interest. Inside the config file, this size is set to 0.5, and this is what Partiview should return. However, if you wish to increase the size to span the cube, set the size to 1 using the command `censize 1`. Conversely, if you want the Point of Interest marker to disappear, set the size to 0.

Selecting a New Point of Interest Next, let's move the Point of Interest. Set the Point of Interest size to 0.5 again and choose a particle that you want to set as the Point of Interest. There are two ways to set the location of the Point of Interest equal to that of a particle's location. While placing the cursor on top of the particle, either

- Press the `[Shift]` key while you click the middle mouse button on a three button mouse
- Press the `[Shift]` key while you type the letter `p`

Try this with a few points. You might notice that if two points are close together, then Partiview may have trouble choosing which point you want. In this case, move the data around a little to isolate the point. You will see a remnant of the original Point of Interest at $(0, 0, 0)$, this is normal.

Setting the Point of Interest If you know an x, y, z location where you want to place the Point of Interest, you may enter it via the **center** or **interest** commands. These commands are identical and take

x, y, z values to set the Point of Interest to some arbitrary point. Try this by setting the Point of Interest to a point outside the data cube, like

```
center 2 0 0
```

Now the Point of Interest should be above the data cube along the red axis and you will now orbit about this point.

4.6 Resizing the Viewing Window

Partiview can run at any screen size you wish. In this section, we will discuss how to change the size of this window and how to separate the GUI from the Viewing Window, allowing you to run Partiview full screen without the appearance of the GUI.

Changing the Window Size The window size can be set to your liking using the **winsize** command. This command takes the optional arguments *xsize* and *ysize*. Start a Partiview session and let's experiment. If we type

```
winsize
```

Partiview will report the current window size. To set the size, supply the *xsize* and *ysize* arguments; for example,

```
winsize 400 400
```

sets the window to be square. We mentioned that the *ysize* argument was optional. If left out, the same aspect ratio is maintained when given a new *xsize*. Try decreasing the window size with an *xsize* of 200, and see if the window remains square.

```
winsize 200
```

The window does indeed remain square—Partiview reports a window size of 200 by 200.

Setting the Window Location The location of the Viewing Window on your screen is set using the $\pm xpos \pm ypos$ arguments. These arguments set the location of the screen in pixels from either the top left corner (using the '+' arguments) or the bottom right corner (using the '−' arguments). For example, if you want the Viewing Window to appear in the upper left corner with a small space between the window and the edge of the screen, you might set the *xpos* to 20 and *ypos* to 40 in the command:

```
winsize 300 400 +20+40
```

Note that the window begins at the top of the GUI and does not include the window bar at the top that your operating system attaches on. Using a minus sign in place of the plus signs would move the window in the lower right corner of your screen.

Separating the GUI from the Viewing Window To separate the GUI from the Viewing Window, type the command **detach** at the Command Line and Partiview will separate into two windows. This allows for more flexibility in distributing windows on your screen. Both **winsize** and **detach** are control commands so, if you wish to issue them in a file, you will need to preface them with the **eval** command.

Running Partiview Full Screen By now, you have probably figured out how to run Partiview in full screen mode. Simply set the **winsize** to be your screen resolution and use **detach** to move the GUI off screen or minimize it to the bottom. Try this by issuing the commands

```
winsize xsize ysize +20+20
detach
```

replacing your screen resolution for the *xsize* and *ysize* arguments. You can now minimize the GUI or place it at the bottom of the screen, setting the window to be "always on top" if your operating system allows it.

With the GUI minimized and out of sight, it is still possible to type commands into Partiview. You gain access to the Command Line via the `[Tab]` key—control is switched such that the Command Line has focus once the `[Tab]` key is pressed. Let's make the window smaller using the **winsize** command along with `[Tab]` in this way:

```
[Tab]
winsize 500
```

This should reset the Viewing Window to be 500 by 375. This process can be used to enter any command, whether you want to change particle luminosity, the size of the Point of Interest, or the color of an object.

The .partiviewrc File Your window setting, and other preferences, can be placed in a config file called `.partiviewrc`. Partiview reads this file on startup before reading any other files (like `.cf` or `.speck` files). Depending on your operating system, creating this file can be difficult, particularly for Windows. We suggest creating and saving the file in a text editor. In addition, Windows and Macintosh consider filenames beginning with a period to be hidden system files, further complicating this little file's existence. . In UNIX-based environments, this is trivial though.

Once you have this file, you can place the **winsize** and **detach** commands in it to run all Partiview sessions full screen. Other commands can also be placed in this file as long as you want them to apply to all data you display. For example, you could set the field of view here, or your initial position. This file *must* be in the same folder that contains the Partiview executable file for it to take effect. We often use this file to start Partiview in full screen and either move the file out of the Partiview folder or rename it when we want to run in work mode.

4.7 Exporting the Display

It is sometimes useful to save the graphics output from Partiview by taking a screenshot. You can do this using the **snapset** and **snapshot** commands.

The **snapset** command sets the output filename for the image. Output images are written into a compressed Portable PixMap (`.ppm`) file by default. However, other image types can be saved (`tif`, `jpeg`, `bmp`, etc.) provided you have the `convert(1)` program installed and in your path (this applies to UNIX-based systems (Linux and Macintosh), Windows users will need to convert the image using another image processing program or install a UNIX emulator).

The **snapset** command takes the *filestem* argument. You simply supply an ordinary name, to which Partiview will append the frame number (starting at zero) along with the file extension (`.ppm`) then the gzip compression extension (`.gz`). Try this in Partiview with these commands:

```
snapset foo
snapshot
```

You should see a file in your Partiview folder called `foo.000.ppm.gz`. More details on these commands can be found in the command listing.

5

Changing Group Properties

Each of these sections describes how you can alter the view of a particular data group. Each data group has settings that pertain to the points, the polygons, the luminosity, and color of the particles—the appearance of data in a particular group. There are many other group-specific commands that we will discuss in the following few chapters.

5.1 Drawing Points

If you inspect the Test Data in Partiview, you will notice that the points that make up the cube vary in size. This is because Partiview is intelligent in how it draws particles, taking into account your distance from the particle to properly draw the perspective.

What determines this size though? The answer to this question will be discussed in the next several sections, culminating in a description of the luminosity of particles. The problem is this: the commands that affect the size (i.e., the luminosity) of a particle are dependent on knowing the commands that set the specific attributes of the particles. However, these attributes cannot be set outside the context of the various commands that set the luminosity. So, we have decided to introduce the specific attributes which will then be discussed in concert with one another in the section on luminosity.

In this section, we intend to introduce those attributes that affect points, the basic graphical element that represents data in Partiview. If you load the Test Data, you will notice that the points are displayed. These can be turned off either by clicking on the Point Toggle Button (green indicates the points are on),

or using the **points** command. This command by itself toggles the points on and off, or you may append the **on** or **off** commands (as in `points on`) to explicitly tell Partiview to turn the points on or off.

We can set the range of apparent sizes for the points using the **ptsize** command. The range can be defined using the *minpixels* and *maxpixels* arguments. If you type

```
ptsize
```

Partiview will report to you the current settings on the range from *minpixels*, representing the faintest points that will ever be drawn, to *maxpixels*, which represents the brightest points that will ever be drawn. We say "that will ever be drawn" because this sets the size of points across all luminosity conditions, from up close and bright to far away and dim. What you are setting with **ptsize** is the range of pixel sizes in which points will be drawn and this depends on the distance to the point as well as its intrinsic brightness. If the luminosity settings determine that a point's size is less than *minpixels*, then that point will not be drawn.

The Test Data is pre-loaded with the command `ptsize 0.05 10`, which produces points that range in size from 0.05 to 10 pixels. Of course, when you view the Test Data, you aren't seeing points of size 0.05 pixels. However, if you zoom away from the data cube, you will see the points shrink smaller and smaller until they disappear altogether.

Experiment with the **ptsize** command arguments using the Test Data by initializing your view with these commands:

```
label off
censize 0.1
slum 0.001
jump 0 0 10 0 0 0
```

This should make the points extremely small (you may even have to squint to see them or increase their **slum** value a little). If you adjust the *maxpixels* argument,

```
ptsize - 100
```

you will notice that nothing happens to the points. This is consistent since the points are far away; we are seeing the effect of the *minpixels* argument as the points fade out. So, let's adjust the *minpixels* argument up to 0.5 from its current 0.05 value using the command

```
ptsize 0.5
```

The *maxpixels* argument will remain the same if nothing is specified. You will notice the points get brighter, but some of the points disappear or turn on and off if you are orbiting around the cube. Now make *minpixels* equal to one:

```
ptsize 1
```

Now you will see that the points get brighter (larger) but more of them disappear. What is going on here?

Partiview computes the size of a particle based on its luminosity settings and distance. Those particles that have computed sizes smaller than *minpixels* are randomly subsampled. This is to prevent having hundreds of points of equal luminosity and distance pop on simultaneously as you fly through a complex data set with thousands of points. With a random subset chosen, they pop on gradually and less conspicuously. If you continue to increase *minpixels* by intervals of 1, you will see different subsamples displayed.

How can you prevent this from occurring? As with everything in this section, it depends on a combination of factors. If you want your data to fade out gradually as you fly away from it, then you will want to set a low *minpixels* argument. If you want large points, then you'll want to set the luminosity factors high. For example, if you are seeing a subset of points now, increase the **slum** factor by moving the Slum Slider up a little. The points now have enough intrinsic luminosity so that their computed size is greater than *minpixels*.

Generally, a good *minpixels* value is around 0.05, while the *maxpixels* size is more aesthetic, depending on how prominent you want the points to appear. It is likely that your graphics card will impose some upper limit to the point size, perhaps as low as 10 pixels.

Rendering the Point Points can either be drawn as squares, which are more efficient for your computer to draw, or circles, which are more computationally taxing to draw. These two rendering settings can be toggled using the **fast** command. Starting with a fresh Test Data session, or slumming the current particles up so that all points are clearly visible and large, type

```
fast
```

and the points should become squares with the report `fast on` followed by the **ptsize** settings. Issuing this command again will toggle back to `fast off`, drawing round points and telling you this is the better

rendering option, although it comes at a performance cost.

Because points are somewhat limited in their use, we will now discuss how to place a polygon on a point, giving you more power in how you choose to display your data.

5.2 Placing Polygons on Points

A polygon is a two dimensional, multi-sided surface. In Partiview, polygons can be used to pictorially represent different data types, emphasize a particular data attribute, such as size or orientation, or even to build three dimensional objects. In this section, we will explore how to place polygons on points and alter their characteristics.

Let's first explore polygons by starting Partiview without any data pre-loaded. Once you have Partiview up, add a data point at the origin and set some parameters for that point, such as:

```
add 0 0 0
lum const 1
slum 1
color const 1 1 1
censize 0
```

With the Point of Interest axes off, you can see our lone white point (although it may be quite dim—you might need to move closer to see it). Now, let's turn on the polygons using the Polygon Toggle button. Nothing happens. Why?

Sizing Polygons using polysize First check the size of our polygons using the **polysize** command. Type

```
polysize
```

to see that Partiview reports zero. Let's try setting the size to 1:

```
polysize 1
```

Has the entire display turned gray? Perhaps this is too large, let's fly out a bit and find the edges of the polygon. Okay, now we can at least see the edges of the polygons, although it seems awfully large. How does the polygon size relate to our coordinate system?

Let's explore the size ratios to discover the scale for polygon size. First, let's change the number of sides the polygon has. If you issue the **polysides** command, without its integer argument, you will see that the default is 11. Let's change this to 4,

```
polysides 4
```

making it a square. Now, set the Point of Interest size to 1 and see how large the axes become relative to the polygon

```
censize 1
```

We see (if we fly away from the origin) that the unit distance is tiny relative to our polygon (recall if we set the center size to 1, the length of each axis from the origin is 1). With the **lum const**, **slum** and **polysize** all being equal to one, something must be creating this large size, or the poly size is just naturally larger.

Is the poly size an order of magnitude larger? Try

```
censize 10
```

This seems to work, so with the luminosity settings all equal to one, and the polygon size equal to one, the actual size of the polygon in our coordinate system seems to be 10 units of half-length. If our theory is true, we should be able to increase or decrease the **lum const**, **slum**, and/or the **polysize** by an order of magnitude to see if this relationship is true. Let's increase the **lum** by a factor of 10 and decrease the **slum** by a factor of 10.

```
slum 0.1
lum const 10
```

Does the size of the polygon stay the same relative to our center size? Yes. Also, try setting the **lum const** up and see if an increased center size matches:

```
lum const 1
slum 1
lum const 10
censize 100
```

Similarly, an order of magnitude increase in size with an order of magnitude increase in luminosity via the commands

```
polysize 0.1
censize 10
```

also scales properly. So, we now know the size of the polygon relative to the luminosity and luminosity scaling factor and can size the polygon according to this formula. The table below shows how the various elements affect the size of the polygon.

Table 5.1 – Scaling relationship between **lum const**, **slum**, and **polysize** which determine the size of polygons. We adjust the luminosity and its scaling factor to keep **polysize** equal to 1. If the value of **polysize** were 0.1, then the actual size would decrease by a factor of ten and, similarly, if the value of **polysize** were 10, then the actual size would increase by a factor of ten.

polysize	lum const	slum	Actual Size of Polygon
1	1	1	10
	10	1	100
	1	10	100
	10	10	1000
	1	100	1000

To have small polygons on data that are bright, increase the **lum** and **slum** values and set the **polysize** to be very small. To have large polygons on data that have low **lum** and **slum** values, increase their **polysize**.

Sizing Polygons According to a Data Variable In addition to setting the polygon size explicitly using the **polysize** command, you may also assign a column in your data to represent the polygon size. This is accomplished using the **polylumvar** command. If you type

```
polylum
```

Partiview will report `polylumvar point-size`, meaning the size of the polygon is being set according to the luminosity of the particles (as we discussed above). However, you can set the size of the polygon to scale from some data variable by specifying the **datavar** name. So, for example, if you start the Sample Data set and turn the textures off (use the Texture Toggle Button) so that you have a view of the

square polygons, you can set the size of these polygons based on a column in the data file. Let's get a report of the defined data variables in the file by typing

```
datavar
```

Since the `coloridx` seems like a reasonable range (1–10), we will change the polygon size to scale to these values with the command

```
polylum coloridx
```

This increases the size of the polygons, each one sized according the the value of the `coloridx` data variable. Fly out of the data set to see all of the various sizes that belong to each color. Now the polygon size depends on the color, rather than the distance from each point. So, you may find a large white polygon and a small purple polygon at similar distances, but their polygon size is different. Type

```
polylum point-size
```

to return to sizing by distance.

Setting the Properties of the Polygon Use the **polysides** command to set the number of sides a polygon is drawn with. The default value is 11, which draws round-ish polygons. However, the most computationally efficient polygon shape to draw textures on is a square; therefore, if you're drawing textures on your polygons, use `polysides 4` if appropriate. Values for the number of sides on a polygon range from 3 (triangle) up to 16 (sometimes called a hexadecagon).

The Opaqueness of the Polygon The opaqueness of the polygon can be set using the **alpha** command. **alpha** takes on values between 0 and 1 with a default of 0.5. A value of 0 corresponds to zero opaqueness (complete transparency or invisibility) and a value of 1 corresponds to a polygon that is completely opaque (and block your view of anything behind it). A value of 0.999 will make the polygon quite bright without making it fully opaque. Try setting **alpha** to 0.2 to see more transparent polygons. Also, try using the Alpha Slider to quickly adjust the transparency.

The polyminpixels Command The **polyminpixels** command allows you to set limiting sizes that restrict the size of a polygon or determine when the polygon should be drawn. The *minpixels* argument

sets the minimum size in pixels that a polygon must be before it is drawn. The *maxpixels* argument sets a clamp on the size of the polygon so that increasing the luminosity beyond this threshold will not increase the size of the polygon.

Setting the Polygon Orientation By default, polygons face the screen at all times; however, you can fix their orientation in space if that is desired. This is useful when texturing polygons that represent a fixed object.

The polygon orientation is specified in the data file as a series of 6 numbers that describe two orthogonal vectors, $\vec{u}$ and $\vec{v}$, in the plane of the polygon. The first of these 6 columns is defined in a **datavar** command and the orientation is set using the **polyorivar** command. A typical data file might look like:

```
datavar 0 lumin
datavar 1 color
datavar 2 txnum
datavar 3 orient
texturevar txnum
polyorivar 3
0 0 0 10 2 1 1 2 0 -2 1 0
1 1 1 40 1 1 1 0 1 0 1 0
```

where the first three columns are x, y, z, the fourth is the luminosity, the fifth is the color index, the sixth is the texture number, and finally columns 7–12 are the 6 numbers that make up the orientation of the polygon. Notice how the **datavar** and **polyorivar** commands are used to define the orientation information. Some sample orientations are shown in Table 5.2.

If you wish to have a mixture of screen-facing polygons along with specific orientations within one data set, you may use all 9s to specify that you want a particular polygon to be screen-facing.

You may also toggle between screen-facing and specified orientations using the **polyorivar** command at the Command Line. Typing

```
polyorivar -1
```

toggles to all polygons facing the viewer. You may go back to your orientations by using the **datavar** number specified in the data file. Using our example above, you would type:

```
polyorivar 3
```

Table 5.2 – Examples of the $\vec{u}$ and $\vec{v}$ vectors that determine the polygon orientation.

u_x	u_y	u_z	v_x	v_y	v_z	**Polygon Orientation**
1	0	0	0	1	0	*xy* plane
1	0	0	0	0	1	*xz* plane
0	1	0	0	0	1	*yz* plane
0	1	0	0	0	10	*yz* plane with $R_z = 10 \times R_y$
1	0	1	0	1	0	rotated about *y* axis by 45°
1	2	0	-2	1	0	rotated square about the *z* axis
9	9	9	9	9	9	Polygons are always facing the screen

If you specify any number other than -1, Partiview will assume the orientation information begins at that column. For example, if you entered `polylumvar 0` the orientation of the polygon will be determined by columns 4–9, whether the data are relevant or not.

5.3 Drawing Textures on Polygons

Often we wish to represent a class of objects as a single polygon so that they are easily seen. However, sometimes we want to place a texture on our polygon to create a sense of realism. In this section we will describe how to place a texture on a polygon and discuss some of the basic settings that determine how the texture will be drawn.

Adding Textures To add textures to your polygons, all you need to do is include one or more **texture** data commands in the data file. The **texture** data command takes as arguments the image file name and texture number you want to assign that image to (as well as many options which are discussed in the command listing. Then, along with the **datavar** and **texturevar** commands, the textures are drawn on the specified polygon. For example, your data file might look like this:

```
datavar 0 txnum
texturevar txnum
texture 30 image1.sgi
texture 31 image2.sgi
0 0 0 30
1 1 1 31
-1 -1 -1 31
0 0 2 30
```

where we have defined data variable zero as the column to hold our texture information as specified by **texturevar**. As a warning though, there appears to be a bug in the software in that it doesn't always load the correct texture or it will load two textures and display them under various luminosity conditions. This may be due to the ordering of texture numbers in the data. A possible solution involves insuring the data are ordered by texture number, but no thorough investigation has occurred on this bug.

Turning textures on and off The easiest way to turn the textures on and off is to use the Texture Toggle Button. However, the **texture** Control Command will also toggle the textures on and off. This is accomplished by issuing

```
texture
```

in the Command Line or within a config file. You may append the **on** or **off** commands to **texture** as well. Note that the command in the previous paragraph to assign a texture number to an image file is the Data Command and would require an **add** preface if used at the Command Line.

Scaling Textures Textures can be scaled to fit their polygon, appear small, or appear large, overflowing the polygon edge. The command to change the texture scaling is **txscale**. The default value is 0.5, which is a perfect fit (the image is fitted to the polygon) if the number of polygon sides is 4. Using a texture scale smaller than this will stretch the image so that it overflows the polygon size and is truncated at the polygon's edge. Using a scale larger than 0.5 shrinks the texture on the polygon. Typically, there is no need to change this value from the default 0.5.

5.4 Setting the Luminosity of Particles

When we talk about the luminosity of a particle, we are really referring to its size, or more specifically, how large or small the graphics engine draws the point. You may have particles that you want large when they are nearby, but disappear when they reach a certain distance. You may want your polygons to disappear at some distance but your points to remain visible far away from the viewing location. The size of the points and often the polygons are subject to a combination of settings that determine its overall luminosity. We will begin by describing each of these commands and then wrap up at the end with a discussion on our approach to luminosity in Partiview.

The lum const Command The **lum const** command sets all particles in the active data group to a constant luminosity specified by the optional argument *L*. Here the size is directly proportional to the value of *L*.

The lum *datavar* Command Rather than set all particles to a single luminosity value, it is often useful to set the size of the points relative to some attribute of the data (like intrinsic brightness, for example). This is achieved using the **lum** *datavar* command. Provided you have a column of data that represents the luminosity, you may set the size of your particles to be proportional to the value of these data. If you have defined a data variable in the data file, such as

```
datavar 0 lumin
```

then you can use that to set the luminosity using the command

```
lum lumin 0 1
```

which reads the data in the column pointed to by `lumin` and renormalizes it over the range 0–1. The default for *min max* are the minimum and maximum values of the data variable, however, if the range 0–1 is specified, then the column data are renormalized over those values.

To explore these two types of **lum** commands, let's bring up the Complex Data. If you type

```
lum
```

Partiview will report `lum-by 0(lumin) 0 1 [0.5..50 mean 25.1 over 27]`. Translation: the luminosity is being read from data variable 0, called `lumin`, and ranges over 0 to 1. The actual `lumin` data in the file ranges from 0.5 to 50 and has a mean of 25.1 over the 27 data points. Now, set the particles to a constant luminosity of 1

```
lum const 1
```

Now all the particles have the same size, disregarding perspective effects. You may toggle back and forth between a constant luminosity and one set by the data variable `lumin` by issuing the commands

```
lum lumin
lum const
lum lumin
```

Luminosity Scale Factors Given the luminosity from either of these two **lum** commands, a scaling factor is multiplied to brighten all or a portion of particles in the data group. These scaling factors are:

psize This command is a global scale factor on the luminosity. If you type **psize** in the Command Line it will report the current value of this scale factor. A small value will draw the points smaller, reducing the range over which the points can be seen. If you'd like to be able to fly far away from your data while continuing to see the points, then you'll want a high value for **psize**. When you are setting the luminosity with the **lum const** command, **psize** is essentially equal to the **lum const** value. For example, the particle size will be equal if **lum const** is 1 and **psize** is 10 *and* if **lum const** is 10 and **psize** is 1. Note that **psize** has no effect on the size of the polygons.

slum This is a scaling factor that is relative to a particular data variable. For example, in the Complex Data, we have defined the `lumin` and `coloridx` data variables. We can set a **slum** scaling for each of these data variables in the following commands:

```
lum lumin 0 1
lum coloridx 0 1
```

Each of these "lums" can have separate scalings via the **slum** command. Let's set these scalings first by selecting the **lum** we want to scale, then setting the scaling factor using **slum**:

```
poly on
lum lumin
slum 0.1
lum coloridx
slum 0.5
```

You may notice that the size of the points vary depending on the value of the data variable. In the `lum lumin` case, all the points appear to be the same size since their **slum** value has saturated our **ptsize** settings. However, with the smaller **slum** value in the `lum coloridx` case, we see that points of higher color indices have higher luminosity. The polygons show the effects of these slum values clearly. Although the `coloridx` **slum** value is higher, the polygons are smaller due to the intrinsic brightness from the `coloridx` data in the file.

Particle Sampling If you are sampling your data using **every**, the remaining particles will be brightened to keep the total brightness the same.

Distance of the Particles Partiview determines the distance from the viewer to the particle then computes the light fall off. This fall off in the brightness can be computed in several ways. Using the **fade** command, you can set this fall off to the following options:

planar is a $1/r^2$ light fall off with the distance measured from the view plane.

spherical is also a $1/r^2$ light fall off but with r measured as the true distance from the viewpoint (center of your screen). This is the default.

linear *refdist* provides a $1/r$ light fall off that, while not physically relevant, can be useful for sense of depth and entertaining questions like: What would the world be like if the light fall off law was $1/r$ instead of $1/r^2$?

const *refdist* sets a constant distance for all particles that is independent of the actual distance.

To see the difference between these, the **fade** command can be issued as

```
fade planar
fade const 10
fade linear 2.5
fade spherical
```

Typically, this command is not needed since the default gives the correct inverse square law for light.

The "Final" Luminosity Each of these commands discussed above are considered in the final luminosity calculation to determine the size of the particle. The formula may be expressed:

$$\text{Apparent Brightness} = \frac{\textbf{psize} \times \textbf{slum} \times \text{Sampling Factor} \times \textbf{lum}}{(\text{Particle} - \text{to} - \text{Camera Distance})^2}$$

where the **slum** value is that of the current **lum** variable, the **lum** value is that of the data variable linearly mapped, and the particle-to-camera distance is affected by the **fade** command.

How do you set these values for your data? Consider everything we've discussed in these past few sections that directly or indirectly affect the particle size and brightness: **ptsize**, **polysize**, **polyminpixels**, **alpha**, **polylumvar**, **fade**, **lum const**, **lum** *datavar*, **psize**, **slum**, **every**—am I missing any? With all of these settings and options, what is the best way to set the brightness of your data?

Let's eliminate a few of these options right away. If we're not using the **every** command, then the Sampling Factor will be 1 and our overall luminosity will not be determined by these factors.

The **ptsize** range is an intrinsic property of the points and should be set to your aesthetic. Typically one wants point sizes that fade to small sizes as the distance increases and have a maximum size that's not too large. Typical min-max values are 0.05 to 5.

The **fade** is correct to $1/r^2$ by default, so we need not worry about the denominator.

Considering points, not polygons, we only need to adjust the **lum**, **psize**, and **slum** values. These three values balance one another to produce the final particle size. Let's assume that the **lum** value is either set to a constant 1 or set to the data-specific luminosity value. Then, the final two parameters to set are the **psize** and **slum** values.

Given the difference between **psize** and **slum**, the former being a global scale factor while the latter is a data variable-specific scale factor, we should first set the **psize** value so that all particles appear more or less as we would like to see them. Next, we should use the **slum** factor to fine-tune the brightness, particularly for polygons. Once polygons are added to the equation, some of your values might have to be adjusted. However, **psize** does not affect the size of the polygons, so **slum** and **polysize** should be used together to get the right polygon size and brightness.

These are our general guidelines for setting the brightness of data. Typically, if you'd like to see your data from afar, you will want a high value for **psize**. Additionally, you can set a minimum size for polygons using **polyminpixels** which will alter the apparent brightness of the particles at larger distances.

5.5 Coloring Particles and Objects

There are three fundamental ways in which data and objects are colored in Partiview. The simplest method is to give your data a constant color using the **color const** command. Another, more complicated method involves giving each particle its own color via a data variable using the **color** *datavar* command, which then refers to a column of data in the data file or a separate color map file. Finally, one can color an object (not particle data) using the **cment** command.

To begin our exploration of color, let's load the Complex Data and give the points a high luminosity with this command:

```
lum const 1000
```

As you can see, the particles are multi-colored.

Setting a Constant Color Let's replicate what we saw in the Test Data and make the particles all one color. This is accomplished using the **color const** command:

```
color const 0.5 0.5 0.9
```

for a nice gray-blue color. Partiview reports that you are now `coloring-by rgb 0.5 0.5 0.9`, meaning it recognizes that we are setting a constant color to all particles.

Setting a Color Variable A color variable for each particle can be set within a `.speck` file. If you inspect the `complex.speck` file (or issue the **datavar** command to get a report on the data variables defined in that file), you will notice that data variable 1 holds the coloring information for each point. Using the **color** *datavar* command, change from a constant color to assigning colors from that column via

```
color coloridx
```

Since `coloridx` refers to **datavar** 1, you may also refer to this number rather than the name in an equivalent command `color 1`. However, we recommend learning data variable names since data columns can change. With this command, Partiview reports: `coloring-by 1(coloridx) exactly (cindex=data+0; data 0..6, cmap 0..7)`. This means that Partiview is coloring the data exactly according to data variable 1, called `coloridx`. Exact coloring means that the color index for a point will exactly match the color indexes in the color map file.

If you inspect the `coloridx` data, you will notice the correlation: a value of zero for the center point, a value of 1 for the $x = 1$ plane, and a value of 2 for the $x = -1$ plane. The coloring is consistent, but where are the actual colors coming from? To answer this question, we need to understand color maps.

Creating a Color Map For the Complex Data, we have created a simple color map file called `complex.cmap` where we define a group of colors. A color map file takes the form `R G B alpha` with the number of colors in the file appearing in the first line. The `complex.cmap` file looks like this:

```
8
0.5 0.5 0.5 0.3 # 0.gray
1.0 0.0 0.0 1.0 # 1.red
0.0 1.0 0.0 1.0 # 2.green
0.0 0.0 1.0 1.0 # 3.blue
1.0 1.0 0.0 1.0 # 4.yellow
1.0 0.5 0.0 0.6 # 5.orange
0.0 1.0 1.0 1.0 # 6.aqua
1.0 1.0 1.0 0.3 # 7.white
```

We see that the file begins with the number 8, then lists eight colors. Comments are also allowed in the file (anything after the #). Particles can be colored as an exact match to the color entries, or distributed over a range using the entries as a guide.

Using a Color Map A color map file is read in using the **cmap** command. In the Complex Data, the center particle (`coloridx` = 0) is gray, the x = 1 plane (`coloridx` = 1) is red, and so on.

There are two ways to use a color map file. One method involves using the **color** *datavar* **exact** command to use the exact colors in the color map file, while the other method distributes the coloring over a continuous range that is based on the color map file.

The Complex Data is loaded using exact coloring which is set in the `complex.cf` file. If you want to color the particles over a continuous range, type the command:

```
color coloridx -exact
```

The **-exact** part of this command removes the exact coloring and colors the particles over a continuous range. In this case, the range has been unspecified and defaults to the range of the `coloridx` data variable. You can specify that the colors be distributed over half the range using this command:

```
color coloridx 0 3
```

Summing up the commands:

color *datavar* ⟨**exact** | **-exact**⟩ [*baseval*] Color particles by mapping one-to-one the color *datavar* to the color index in the color map file. For example, data with a `coloridx` of 2, will be colored green (using the `complex.cmap` file), data with a `coloridx` of 7 will be white, and so on. Also, out of range values are assigned to the first or last entry in the file. For example, a `coloridx` of 30 will be colored white.

If you use negative values, the colors will be re-scaled according to where the negative value appears in your data. Any data prior to this point will be colored gray, the first entry in the color map file. We recommend using positive values for your color variable for more predictable results.

If you wish to start at some base color other than the first color in the file, then you may use the *baseval* option. For example, try setting the *baseval* to 2, and see what happens to the data. Now, data of `coloridx` = 0 will be mapped to the third color in the file. The rest of the colors follow from there, adding on one from the base. The values of `coloridx` that are attempting to reference colors that are not in the color map file will be given the last color and treated as out of range, so they will be colored white in our example color map file. If you were to start at *baseval* = 7, then all the points would be colored white.

The **exact** and **-exact** toggle exact coloring on and off, respectively. Exact coloring correlates the colors set by the *datavar* from the data file and the contents of the color map, given there's no *baseval* set. With exact coloring off, colors are now distributed over a continuous range (just as **color** *datavar* [*minval maxval*] is). Type the command

```
color coloridx -exact
```

to turn off exact coloring. Note that using the **color** *datavar* [*minval maxval*] command after using **exact** coloring will not remove the exact coloring, you must use the **color** *datavar* **-exact** command.

color *datavar* [*minval maxval*] Color particles using *datavar*. By default (without the *min max* arguments), the colors are mapped to colors between the second color and the $n - 1$ color in the color map, in this case red to aqua. The first and last colors (gray and white in this file) are used for out of range values. If we supply the *minval maxval* arguments, the colors are mapped over this range. For example, if we issue the command

```
color coloridx 1 7
```

we replicate the coloring of the **exact** command (if you issued the **-exact** command prior). Note that the center point is gray not because we are assigning it that color, but because it is out of the 1–7 range, and, therefore, receives the first (out-of-range) color. If we adjust the *maxval* range down to 6, you will notice that the last point $(x, y, z) = (0, 0, -1)$ is out of range, so it changes from aqua to white. Continue shrinking the range to 1–5. Now, you see a major shift in color. Now, all points in the yz plane (aside from the point on the origin) change color as the distribution is over a smaller range, making fewer colors available. Continue shrinking the range and see what happens.

Customizing a Color Index There are times when you want to set the color of one index value. The command to do this is **cment**. This command can be applied in many situations: coloring particles, objects, labels, and boxes. For example, let's say we want to change the $x = -1$ plane from green to purple. First, set the coloring of the particles back to exact using the command

```
color coloridx exact
```

Since we can see from the data file that the $x = -1$ plane points have the `coloridx` value of 2, let's just assign that color index a different color via the command

```
cment 2 1 0 1
```

This sets the color index 2 to the color purple $(R, G, B) = (1, 0, 1)$.

We will also use this when defining objects like ellipsoids, meshes, and other objects. Let's draw a sphere and assign it the color green. Define a new data group called `sphere`, then draw a green sphere using the commands:

```
g2=sphere
add 0 0 0 ellipsoid -r 100 -c 20 -s wire -n 36,19
cment 20 0 .7 0
```

You may notice that if you try to define a color index of 20 in group 1, Partiview would have told you the color index was out of range. This is due to the fact that the data group is using the `complex.cmap` color map that has color indices ranging from 0–7. There are similar commands for adding color to labels and boxes, which we discuss next.

Coloring Text There are two commands needed to color text in Partiview. The color is set using the **textcment** command and assigned to that color index using the **textcolor** command. We define a text color index in the config file `complex.cf` using the command

```
textcment 1 .2 .4 .6
```

We then issue the command

```
textcolor 1
```

in the data file, `complex.speck`, where the labeling data is located. This command tells Partiview that the labels should be colored according to color index 1. While the **textcolor** command cannot be entered at the Command Line, **textcment** can be altered interactively. Knowing that the color index of the text color is 1, you can alter the color by reassigning the color from the bluish color to an orange color (provided you have the correct group selected) using the command

```
textcment 1 .6 .4 .2
```

To change the color index number, you must change the argument of the **textcolor** command in the data file, then re-load the data file.

Coloring Boxes Let's create a box around our sphere that's centered on the point $(x, y, z) = (0, 0, 0)$ and with half-lengths of 100:

```
add box -n 1 -l 4 0,0,0 100,100,100
```

A blue box appears; the `-l 4` argument sets the color index to 4 which is blue in Partiview's default color map. We can change this color via the **boxcment** command. Similar to the **cment** command, we can change the color to red by issuing the command

```
boxcment 4 .7 .1 .1
```

We can also color the box according to a color map file by reading that file using the **boxcmap** command. Change the box color to read from the file using this command:

```
boxcmap complex.cmap
```

Now the box color, still set to a color index of 4, is yellow, according to the 4th color of the color map file.

Finally, the box axes can be highlighted to show the fundamental red, green, and blue axes by using the **boxaxes** command.

```
boxaxes on
```

adds a red, green, and blue fade to the x, y, and z axes, respectively. To change back, just toggle the **boxaxes** off.

5.6 Label Properties

Including labels in Partiview is quite simple. To add a label anywhere in space, include a line in a data file (or issue it interactively at the Command Line with the **add** preface) that has this format:

```
x y z text your label
```

The **text** command will place the label `your label` at the point (x, y, z). These lines can either be included in a `.speck` file or in their own file with the `.label` extension. By default, each label is displayed with its label axes, a local red, green, blue axes, aligned to the world coordinates. These can be turned off using the **laxes off** command. The labels themselves can be toggled on and off using the Label Toggle Button or the **labels** command.

Setting the text color If you wish to change the text color from the default white, you may use the **textcment** and **textcolor** commands. **textcment** allows you to assign a color to a color index, such as

```
textcment 14 0 1 0
```

where we assign the color green to color index 14. Once we've set a color index, we then use the **textcolor** command to assign the color to a color index, like

```
textcolor 14
```

Using these two commands allows us to set the text color for a given data group. Currently, there is no way to introduce a column that represents the text color. If you want several text colors for the labels

within one data group, you will have to define multiple color indices with **textcment** and set the color with **textcolor** then list those labels you want with that color. Following that will come the next block of **textcolor**, data lines, etc.

Setting the Size The size of your labels can be set using the **labelsize**, or **lsize**, command. The value of **lsize** is relative to the world coordinates. If you set **lsize** to be 1, then the entire label (including the height of capital letters and those that dip below the rule, like 'y' or 'g') will be one unit high. For example, start up Partiview without any data, then add a label at the origin by issuing this command at the Command Line:

```
add 0 0 0 text My Label!
```

Now set the center size and the label size to be equal

```
censize 1
lsize 1
```

and fly away to see the entire label. You can see that the height from the bottom of the 'y' to the top of the 'M' is about 1 unit.

Turn off the label axes using `laxes off`.

By default, the size of a label depends on its distance from you. Nearby labels will appear larger than those far away. If you wish to alter this relationship between label size and distance, then you may specify a value in the **text** command. For any label data, instead of `0 0 0 text (0, 0, 0)`, you can specify that this point's label be twice the size of all other labels in the data group using this command:

```
0 0 0 text -size 2 (0, 0, 0)
```

Setting the minimum label size Using the **labelminpixels** command, you can set the minimum size in pixels above which a label will be displayed. This is often useful to reduce label clutter in the display. For example, setting a **labelmin** of 8 means you are only drawing labels of 8 or more pixels in height. The **labelmin** value is, of course, dependent on the value of **lsize**. If **lsize** is high, then **labelmin** will need to be higher if you want to draw only the larger labels.

6

Drawing Objects

In addition to displaying particles, Partiview can also draw objects that add context to your data. These include boxes, spheres, ellipsoids, and meshes. For each of these objects, there exist Data Commands and Control Commands. For example, the Data Command **box** sets the dimensions and displays a box. The Control Command then toggles the box on and off. This is true for all the objects we will describe in this section.

6.1 Boxes

There are two types of boxes in Partiview: a normal box and a clip box. A normal box is one that simply exists in space without affecting the data group it belongs to. A clip box, on the other hand, defines a box outside of which data are not drawn, thereby highlighting a portion of data within a given data group. In this section, we will discuss the normal box, leaving the discussion of clip boxes to
"Creating Data Subsets."

Drawing a box Let's start by drawing a box around the data cube in the Test Data set. The data extend from -1 to $+1$ in all directions and, rather than draw the box *on* the points, let's draw one just outside these points using the **boxes** command. **boxes** takes a few arguments, namely the coordinates, the box color, and the box number. The coordinates can be entered in one of two ways. Either using the center point along with the x, y, and z half-lengths, or using the minimum and maximum values for x, y,

and z. If we were to draw a box using the center point and half-lengths that encompassed the data cube, we would use

```
add box -n 1 -l 1 0,0,0 1.2,1.2,1.2
```

This produces a box that is numbered 1 (**-n** 1), colored blue (**-l** 1), and is located just outside the Test Data cube (at $x = y = z = 1.2$) centered on the point $x = y = z = 0$. An equivalent command using the minimum and maximum pairs would be

```
add box -n 1 -l 1 -1.2,1.2 -1.2,1.2 -1.2,1.2
```

Let's set the Point of Interest marker to zero (`censize 0`) so that it's not in the way.

Turning Boxes on and off Because the above command is a Data Command, we must preface it with the **add** command when we want to enter it in the Command Line. The **boxes** Control Command toggles a box on and off. Typing

```
box
```

will toggle the box between the **on** state, the **only** state, and the **off** state. The **on** and **off** state are self-explanatory. The **only** state turns the points off, leaving the box on. Note that if the polygons are on, then the they will remain displayed while the points are turned off. These states can also be toggled using the Boxes Toggle Button. Left-clicking on the button will toggle between the **on** and **off** state. Right-clicking on the button will provide you with a menu to choose from one of these states. The button will appear green in the **on** state and red in the **only** state.

Displaying the Box Label While its usefulness may be debated, the box number can be shown as a label at the center of the box. By turning on **boxlabel**, the label (the number 1 since we specified a **-n** 1 in the box definition) will appear in the center of our box. More practical is a user-customized label that is relevant to the purpose of the box. For example, we may want to call these data the "Center Data," in which case we can add a label manually (or in a data or config file). To do this interactively, type the following in the Command Line:

```
add 1.25 1.25 1.25 text Center Data
```

You will need to have the labels on for the label to appear.

Creating Multiple Boxes Of course, you may create a number of boxes within one data group and multiple boxes in multiple groups. When you create a box, always be mindful of the active data group as the box will belong to that group. One of the powers of multiple boxes within one data group is that boxes can be grouped by their color. To experiment with this, let's start Partiview with the Sample Data. This provides us with a larger data set to work with. Increase the center size to 100 so that we can see our coordinate axes.

Let's create several boxes of various colors by typing the following commands.

```
add box -n 1 -l 10 10,10,0 10,10,10
add box -n 2 -l 15 -10,0,-10 10,5,5
add box -n 3 -l 10 -50,20,10 40,10,5
add box -n 4 -l 10 -50,-25,0 20,10,60
add box -n 5 -l 25 -20,-40,10 2.5,35,2.5
```

Clarify the view by turning off the points and polygons, reducing the Point of Interest axes to 10, and turning on the box labels. This is most easily accomplished using these commands:

```
points off
poly off
censize 10
boxlabel
```

Showing and Hiding Boxes We can toggle a subset of boxes on and off by using their color index set with the **-l** option. Using the **showbox** and **hidebox** commands, we will turn off all boxes using the command

```
hidebox 10 15 25
```

Now, turn on the three boxes of the same color index

```
showbox 10
```

Or, turn off these boxes and show the other two

```
hidebox 10
showbox 15 25
```

This is how you can use several boxes to highlight portions of data that are related to one another.

Coloring a Box Like coloring particles, coloring a box can take place via a color index which refers to a color map file, or the color index can be set interactively for that data group using the **boxcment** command. If we desire to change the color of those boxes that have the color index 10 to orange, then we would type

```
boxcment 10 0.7 0.4 0
```

(use the `showbox 10` command if they are currently hidden). We can also color boxes using defined colors in a color map file, see the discussion in "Coloring Particles and Objects."

Scaling Boxes All boxes defined in a data group can be scaled using the **boxscale** command. This command takes a floating point value to describe the scale factor by which the size of the box will increase or decrease. For example, to double the size of all boxes, then return to the defined size, then quarter the size, issue these commands and see what happens.

```
boxscale 2
boxscale off
boxscale 0.25
boxscale 1
```

Note that `boxscale off` and `boxscale 1` are the same. Also, the labels do not scale, you may want to turn them off using `boxlabels off`.

Changing the Point of Interest to Box Center The Point of Interest can easily be shifted to the center of a box using the **gobox** command. This command takes the box number that you want to bring into focus by moving the Point of Interest to its center. If we bring the labels back, then we can shift the Point of Interest around via the commands:

```
gobox 4
```

and so on for the other boxes defined with the **-n** *boxnumber* option and argument.

6.2 Spheres and Ellipsoids

Spheres and, more generally, ellipsoids, can be used in Partiview to represent structural elements, coordinate systems, and even planes and lines (we'll explain). The **ellipsoid** data command is

somewhat complicated for two reasons: one is the fact that the parameters to define an ellipsoid are complex, another is that there exists both a Data and Control version of the command. However, we expect that by the end of this section, you will have a good working knowledge of the **ellipsoid** command.

Control Command: State Change The **ellipsoid** Control Command is fairly self-explanatory and has two arguments:

> **ellipsoid** [**on** | **off**]

If an ellipsoid has been defined for a particular data group, using **ellipsoid on** or **ellipsoid off** will turn the object on or off (assuming the data group for which it is defined is the active data group). As for defining the parameters of an ellipsoid, you'll use the Data Command for that.

Data Command: Drawing an Ellipsoid The **ellipsoid** Data Command defines the parameters of the ellipsoid and takes the form:

> **[add]** *xcen ycen zcen* **ellipsoid** [**-r** *xrad*[*,ycen,zcen*]] [**-c** *colorindex*]
> [**-s** ⟨**solid** | **plane** | **wire** | **point**⟩] [**-n** *numlong*[*,numlat*]] [*transformation*]

An ellipsoid is defined by its center point and its radii in the x, y, z directions. In a data file, the control command would be prefaced by the **eval** command and this data command would appear without the **add** preface. Start up the Test Data and define a new data group

```
g2=ellipsoid
```

Now, let's draw a simple ellipsoid centered on $(0, 0, 0)$ with radii of (*xrad*, *yrad*, *zrad*) = $(5, 10, 15)$.

```
add 0 0 0 ellipsoid -r 5,10,15
```

This will cause the screen to go gray. If you pull away from the data, you will see that a white, solid ellipsoid has been drawn (increase the center size to see the proper perspective). This is not really what we want though. The default style is **solid**, but we want to change that to **wire** so that we can see the entire ellipsoid. Also, let's change a few other attributes.

Define another group and let's set ellipsoid *colorindex* to 5, its style to **wire**, and the number of latitude and longitude lines to 30. The command looks like:

```
add 0 0 0 ellipsoid -r 5,10,15 -c 5 -s wire -n 30
```

This results in a somewhat odd looking object, but we now see the wire-frame ellipsoid.

The style argument can either be **solid**, **plane**, **wire**, or **point**. Define another group and issue the above **ellipsoid** command trying these styles. (Just define a new group in the Command Line, then hit the up arrow key to scroll up to the ellipsoid command, where you can use the arrow keys to edit the line.)

If you try the other styles, you will find that **plane** draws three ellipses in the $x = 0$, $y = 0$, and $z = 0$ planes. The **point** style draws points at each vertex. These can often be difficult to see though. You should change the **alpha** value to 1 and choose a color that will stand out, like yellow. If you've already defined your ellipsoid using the *colorindex* of 5, simply change the color for that *colorindex* using the **cment** command—`cment 5 1 1 0` will change it to yellow. It also helps to have a lot of vertices, so *numlat* and *numlong* should be large.

Drawing a Sphere To draw a sphere, you simply want the three radii for the ellipsoid to be equal. If you supply only one radius, the other two will be set equal to it. For example

```
add 0 0 0 ellipsoid -r 10 -c 5 -s wire -n 30
```

will draw a sphere of radius 10. If we want to use a sphere for mapping coordinates, then it's useful to draw lines of latitude and longitude at intervals that make sense. To draw lines at 10° intervals, the number of longitude lines would be 36, while the number of latitude lines would be 19. You probably were expecting 36 for a 360°, but where does 19 come from? For the lines of latitude, you must take into account the two poles. If you want lines every 15 degrees, then use the option **-n** `24,13`.

Drawing a 2-D plane Using the **ellipsoid** command, you can actually draw a circular, two-dimensional plane. This can be done by setting one of the radii to zero. For example, re-start Partiview with the Test Data. Now, let's draw an ellipse in the *xy* plane out to 1 unit of distance. Turn off the labels, then run this command

```
add 0 0 0 ellipsoid -r 1,1,0 -c 16 -s wire -n 36
```

You may want to adjust the **alpha** level so that the plane is more subtle. Unfortunately, the grid lines are not as useful as they would be if they were evenly spaced. However, this can be useful for demonstration purposes.

Drawing a Line Finally, if we set two radii to zero, we get a line. Turn off the plane by typing

```
ellipsoid off
```

in the Command Line. Now, let's draw lines from point to point. First, try to draw a line from $(x, y, z) = (1, 1, 1)$ to $(1, 1, 0)$. This would put the center at $(x, y, z) = (1, 1, 0.5)$ and the *zrad* at 0.5 (halfway between 0 and 1). Let's try it.

```
add 1 1 0.5 ellipsoid -r 0,0,0.5 -c 10 -s wire -n 30
```

The unfortunate thing is that lines cannot be drawn off plane. They can, however, be drawn then transformed using the **tfm** command. However, this is not a viable solution for line drawing in Partiview. It makes more sense to draw lines using the **mesh** command which we discuss below.

6.3 Meshes

Meshes are used to draw lines from one point to another. A mesh offers the flexibility to draw virtually any shape within Partiview, from a line between two points to complex, three-dimensional grids and surfaces.

The **mesh** command is typically used in a file and takes the form:

```
mesh -t texnum -c colorindex -s style {
numu numv
x1 y1 z1 u1 v1
x2 y2 z2 u2 v2
...
xN yN zN uN vN
}
```

where the **mesh** command is issued with the **-t**, **-c**, and **-s** options (see below). After the mesh options is an open curly brace followed by *numu numv*, which specifies the dimensions of the mesh.

If you wish to draw a line between points, then *numu* will be 1 while *numv* will equal the number of points to connect. If you want a square, 4000 $\times$ 4000 grid with lines every 200 units, then *numu numv* will both equal 21. The data points (x, y, z) as well as the optional u and v coordinates for texture mapping are listed below these dimensions. The **mesh** command is concluded with a closing curly brace.

The **-c** option sets a *colorindex* that points to a color set using the **cment** command. The **-t** option loads a texture whose *texnum* is set using the **texture** command. Finally, the **-s** option sets the style of the mesh. The style can take the following values:

solid draws a filled surface,

wire draws a wire-frame connecting each of the vertices,

point draws a point at each vertex, leaving them unconnected.

For example, start Partiview with the Test Data and load the file `mesh.speck` via the command

```
read mesh.speck
```

You will see a few meshes appear and upon inspection of the `mesh.speck` file, you will notice that there are three mesh commands. The first connects seven points into the pyramid whose apex is at $y = 0$. The second is a square grid in the xz plane, and the third draws a solid mesh in the $y = -1$ plane upon which the blue ring texture is placed. Use this file as an example for other meshes you'd like to create.

7

Creating Data Subsets

This section describes commands and functions in Partiview that allow you to display a subset of particles within one data group. These subsets can be random, specified over a range, or display data from a specified data variable. Once you have subsets you'd like to keep, they can be saved in selection expressions, then easily toggled on and off.

7.1 Displaying a Random Subset

You can display a random subset of particles using the **every** command. **every** takes an integer argument that signifies the number of data points to sample. For example, using the Sample Data, fly out and increase the **slum** value so that you can see most of the data points in your view (that is, you are essentially outside the data set, but you can still see most of the particles). If you issue the **bound** command, you should see that the Sample Data has 10,420 specks, or particles. The default is to display every one of these particles, which corresponds to the command `every 1`. To display every second particle you would type

```
every 2
```

and so on. As you type in higher numbers, you will notice the number of particles decreasing. **every** reports the current status back to you, for example, for `every 2`, the report is

```
display every 2th particle (of 10420)
```

which is written to the Console Window upon issuing the command, or by issuing the **every** command without an argument.

As you type in higher and higher numbers, the data will retain its shape, but will be more sparse. If you enter `every 100`, you will still notice a spherical distribution, but with far fewer particles. You may also notice the particles increasing in brightness. This is because the overall apparent brightness remains the same, thus the remaining particles must brighten to compensate.

7.2 Clip Boxes

A clip box differs from a normal box in that all particles outside the defined box are not drawn—the data are clipped. Clip boxes are defined in a similar fashion as boxes are in Partiview. The command **clipbox**, or its equivalent shortened version **cb**, are Control Commands that define a clip box. To explore how clip boxes work, load Partiview with the Sample Data.

Defining a Clip Box We can specify the dimensions of a clip box in the same way we specify the size of a box. Use either the center point along with half-lengths in the x, y, and z direction, or specify the minimum and maximum in each of these coordinates. For example, using the Sample Data, we can see what the extent of the data are in each direction using the **bound** command. Issuing this command will report the information about the data in the format:

```
num specks in range xmin ymin zmin ..  xmax ymax zmax (object)
midbox xcen ycen zcen boxradius xlen ylen zlen (object)
mean xavg yavg zavg (object)
```

This gives you all the information you need to draw a box. Let's draw a clip box centered on the point $(x, y, z) = (0, 0, 0)$ and half-lengths (or radii) of 50 in the x and y directions and 100 in the z direction. This can be achieved using either of these commands:

```
cb 0,0,0 50,50,100
cb -50,50 -50,50 -100,100
```

Note the difference in our use of commas in these two commands. The placement of commas and spaces (or lack of) are important for these commands.

Turning a Clip Box On and Off To turn the clip box on or off, use the **cb on** or **cb off** commands. If you would like to hide the clip box while retaining the data clipping, you may use the **cb hide** command to remove the box. However, if you have polygons on, they will remain visible in **hide** mode—you may turn the polygons off if you wish. **cb off** will bring all the data back into view.

7.3 Thresholding Data

If you think of a clip box as defining data thresholds in *x*, *y*, and *z*, the **thresh** command allows you to threshold particles according to data variables you define via the **datavar** command. The **thresh** command takes several forms, so let's explore these one by one using the Sample Data.

Reporting the Data Variables Let's fly out so that we can see most of the Sample particles in our view. You will likely have to increase the brightness (**slum** value) of the particles using the Slum Slider. To get an idea of what data variables are available for threshing, issue the **datavar** command without any arguments (you may want to stretch the Console Window in Partiview by placing your mouse at the base of the Command Line and pulling down to expand the size of the Window). **datavar** will report each data variable that is defined in the `.speck` file in the format:

`datavar` *num name min.. max* `mean` *meanvalue* `[default]`

You can see that the zeroth data variable, `coloridx`, ranges from 1 to 10 with a mean of 5.5 and Partiview sets the zeroth data variable to be the color.

Thresholding on color If we want to threshold on the color, we should familiarize ourselves with the Sample color map file `sample.cmap`. You will see that color 0 is red, 1 is green, 4 is orange—you can check the file. Since our color indices range from 1 to 10, there will be no red particles. Let's threshold these data by color. If you want to see just the particles with a color index of 1, then we would type

```
thresh coloridx 1 1
```

which displays all the green particles. To see both green and blue: `thresh coloridx 1 2`. We can also refer to the data variable number, 0, in the **thresh** command as well. Let's see this by viewing the aqua particles, then the purple particles in these commands

```
thresh coloridx 5 5
thresh 0 6 6
```

We do not recommend using the data variable number when referring to a variable since columns within a data file can shift between data processing iterations.

Upon issuing the **thresh** command, a report is written to the console in the form:

`thresh` *varnum*`(`*name*`)` `min` *minval* `max` *maxval* `(`*num* `of` *total* `selected)`

where *varnum* is the data variable number that corresponds with the *name*, *minval* and *maxval* are the minimum and maximum values of the thresh, and *num* is the number of particles displayed out of the *total*. To return seeing all the particles, use the **see** command

```
see all
```

which will return all particles to view.

Threshing data according to some value Instead of providing a minimum and maximum range over which to threshold data, it is also possible to threshold data above or below a given value. For example, let's threshold these particles according to the value of its label (data variable `label`). We can see from **datavar** that the values range from 0 to 120,250. Let's look at all the particles that have a `label` less than 10,000. We can see these particles using the command

```
thresh label < 10000
```

Now, increase this value to 40,000 so that you're only seeing particles below `label` = 40000. Now let's reverse it using the $>$ sign so that we're seeing all particles above `label` = 40000 using

```
thresh label > 40000
```

In this way, we can thresh over a range given one value from the data.

Seeing all your data To return all the particles in the group to view, you can either use the **thresh** command without specifying *min* and *max* values, you can replace the *min* and *max* values with dashes, you can use the **thresh** *datavar* **off** command, or you can use the **see** **all** command. Each of these would be executed in the following way:

```
thresh label
thresh label - -
thresh label off
see all
```

Reversing the Thresh You can reverse the thresh results by using the command **see -thresh** as well. For example, if you thresh on `label` to see all data with `label` $<$ 50000, you can reverse the thresh, that is, see all data with `label` $>$ 50000 by issuing

```
see -thresh
```

We will talk more about the **see** command and setting up selection expressions with **thresh** later in this chapter.

7.4 Selecting Data with Specific Values

The **only** command allows you to select a subset of a data group based on particular data attributes. When we do a `thresh datavar 2 2`, we are choosing only those particles with the values of the data variable `datavar` equal to 2. There is a more sensible way to do this using the **only** commands.

Choosing particular values to display If you wish to pick out portions of a data group according to particular values of a data attribute, the **only=** command is what you want to use. The command has the form:

only= *datavar* ⟨ *value* | *minval maxval* | $<$*maxval* | $>$*minval* ⟩

Let's try using this command by selecting only the particles in the Sample Data with a `coloridx` of 6 via the command:

```
only= coloridx 6
```

Now, we can select a range of values like the colors 2,3,4,5 and 8 using this command:

```
only= coloridx 2-5 8
```

Also, you can select data using the greater-than and less-than signs, as in:

```
only= coloridx < 7
only= coloridx > 7
only= coloridx < 2 > 8
```

although this last statement seems better suited for the **thresh** command. With the last command, we are viewing only those particles of `coloridx` 1,2,8,9, and 10. Now, let's alter the selection of just these data.

Choosing a subset from a subset Once an **only** condition has been defined, data can be further added or subtracted from the current subset by using the **only+** and **only-** commands. These have the same arguments as the **only=** command above, but they add or subtract particles from the current data subsample.

Let's thresh on the luminosity of the particles. Out of the subset of data we have now, let's remove those particles that have a luminosity less than 50. Typing **datavar** will show you that the `lumin` data variable ranges between 10 and 100 and has an average of about 50. So, we are asking for the brighter particles from those that have a `coloridx` of 1,2,8,9, or 10. Executing

```
only- lumin < 50
```

removes the dimmer points, further shrinking the data subset that is displayed. Now, let's add some data in. Say we want to see the dimmest points along with the brighter ones. Let's add some data using the **only+** command:

```
only+ lumin < 20
```

Now, the number of points increases as expected. These three commands allow for complete customization of data display.

7.5 Selection Expressions

The **sel** command defines a selection expression that allows you to save a data threshold, then re-issue the named selection expression later with the **see** command.

To demonstrate this command, let's start Partiview with the Sample Data set. If we issue the `datavar` command, we can see that we have a color, luminosity, and label variable. Let's threshold the data by color via the command

```
only= color 4
```

This displays the orange particles, of which there are just over 1,000. Now, let's save this setting to a selection expression called "orange" using the command

```
sel orange = thresh
```

Refine the displayed data by removing dimmer data using the command

```
only- lumin < 80
```

and call this "brightorange" in the command

```
sel brightorange = thresh
```

Now, we can toggle between these modes using the **see** command. The **see** command takes one argument, the name of a selection expression. There are several pre-defined selection expressions: **all**, **none**, and **thresh**. Let's return all the Sample Data to view using the command

```
see all
```

Now we can toggle our saved expressions in these commands

```
see orange
```

displaying only those particles with `color` = 4. Now, we can see only the bright particles using the other expression we defined earlier

```
see brightorange
```

If we have only one threshold for a data set, we can use `see thresh` to toggle between seeing all the data and only the threshed data.

Selection expressions are best defined in config or data files to pre-load different views of your data set.

8

Statistical Reports

Partiview has a few utilities that report general statistical information about the data displayed for a particular group. We describe these in this section.

8.1 The Spatial Extent of the Data

The spatial characteristics of a data group are printed to the Console Window when the **bound** command is used. If we open the Complex Data and type the command

```
bound
```

Partiview will report several lines of information about the data group. The format of the report is:

```
num specks in range xmin ymin zmin ..  xmax ymax zmax (object)
midbbox xcen ycen zcen boxradius xrad yrad zrad (object)
mean xavg yavg zavg (object)
```

where *num* is the number of data points, *xmin, xmax* are the minimum and maximum points in the *x* direction, *xcen ycen zcen* is the center of the data distribution, *xrad yrad zrad* are the half-lengths of the data distribution, and *xavg yavg zavg* are the averages for each coordinate.

Appended to the end of each line of the report is the coordinate system to which these numbers are relevant. Without an argument, **bound** reports the 3-D extent of the data in object coordinates, the

coordinate system defined by the data itself. So, for the Complex Data, we see that the particles range from -1 to +1 in each coordinate with the center and mean at $(0, 0, 0)$. If the command

```
bound w
```

is run, then the report is in terms of the world coordinate system. This produces the same report in the Complex Data, however, if we transform these data using the **tfm** command

```
tfm 4 4 0 0 0 0
```

then rerun `bound`, you will see that the report reflects the original data values. Running the `bound w` command accounts for the newly transformed data values.

8.2 Generating a Histogram of the Data

A histogram of a data variable within a data group can be generated with the **hist** command. Using the Sample Data, let's explore how this command works. **hist** takes the form

hist [**-n** *numbins*] [**-l**] [**-c**] [**-t**] *datavar* [*minval*] [*maxval*]

where

-n *numbins* sets the number of bins for the histogram (default = 11),

-l sets the intervals to be logarithmically-spaced,

-c counts only the particles displayed within a clipbox,

-t counts only particles in a subset defined by **thresh** or **only=**,

datavar is the data variable to return the distribution for,

minval is an optional minimum value for the distribution range, and

maxval is an optional maximum value for the distribution range.

To see how this can be useful, let's generate some histograms of the Sample Data on a data variable we're familiar with, the color index `coloridx`. Recall that these values range from 1 to 10 and are fairly evenly distributed since the mean value is around 5. Let's see how many are in each color with

```
hist -n 10 coloridx
```

This should report:

```
hist -n 10 0(coloridx) 1 10 =>
Total 10420, 0 < min, 0 > max, 0 undefined, 0 clipped, 0 threshed
0 < 1
1007 >= 1
1026 >= 2
1070 >= 3
1060 >= 4
1059 >= 5
1024 >= 6
1071 >= 7
1009 >= 8
1083 >= 9
1011 >= 10
0 > 10
```

which makes sense with an even distribution between 1 and 10. Now, we could reduce the distribution of the histogram in half by specifying only 5 bins

```
hist -n 5 coloridx
```

which now creates bins that are wider, that is, provide less resolution on the data. We can also specify logarithmically-spaced bins with the **-l** option, as in

```
hist -n 5 -l colorindex
```

If we thresh these data according to luminosity, as in

```
thresh lumin 50 100
```

we can now get a histogram on just the subset of data by including the **-t** option

```
hist -n 10 coloridx -t
```

Note that once data are threshed and the **-t** argument is used, the results from **hist** will reflect the current threshold, even if you re-enter the command without the **-t** option. To return a histogram of all the data, use the **see all** command, then enter the **hist** command.

If we have defined a clip box, we can use the **-c** option

```
hist -n 5 -c coloridx
```

to generate a histogram based on the remaining unclipped data, whether the clip box is on or off.

Providing a *minval* sets the minimum value of the data variable to begin the histogram while *maxval* sets the upper limit on the histogram. Try entering

```
hist -n 10 coloridx 2
```

This will report that there are 1007 points with `coloridx` less than two and 1026 particles with `coloridx` equal to two. If you run

```
hist -n 10 coloridx 3
```

you are setting the lower limit of the distribution so that now the histogram will report 1007 + 1026 , or 2033 particles less than 3, the base value. Similarly, a maximum value can be specified either with a minimum value specified or with a $-$ in place of the *minval*, causing Partiview to substitute it with the minimum from the data.

Appendix A

Keyboard Controls

Table A.1 – Partiview keyboard shortcuts, typed in the Viewing Window.

Key	Function
[Tab]	Changes the focus to the Command Line to type a command.
[ESC]	Exit Partiview (also see the **exit** command).
f	Change to Fly flight mode.
o	Change to Orbit flight mode.
r	Change to Rotate flight mode.
t	Change to Translate flight mode.
[Shift]	Allows finer control during flight.
[Control]	Modifies the Orbit Flight Mode without having to change modes. Left button flying changes from orbit to pan. Right button flying changes from forward/backward to rotate.
cw	Resets the camera position to $(x, y, z) = (0, 0, 3)$.
p	Identifies the nearest object under the mouse cursor. The result is printed to the Console Window.
[Shift]-p	Change the Point of Interest to the selected object. This will change the point about which rotations and orbiting take place.
s or S	Toggles the stereo viewing mode on and off (see the **stereo** command).
[Shift]-o	Toggle the Perspective Mode on and off.
v	Increase the Field of View by roughly 12° (see the **fov** command).
[Shift]-v	Decrease the Field of View by roughly 12°.

Appendix B

Partiview Commands

This section provides an alphabetical listing of the Partiview command set. We recommend reading Chapters 3–8 to explore each of these commands and concepts in greater detail.

Data Versus Control Commands Partiview has two distinct kinds of commands: Data Commands and Control Commands. Control Commands are meant to be issued within Partiview, while Data Commands are meant to be issued within a data file. However, in Partiview, any command can be issued interactively, in a config file, or in a data file by using the proper prefix command. These prefix commands are **add** for Data Commands and **eval** for Control Commands. In the table below, we outline when to use these prefixes.

Table B.1 – Issuing Partiview Commands. If the command requires a prefix, the prefix is typed before the command.

	When issuing a command in...	
Command Type	**The Command Line (interactively)**	**A Data File (pre-loaded)**
Data Command	Preface with **add**	no prefix
Control Command	no prefix	Preface with **eval**

Partiview Command List This is not a complete listing of Partiview commands; however, we include many of the commands that you will find useful. You may consult information on the Partiview web site for more information.

Bold face page numbers mark where a concept is defined and explained. Slanted page numbers denote where that command is controlled from the graphical user interface (GUI).

add

The **add** command is used before any Data Command run from the Partiview Command Line. All Data Commands in this listing have the **[add]** preface shown to distinguish Data Commands from Control Commands. **add** is not necessary if you are issuing the command inside a data file.

Example Pages: 28, 74, **90**

See Also: **eval**

alpha [**+, *,** /][*value*]

The **alpha** command describes the opacity (opaqueness) of an object. The *value* argument ranges between 0 and 1, 0 describing no opacity (completely transparent), 1 setting maximum opacity. The default is 0.5.

Example Pages: *22*, **55**

See Also: **texture** Data Command, **lum**

async *command*

Run a UNIX command as a subprocess of Partiview. The output of the subprocess is interpreted as a stream of control commands. This allows Partiview to be driven externally by another program, such as another GUI or a shell script with Partiview commands prefaced by the shell command `echo`. For example, if you make an executable file called `file.sh` with the commands

```
echo g1 on
echo g1 lsize *2
echo g2 slum *3
```

then these commands will be executed in sequence upon running the file via the command

```
async ./file.sh
```

issued at Partiview's Command Line. Note that once a subprocess is started, it cannot be interrupted unless Partiview itself is terminated.

NOTE: This command works only on UNIX-based operating systems (Linux and Macintosh) and does not run in Windows (unless you're running a UNIX emulator).

Example Pages: –

See Also: –

bgcolor [*grayscale* | *R G B*]

Reports the background color of the display in red, green, and blue colors. Including the *R G B* arguments will set the background color. For example, `bgcolor 1 1 0` will produce a yellow background, while `bgcolor 1 1 1` will give you a white background. We mostly use a black background, `bgcolor 0 0 0`. Specifying only one color sets a *grayscale* (*R* = *G* = *B*) for various levels of gray (0–1).

Example Pages: **37**

See Also: **color const**, **color**

bound [**w**]

Reports the number of data points and the 3-D extent for the active data group. Also reports the parameters needed for box dimensions. With the **w** argument, the report is in world coordinates, otherwise **bound** reports in object coordinates.

Example Pages: 79, **85**

See Also: **clipbox**, **boxes** Data Command

boxaxes [**on** | **off**]

Toggles the box axes display mode which enhances the colors of the box. The red, green, and blue axes that are seen in the Point of Interest are reflected in the corresponding axes on the box.

Example Pages: **67**

See Also: **boxes** Data Command, **boxes** Control Command, **boxcmap**, **boxcment**, **boxlabel**, **boxscale**, **clipbox**, **gobox**, **hidebox**, **showbox**

boxcmap *filename*

Selects a color map file for coloring a box or many boxes assigned to the active data group. Refer to this color index using the **-I** option in the **boxes** Data Command.

Example Pages: **67**

See Also: **boxes** Data Command, **boxes** Control Command, **boxaxes**, **boxcment**, **boxlabel**, **boxscale**, **clipbox**, **gobox**, **hidebox**, **showbox**

boxcment *colorindex* [*R G B*]

Assign a *R G B* color to a color index specified by *colorindex*. Refer to this color index using the **-I** option in the **boxes** Data Command.

Example Pages: **67**, 73

See Also: **boxes** Data Command, **boxes** Control Command, **boxaxes**, **boxcmap**, **boxlabel**, **boxscale**, **clipbox**, **gobox**, **hidebox**, **showbox**

box[es] [**on** | **off** | **only**]

If a box has been defined (see the **boxes** Data Command to define a box), the **box[es]** Control Command toggles the box display. The **on** | **off** arguments turn the box on and off. The **only** argument turns all the particles (points) off; however, if polygons are on, they will remain displayed.

Example Pages: *21*, **71**

See Also: **boxes** Data Command, **boxaxes**, **boxcmap**, **boxcment**, **boxlabel**, **boxscale**, **clipbox**, **gobox**, **hidebox**, **showbox**, **bound**

[add] box[es] [**-n** *boxnumber*] [**-l** *level*] *coordinates*

Use the **box[es]** Data Command to draw a box of number *boxnumber*, color index *level*, and size specified by the *coordinates* argument. The *coordinates* can be specified in two ways:

xmin,xmax ymin,ymax zmin,zmax draws a box using the minimum and maximum values for each coordinate. These values define the 6 planes that make up the box. In this format, the six values are written in coordinate pairs. (Note the use of commas to separate each coordinate's minimum and maximum value and the use of spaces to separate the three coordinates.)

xcen,ycen,zcen xrad,yrad,zrad draws a box centered at *xcen,ycen,zcen* and with half-lengths (or "radii") given by *xrad,yrad,zrad*. (Note the use of commas to delineate each center and radius value and a space to separate the center and radius groups.)

Boxes can be turned on or off using the **boxes** Control Command.

Example Pages: **70**

See Also: **boxes** Control Command, **boxaxes**, **boxcmap**, **boxcment**, **boxlabel**, **boxscale**, **clipbox**, **gobox**, **hidebox**, **showbox**, **bound**

boxlabel [**on** | **off**]

If a box was added using the **-n** option, then *boxlabel* toggles the box label on or off. The label, which is just the box number, appears at the center of the box. If you wish to put a more descriptive label on a box, add a label using the **text** command.

Example Pages: **71**

See Also: **boxes** Data Command, **boxes** Control Command, **boxaxes**, **boxcmap**, **boxcment**, **boxscale**, **clipbox**, **gobox**, **hidebox**, **showbox**

boxscale [*scalefactor*] | [**off**]

Scales the size of all boxes in the active data group by *scalefactor*. For example, the command `boxscale 2` will increase the size of all boxes in the data group by a factor of 2. `boxscale off` and `boxscale 1` both set the boxes to their normal size as defined by the **boxes** Data Command.

Example Pages: 73

See Also: **boxes** Data Command, **boxes** Control Command, **boxaxes**, **boxcmap**, **boxcment**, **boxlabel**, **clipbox**, **gobox**, **hidebox**, **showbox**

cb [**on** | **off** | **hide**] | [*boxparameters*]

See the **clipbox** command.

Example Pages: –

See Also: –

censize [**+, *,** /][*radius*]

Sets the size of the Point of Interest Cartesian marker. The units of *radius* depend on the units of your data. If you are displaying data with units of meters, then `censize 1` would set each axis (from the origin to the end) to one meter. You can also scale the current value by adding a constant, multiplying, or dividing by a constant, such as `censize /2` to divide the current size by two.

Example Pages: *22*, 44

See Also: **center**

cen[ter] [*x y z*] [*radius*]

Sets the position of the Point of Interest at the point (*x*, *y*, *z*). This is the rotation point in the `[o]rbit` and `[r]otate` flight modes. It is also the point where the three dimensional Cartesian axes are displayed. The optional *radius* argument sets the size of the Point of Interest in the same way the **censize** command set the size.

Example Pages: 44

See Also: interest

clearobj

Clear all data in the current data group. This is helpful when re-loading data into a data group. To clear all data in all groups, use the **gall** command before **clearobj**.

Example Pages: –

See Also: **read**, **include**

clip [*nearplane*] [*farplane*]

The **clip** command reports and sets the distances of the near and far clipping planes. The *nearplane* argument sets the distance to the near clipping plane. In computer graphics, the near clipping plane is an unseen plane parallel to your screen. Between your screen and this plane, your data will be invisible. Similarly, the far clipping plane defines the point beyond which no data are drawn by the computer.

The *nearplane* and *farplane* arguments must be positive. The value of these arguments may produce unwanted effects depending on your operating system, such as flashing and blinking. Some operating systems cannot handle a *nearplane* less than 1, or a *farplane* to *nearplane* ratio over 10,000. You may need to experiment.

Example Pages: 42

See Also: –

clipbox [**on** | **off** | **hide**] | [*boxparameters*]

A clip box is a box that highlights a portion of a data set by turning off all data outside the six planes that make up the specified box. The dimensions can be set by specifying the *boxparameters* arguments. These are of the same format as the **boxes** Data Command and can be given in two ways:

xmin,xmax ymin,ymax zmin,zmax draws a clip box using the minimum and maximum values for each coordinate. These values define the 6 planes that make up the box. In this format, the six values are written in coordinate pairs. (Note the use of commas to separate each coordinate's minimum and maximum value, but spaces are used to separate the three coordinates.)

xcen,ycen,zcen xrad,yrad,zrad draws a clip box centered at *xcen,ycen,zcen* and with half-lengths (or "radii") given by *xrad,yrad,zrad*. (Note the use of commas to delineate each center and radius value and a space to separate the center and radius groups.)

Using the **on** and **off** arguments turns the pre-defined clip box on or off. The **hide** argument continues clipping the data (points only, polygons will remain on) but turns the box off.

Example Pages: **79**

See Also: **bound**, **cb**, **boxes** Data Command

cmap *filename*

Loads a color map file *filename* for coloring particles in the active data group. For information on color map files, see "Coloring Particles and Objects."

Example Pages: **64**

See Also: **cment**, **color**, **color** *datavar* **exact**

cment *colorindex* [*R G B*]

Reports the red, green, and blue values of the *colorindex*. If the *R G B* arguments are specified, the *colorindex* is assigned the *R G B* colors. The *colorindex* should be a positive number and the values of *R*, *G*, and *B* all range from 0–1. See "Coloring Particles and Objects" for more information on coloring particles and objects.

Example Pages: **66**, 74

See Also: **color**, **cmap**, **boxcment**, **textcment**

color

Typing the **color** command without any arguments reports the coloring of the active data group. Reports reflect whether particles are being colored using the **color const** or the **color** *datavar* commands. If you are setting a constant color to all particles, then the report will be of the form:

```
coloring-by rgb R G B
```

while coloring using a data variable generates the report:

```
coloring-by num(name) min.. max mean avg
```

Here, *num* and *name* refer to the data variable, *min.. max* is the range of the data and *avg* is the mean value of the data.

Example Pages: 62

See Also: **color const**, **color** *datavar*, **color** *datavar* **exact**, **cment**, **cmap**, **bgcolor**, **boxcmap**, **boxcment**, **textcment**, **textcolor**

color const *R G B*

Sets all particles in the active data group to the color *R G B*. Each of these range in value from 0–1.

Example Pages: 63

See Also: **color**, **color** *datavar*, **color** *datavar* **exact**, **cment**, **cmap**, **bgcolor**, **boxcmap**, **boxcment**, **textcment**, **textcolor**

color *datavar* [*minval maxval*]

Sets all particles in the active data group to be colored using the data variable *datavar*. If a column of data in your data file holds the coloring information, you may define a data variable using **datavar**, then use the name from that definition as the *datavar* argument. The *minval maxval* arguments are provided to specify a mapping range in color indices. Typically, you would want to set these to a color index between 1 and $n - 1$ since the zeroth

and last entries are used for out of range values in color maps. If these arguments are omitted, then the actual range from the data is used.

Example Pages: 63

See Also: **color**, **color const**, **color** *datavar* **exact**, **cment**, **cmap**, **bgcolor**, **boxcmap**, **boxcment**, **textcment**, **textcolor**

color *datavar* ⟨**exact** | **-exact**⟩ [*baseval*]

This color command maps one to one to the color map file for the data group. Rather than distribute the colors over some range, the **exact** command maps the color index values to their exact number in the color map. For example, if a particle has a color index of 10, then using the **exact** command will set that color to the 10th entry in the color map. To turn off exact coloring, use the **-exact** command. If a *baseval* is included, then the exact color index *N* is mapped to *N* + *baseval*.

Example Pages: 64

See Also: **color**, **color const**, **color** *datavar*, **cment**, **cmap**, **bgcolor**, **boxcmap**, **boxcment**, **textcment**, **textcolor**

datavar [*num*] [*name*] [*minval maxval*]

The **datavar** command attaches a variable name *name* to a column of data. The *num* argument is just the column number minus 4 since the first three columns are always reserved for the spatial coordinates (i.e., **datavar** 0 equals column 4). These definitions should come in the `.speck` files before your data. The variable names can then be used to color, set the luminosity, or threshold data. Without any arguments, **datavar** reports the pre-defined data variables for the active data group. This report has the form:

```
datavar num name min .. max mean avg
```

where *num* is the data variable number (which maps to the column number plus 4) and *name* is the name you've given the data variable. The *minval* and *maxval* arguments set the range of the data. Without these arguments, the range of the data is used.

Example Pages: 26, **34**, 80

See Also: –

detach

Use **detach** to separate the Viewing Window and the GUI into two distinct windows. **detach**, along with the **winsize** command, can be used to run Partiview full screen.

Example Pages: **46**

See Also: **winsize**

disable

Same as the **off** command.

Example Pages: –

See Also: –

dv [*num*] [*name*] [*minval maxval*]

Same as the **datavar** command.

Example Pages: –

See Also: –

ellipsoid [**on** | **off**]

If an ellipsoid has been defined in the active data group, the **ellipsoid** Control Command toggles the display on and off. To define an ellipsoid, use the **ellipsoid** Data Command.

Example Pages: **74**

See Also: **ellipsoid** Data Command

[add] *xcen ycen zcen* **ellipsoid** [**-r** *xrad*[*,ycen,zcen*]] [**-c** *colorindex*]
[**-s** ⟨**solid** | **plane** | **wire** | **point**⟩] [**-n** *numlong*[*,numlat*]] [*transformation*]

The **ellipsoid** Data Command sets the parameters and displays an ellipsoid that will belong to the active data group. The ellipsoid will be centered on the coordinate *xcen*, *ycen*, *zcen* and is drawn with the following options.

-r *xrad*[*,yrad,zrad*] sets the *x*, *y*, *z* radii or semi-major axes. For a sphere (*xrad* = *yrad* = *zrad*), specify a value for *xrad* and omit the *yrad* and *zrad* arguments. For a plane, set one of these arguments to zero. For a line, set two of these arguments to zero. If you omit all of these arguments, the default is *xrad* = *yrad* = *zrad* = 1.

-c *colorindex* assigns a color index to the object. The color is derived from the color map for the active data group. A new color may be assigned using the **cment** command to change the color for the assigned color index. Without this argument, the *colorindex* is set to -1, which is an out of range value and will be assigned the zeroth color in the color map.

-s ⟨ **solid** | **plane** | **wire** | **point**⟩ sets the drawing style. The allowed styles are

solid a filled surface, but without any shading for perspective,

plane draws three ellipses in the *xy*, *xz*, and *yz* planes,

wire draws a wire frame ellipsoid, and

point represents the ellipsoid as points drawn at each vertex. A brighter color helps make an ellipsoid of this style more visible.

-n *numlong*[*,numlat*] sets the number of vertices, or lines of longitude and latitude in **wire** style. The value of *numlong* is equal to the number of longitude lines while *numlat* is equal to the number of latitude lines plus two (for each pole). For example, to create a wire frame sphere with lines of longitude and latitude every 10°, you would use the option **-n** 36,19. Without the *numlat* option, *numlat* = *numlong*.

transformation is a series of numbers that describe a transformation of the ellipsoid from the default world coordinates. The transformation can either be 9 or 16 numbers separated by spaces.

Example Pages: **74**

See Also: **cment**, **alpha**, **tfm**, **add**, **ellipsoid** Control Command

enable

Same as the **on** command.

Example Pages: –

See Also: –

eval *command*

Executes a Control Command from a data file. All Control Commands (which appear in this listing without the **[add]** preface) must have the **eval** preface if they are issued from a `.cf` or `.speck` file.

Example Pages: 27, **90**

See Also: **add**

every *N*

Display a random subset of the active data group by choosing every *N*th particle. To display all particles in the data group (the default), type the command `every 1`. `every 2` shows about half the data, and so on.

Example Pages: 60, **78**

See Also: **bound**, **lum**

exit

Issue this command to quit Partiview. You may also use the `[ESC]` key to quit.

Example Pages: –

See Also: –

fade [**planar** | **sph** | **linear** *refdist* | **const** *refdist*]

The **fade** command describes the light fall off law with regard to distance. Typically, we use a $1/r^2$ law, as it is in nature. However, we can set the intensity to distance relationship using the following options.

planar sets an inverse square fall off ($1/r^2$), with *r* measured as the distance from the view plane (the screen).

sph is also an inverse square fall off, but with *r* measured as the true distance from the viewpoint at the center of the screen.

linear *refdist* gives a $1/r$ light fall off, not physically accurate but perhaps useful to get a limited sense of depth.

const *refdist* sets a constant apparent brightness that is independent of distance. The *refdist* argument is defined as that distance *r* at which apparent brightness should match that of an inverse square law.

With no arguments, **fade** reports the current fade setting. The default is spherical (**sph**).

Example Pages: 61

See Also: lum

fast [**on** | **off**] [*minpixels*] [*maxpixels*]

fast is a command that alters the way points are drawn. If **fast** is off, then the points are drawn at a higher quality. Conversely, if **fast** is on, then the points are drawn more primitively but require less computational resources. **fast** takes its parameters from the **ptsize** command. If no **ptsize** has been defined for the active data group, then you may define the range here by including values for the *minpixels* and *maxpixels* arguments. See ptsize for more on these values.

Example Pages: 51

See Also: ptsize, lum

[add] filepath [**+:**]*path*

A list of directory paths, separated by colons, where data files, color maps, images, flight paths, and other necessary files are located. Include the **+:** argument to append *path* to the current file path. For example, `filepath ./data` sets the file path to the `data` folder, which is in the current folder. To append another path, use `filepath +:./images` to add the `images` folder.

Example Pages: 29

See Also: read

focallen [*distance*]

Reports (without an argument) or sets (with *distance*) the focal length. The focal length affects the stereo display (see **stereo**) as well as the speed of motion in the `[f]ly` and `[t]ranslate` flight modes.

Example Pages: –

See Also: stereo

fov *degrees*

Sets the field of view in the local (screen) *y* direction to *degrees*. Values have the range 0° to 180°. Higher values distort the view, while low values provide a "zoomed in" view. Typing `fov` without any arguments reports the current value.

Example Pages: *22*, 40

See Also: –

g*N*[**=***alias*]

Select or create data group number *N*. If group *N* exists, typing `gN` will make group *N* the active data group. If group *N* does not exist, it will be created. If you append the **=***alias* argument, the data group is given the name *alias* (no spaces between the equal sign). *alias* can then be used by **object** to refer to data groups by name, rather than number.

Example Pages: 29

See Also:

g*N command*

Similar to the **g***N*[**=***alias*] command, without the *command* argument, group *N* will be created, or selected as the active data group if it exists. The *command* argument can take any relevant Partiview Control Command that you would like to apply to group *N*. For example, `g4 color const 1 0 0` will turn all particles in group 4 red.

Example Pages: –

See Also: –

gall -v | *command*

The **gall** command is designed to perform some action on all data groups. If you issue **gall -v**, Partiview will report all the defined data groups and their display status (on or off). If you issue **gall** *command*, then *command* will act on all data groups. For example, if you want to multiply the **slum** value of all data groups, you would enter `gall slum *2`.

Example Pages: 33

See Also: gN-command

gobox *boxnumber*

The **gobox** command shifts the Point of Interest to the center of box *boxnumber*. The *boxnumber* is designated in the **boxes** command using the **-n** option.

Example Pages: 73

See Also: **boxes** Data Command, **boxes** Control Command, **boxaxes**, **boxcmap**, **boxcment**, **boxlabel**, **boxscale**, **clipbox**, **hidebox**, **showbox**

hidebox *level*

If multiple boxes are defined, a subset of boxes can be toggled on and off using the **showbox** and **hidebox** commands. Boxes are grouped by the *level* specified in the **boxes** Data Command. By giving several boxes the same *level* number, these boxes can then be toggled on or off while leaving other boxes unchanged. For example, if you have defined 10 boxes and five of them were defined with a `-l 3` argument, then these five boxes can be turned off using the `hidebox 3` command or turned on using the `showbox 3` command. In addition, a listing of level numbers can be used, such as `showbox 2 23 15` to show boxes that have *level* values of 2, 15, and 23.

Example Pages: 72

See Also: **boxes** Data Command, **boxes** Control Command, **boxaxes**, **boxcmap**, **boxcment**, **boxlabel**, **boxscale**, **clipbox**, **gobox**, **showbox**

hist [**-n** *numbins*] [**-l**] [**-c**] [**-t**] *datavar* [*minval*] [*maxval*]

Reports a histogram of the data field *datavar* in the Console Window. The arguments are defined as:

-n *numbins* sets the number of bins for the histogram (default = 11),

-l sets the intervals to be logarithmically-spaced (as long as the data range does not include zero),

-c counts only the particles displayed within a clip box,

-t counts only particles in a subset defined by **thresh** or **only=**,

datavar is the data variable to return the distribution for,

minval is an optional minimum value for the distribution range, and

maxval is an optional maximum value for the distribution range.

The available data fields are reported to you when you use the **datavar** command on the active data group. It will return the defined data fields and their range. The order of the arguments is important; some may not take effect if they are not issued in this order.

Example Pages: 86

See Also: **bound**, **thresh**, **only=**

[add] include *filename*

Read a file containing data (a `.speck` file) or data commands (a `.cf` file). If you use this in a file, you do not need the **add** preface. This is the same as the **read** command.

Example Pages: –

See Also: **read**

int[erest] [*x y z*] [*radius*]

Same as the **center** command.

Example Pages: **44**

See Also: **center**

jump [*x y z*] [R_x R_y R_z]

The **jump** command, without any arguments, returns the current position in the format *x y z* R_x R_y R_z, where *x y z* is your position and R_x R_y R_z are the rotation angles of your point of view. These three angles are rotations about the *x*, *y*, and *z* axes. If the *x y z* arguments are included, then your position is reset to the *x*, *y*, *z* specified, the rotation angles will remain the same if unspecified. For example, if you are 5 units out on the +*z* axis (`jump 0 0 5 0 0 0`), then you can rotate the view about *z* by changing the R_z parameter (try `jump 0 0 5 0 0 45`).

Example Pages: **39**

See Also: **where**

label[s] [**on** | **off**]

Turns on (or off) the labels for the active data group (if labels are specified). Typing `label` toggles their display, but **label[s]** can also take the **on** or **off** arguments explicitly. The Label toggle button also turns the labels on and off for the active data group.

Example Pages: *21*, **68**

See Also: **labelsize**, **labelminpixels**, **laxes**, **textcment**, **textcolor**, **boxlabel**

labelmin[pixels] [**+, *,** /][*size*]

Describes the minimum size in pixels before a label will be drawn. By default, the size of the label is proportional to its distance. Set the *size* argument low (relative to the defined label size), and labels will be drawn far off into the distance to the point where they are unreadable. Set *size* high (again, relative to the defined label size), and labels will only be drawn nearby.

Example Pages: *22*, **69**

See Also: **labels**, **labelsize**, **laxes**, **textcment**, **textcolor**, **boxlabel**

l[abel]size [**+, *,** /][*size*]

Set the *size* of the labels in the distance units of your data. If, for example, your Partiview session has data that are in units of meters, then the *size* argument will be in meters (as scaled by the graphics, of course). The labels in Partiview are drawn proportional to your distance from them. The size of the labels can also be scaled by the [**+, *,** /], such as `lsize *2.5`. The relative size of the labels within a data group may be controlled using the **-size** argument in the **text** command.

Example Pages: *22*, **69**

See Also: **labels**, **labelminpixels**, **laxes**, **textcment**, **textcolor**, **boxlabel**, **text**

laxes [**on** | **off**]

Toggles the label axes on and off. The label axes (laxes) are an *x*, *y*, *z* Cartesian axis placed at each label location. By default, they are on.

Example Pages: **68**

See Also: **labels**, **labelsize**, **labelminpixels**, **textcment**, **textcolor**, **boxlabel**

lum

The **lum** command without any arguments reports the current luminosity setting for the active data group. If the group is set to a constant luminosity via the **lum const** command, then Partiview reports

```
lum-by constant L
```

where *L* is the value given to all particles. If the luminosity is set to a data variable in your `.speck` file using the **lum** *datavar* command, then Partiview will report

```
lum-by num (name) [min..  max mean avg]
```

where *num* is the data variable number, *name* is the name given to the data variable, *min.. max* is the range of your data, and *avg* is the mean value.

Example Pages: 58

See Also: **lum const**, **lum** *datavar*, **slum**, **alpha**, **datavar**, **fade**, **psize**, **every**, **ptsize**, **fast**, **polysize**, **polylumvar**

lum const [**+, *,** /]*L*

Sets a constant particle luminosity, *L*, for the active data group. This command insures that all points in the given data group have the same luminosity.

Example Pages: 59

See Also: **lum**, **lum** *datavar*, **slum**, **alpha**, **datavar**, **fade**, **psize**, **every**, **ptsize**, **fast**, **polysize**, **polylumvar**

lum *datavar* [*min max*]

The **lum** *datavar* command allows for a variation in luminosity within one data group. If a particular data group has luminosity information as part of the data set (i.e., one of the columns in your `.speck` file is luminosity), then the luminosity data field *datavar* can be mapped to values between *min* and *max*. Without the *min* and *max* arguments, the values are mapped to the range of the data.

Example Pages: 59

See Also: **lum**, **lum const**, **slum**, **alpha**, **datavar**, **fade**, **psize**, **every**, **ptsize**, **fast**, **polysize**, **polylumvar**

[add] mesh [**-t** *texnum*] [**-c** *colorindex*] [**-s** ⟨**solid** | **wire** | **point**⟩]

Given a set of x, y, z points, **mesh** will draw a series of lines connecting the points. If the **-c** *colorindex* argument is specified, the mesh is given the R, G, B color pointed to by the *colorindex* set by the **cment** command. The **-s** option sets the drawing style according to these styles:

solid draws a filled polygon surface (the default),

wire draws a wire-frame mesh (just lines), and

point draws points at each vertex.

The default is **solid** if the **-s** argument is unspecified. The **-t** *texnum* argument specifies a texture number, set via the **texture** command, to map on the mesh. The texture is mapped according to the u and v coordinates specified after the (x, y, z) coordinate. The u and v coordinates range from 0 to 1, rescaling the size of the texture to this range and mapping on the mesh. When expressing the **mesh** command in a file, you must use the following form:

```
mesh -c 1 -s wire {
numu numv
x1 y1 z1 u1 v1
x2 y2 z2 u2 v2
...
xN yN zN uN vN
}
```

Note the use of the opening and closing curly braces after the **mesh** command.
numu numv specify the dimensions of the mesh. For a line between two points, *numu* will be equal to 1. For a quadrilateral grid, the *numu numv* arguments will specify the grid dimensions. For a square grid, these numbers will be equal to one another.

Example Pages: 76

See Also: **ellipsoid** Data Command, **boxes** Data Command

object [*alias* [*command*]]

The **object** command provides a reference to data groups by name rather than number. Typing `object` with no arguments will report the *alias* of the active data group. Including the *alias* argument will change the active data group to the group specified by *alias*. Including the *command* argument will execute the command on the data group *alias*. For example, typing `object stars on` will turn on and activate the data group called “stars.” Typing `object mygroup lsize*5` will activate the data group “mygroup” and increase the label size by 5, but will not display the group if it is off. **object** provides a way to activate a data group without having to remember its group number or access its group button.

Example Pages: 32

See Also: –

object g*N***=***alias*

Issued in a data file, the **object** Data Command defines a group and sets its *alias*.

Example Pages: –

See Also: g*N*

off

Turn off the display of the active data group.

Example Pages: –

See Also: disable

on

Display the active data group.

Example Pages: –

See Also: enable

only= *datavar* ⟨ *value* | *minval maxval* | *<maxval* | *>minval* ⟩

The **only=** command creates a subset of particles from the active data group based on the *datavar* data. The selection criteria can be made in several ways. By including the *value* argument, only the data whose data variable equals *value* will be displayed. To display a range of data, you may specify a minimum and maximum value in *minval maxval*. Or, display only those data that have are greater than or less than a given value using the *<maxval* and *>minval* arguments.

Example Pages: 82

See Also: only+, only-, thresh, datavar, see, sel

only+ *datavar* ⟨ *value* | *minval maxval* | *<maxval* | *>minval* ⟩

Using the same selection criteria as the **only=** command, the **only+** command allows you to add data to the current selection set. If a selection set has been defined using the **only=** command, then the **only+** adds data to the selection. If no selections have been defined, then **only+** acts like **only=** and thresholds the data.

Example Pages: 83

See Also: only=, only-, thresh, datavar, see, sel

only- *datavar* ⟨ *value* | *minval maxval* | *<maxval* | *>minval* ⟩

Using the same selection criteria as the **only=** command, the **only-** command allows you to remove data from the current selection set. If a selection set has been defined using the **only=** command, then the **only-** removes data from the selection. If no selections have been defined, then **only-** removes all data.

Example Pages: 83

See Also: only=, only+, thresh, datavar, see, sel

point[s] [**on** | **off**]

Turn on (or off) the display of points for the active data group. Simply typing **point** toggles the display. The Point toggle button also toggles the display of points.

Example Pages: *21*, **49**

See Also: **ptsize**, **fast**

poly[gons] [**on** | **off**]

Turns on (or off) the polygons. Typing **poly** alone will toggle the polygons on or off. Also, the Polygon Toggle Button will turn the polygons on and off.

Example Pages: *21*, **52**

See Also: **polysides**, **polysize**, **polyminpixels**, **polylumvar**, **polyorivar**, **texture** Data Command, **texture** Control Command, **alpha**, **lum**, **slum**

polylum[var] [**a** | **r** | *datavar*]

Set how the polygons are sized. The default, `polylum r`, sets the radius of the polygon to be equal to the **polysize** value. The **a** argument sets the radius of the polygon according to the area of the polygon set by **polysize**. For example, if **polysides** is 4 and **polysize** is set to 10, then with `polylum r` the radius of the polygon would be 10, while with `polylum a` the radius would be $\sqrt{10}$ (since the area of a square is $2 * r^2$. If *datavar* is specified, then the polygons are sized according to the data variable. The default is -1, which scales the size to those set by the luminosity commands **lum**, **slum**, **polysize**, and a few others.

Example Pages: **54**

See Also: **polygons**, **polysides**, **polysize**, **polyminpixels**, **polyorivar**, **texture** Data Command, **texture** Control Command, **alpha**, **lum**, **slum**

polymin[pixels] [*minpixels maxpixels*]

This command sets the minimum and maximum size (in pixels) for the polygons to be drawn. The *minpixels* value sets the minimum size for a polygon to be drawn, that is, no polygons will be drawn below size *minpixels*. The *maxpixels* argument sets the maximum size for a polygon. Without any arguments, **polymin** reports the current values.

Example Pages: **55**

See Also: **polygons**, **polysides**, **polysize**, **polylumvar**, **polyorivar**, **texture** Data Command, **texture** Control Command, **alpha**, **lum**, **slum**

polyorivar [*num*]

Reports the polygon orientation information. Orientation is specified as 6 columns of numbers in the `.speck` file. The six numbers are two sets of three that describe orthogonal vectors in the plane of the polygon. The first of these numbers is defined using a **datavar** command. **polyorivar** is then used in the data file to tell Partiview which column (**datavar** name) is the first orientation column. The *num* refers to the **datavar** number. The default value of *num* is -1 which draws polygons that always face the screen.

Example Pages: **56**

See Also: **polygons**, **polysides**, **polysize**, **polyminpixels**, **polylumvar**, **texture** Data Command, **texture** Control Command, **alpha**, **lum**, **slum**

polyside[s] [*numsides*]

Set the number of sides of the polygons in the active data group. *numsides* ranges from 3 to 16. The default value is 11, which produces a round polygon. If you are placing textures on your polygons, the optimal number of sides is 4. Without the *numsides* argument, the current value is reported.

Example Pages: *22*, **55**

See Also: **polygons**, **polysize**, **polyminpixels**, **polylumvar**, **polyorivar**, **texture** Data Command, **texture** Control Command, **alpha**, **lum**, **slum**

polysize [**+, *,** /][*size*]

Set the size of the polygons in distance units. The *size* argument, like the **labelsize** command, is relative to the units of distance of the data. The size of the polygon is then scaled by the **lum** and **slum** commands.

Example Pages: *22*, **52**, 54

See Also: **polygons**, **polysides**, **polyminpixels**, **polylumvar**, **polyorivar**, **texture** Data Command, **texture** Control Command, **alpha**, **lum**, **slum**

psize [**+, *,** /][*scalefactor*]

A global scale factor that scales the luminosity of the points by *scalefactor*. A *scalefactor* of 0 gives the points zero luminosity. A high scale factor allows points to be visible from large distances. **psize** does not affect polygon size, only the point size.

Example Pages: **60**

See Also: **points**, **lum**, **lum const**, **slum**

ptsize [*minpixels*] [*maxpixels*]

Sets the range of apparent sizes in pixels for all points in the active data group. Points have a range of sizes that is dependent your distance from them. The size of a point will generally grow as you approach it. However, it is possible to saturate the size by setting a maximum **slum** value, for example. *minpixels* is the minimum size in pixels a point must be before it is drawn. *maxpixels* sets the maximum size in pixels a point will reach. Most graphics systems have an upper limit for the size of a point, which is usually around 10–20 pixels.

Example Pages: **50**

See Also: **fast**, **points**

read *filename*

Use this command to read a data file *filename* into Partiview. The **read** command acts as both a Data Command as well as a Control Command. This means that you may use **read** in the Command Line interactively without the **add** prefix, and you may use **read** in a data file without the **eval** prefix.

Example Pages: **29**

See Also: **include**, **filepath**

see [**all** | **none** | ⟨ **thresh** | **-thresh** ⟩ | *name*]

The **see** command allows you to toggle between various data thresholding scenarios. The **all** and **none** show all, or none, of the particles in the active data group. If a subset of data have been defined using the **thresh** or **only=** commands, then **thresh** displays these settings. **-thresh** is the anti-thresh—it displays the data hidden by the current thresh, and removes the data displayed by the current thresh, the inverse view. Using the **sel** command, selection settings can be saved and used here as *name*.

Example Pages: **81**, 83

See Also: **only=**, **only+**, **only-**, **thresh**, **datavar**, **sel**

sel *name* **= thresh**

The **sel** command allows you to save a threshold scenario with the name *name*. These expressions can then be used with the **see** command. For example, if you used **thresh** or **only=** to display a subset of data, then the command `sel myselect = thresh` would save the threshold setting with the name `myselect`. Then, to toggle views you would type `see myselect`, then `see all` to see all particles again.

Example Pages: **83**

See Also: **only=**, **only+**, **only-**, **thresh**, **datavar**, **see**

showbox *level*

If multiple boxes are defined, a subset of boxes can be toggled on and off using the **showbox** and **hidebox** commands. Boxes are grouped by the *level* specified in the **boxes** command. By giving several boxes the same *level* number, these boxes can then be toggled on or off while leaving other boxes unchanged. For example, if you have defined 10 boxes and five of them were defined with a `-l 3` argument, then these five boxes can be turned off using the `hidebox 3` command or turned on using the `showbox 3` command. In addition, a listing of level numbers can be used, such as `showbox 2 23 15` to show boxes that have *level* values of 2, 15, and 23.

Example Pages: 72

See Also: **boxes** Data Command, **boxes** Control Command, **boxaxes**, **boxcmap**, **boxcment**, **boxlabel**, **boxscale**, **clipbox**, **gobox**, **hidebox**

slum [**+, *,** /][*scalefactor*]

A luminosity scaling factor that increases the brightness of all points in a data group. **slum** is a data variable-specific scale factor, meaning you could have several **slum** settings, one for each data variable. For example, if you have a file with two data variable columns, one for the brightness of the object, defined in a **datavar** command as `lumin`, and one for the actual size called `diam`, then one could set the luminosity, or size, of the particles to either of these using these commands:

```
lum lumin
slum 10.5
lum diam
slum 2.3
```

Once you have set the **slum** value for each of these, then you can toggle between settings using the **lum** *datavar* command, as in `lum lumin` and `lum diam`. Your **slum** settings will remain for each *datavar* so that you don't have to set the scaling each time you wish to view an alternate representation of the data.

Example Pages: *22*, 60

See Also: **lum**, **lum const**, **lum** *datavar*, **datavar**, **polysize**

snapset [**-n** *framenumber*] *filestem*

snapset defines a filename for writing a snapshot of the current view. This is how Partiview exports a screen grab of the OpenGL output. The image is written into a compressed Portable PixMap (`*.ppm`) file by default, but can be written into various image types (`tif`, `jpeg`, `bmp`, etc.) given you have the `convert(1)` program installed and in your path (this applies to UNIX and UNIX-like environments, i.e., not Windows).

The *filestem* argument sets the file name (minus extension). Partiview will append the frame number (beginning at zero) and the file extension (`.ppm`) along with the gzip extension (`.gz`). The *filestem* may also be expressed as a C `printf()`-like format string with the frame number acting as the argument for the format specifier (see examples below). You may also provide the *filestem* with a frame number using the **-n** *framenumber* option. Whatever frame number you begin with, whether it is the default (0) or one specified with the **-n** option, subsequent snapshots will take on the current frame number plus one.

Some examples of the **snapset** command are:

`snapset -n 4 picture`: Here we issue **snapset** with a frame number of 4 and a *filestem* called `picture`. Once the **snapshot** command is executed, the resulting file will be called: `picture.004.ppm.gz`. Note here that the default frame number (004) is 3 digits in length. To have a longer or shorter frame number, we must supply a format specifier.

`snapset -n 20 picture%05d.ppm`: Now we have a frame number of 20 and we tell Partiview to create a file called picture with a frame number that is 5 digits in length (filling in with leading zeros) with an extension of `ppm` without compression. The resulting file name is: `picture00020.ppm`.

`snapset -n 10 picture%03d.tif`: Similar to the above example, except here we set the frame number to 10 and specify a frame number of 3 digits. Then we specify a `tif` extension, creating a TIFF file. The resulting file will be called `picture010.tif`.

once **snapset** is executed, you must use **snapshot** to export the image.

Example Pages: 47

See Also: **snapshot**

snapshot [*filestem*]

Capture a screen grab of the Viewing Window. The argument *filestem* sets the filename just as it does in the command **snapset**. Typically one would set the output type and filename using **snapset**, then issue the **snapshot** command to save the image.

Example Pages: 47

See Also: snapset

stereo [**on** | **off** | **redcyan** | **glasses** | **cross** | **left** | **right**] [*separation*]

Without any arguments, **stereo** reports the current stereo setting. The **on** and **off** arguments are used to turn the stereo display on or off (you may also use the `s` key in the Viewing Window). **redcyan** should be used with red-green or red-blue glasses. If you have red-blue glasses, we recommend setting the particle color to purple (`color const 1 0 1`) so that the image splits into red and blue. The **cross** argument splits the Viewing Window into two identical views for cross-eyed stereo. The **left** and **right** arguments display only the left and right eye's view which may be useful for stereo snapshots. The *separation* argument describes the separation of the two images. Typical values range between 0.02 and 0.1 (or -0.02 to -0.2 if you want to swap eyes). Also see

focallen which sets the distance to a typical object of interest—left- and right-eye images of an object at that distance will coincide on the screen. Virtual world eyes will be separated by a distance 2 $\times$ **focallen** $\times$ *separation* with a convergence angle of 2 $\times$ arctan(*separation*).

Example Pages: –

See Also: focallen

[add] *x y z* **text** [**-size** *value*] *yourlabel*

Typically used in a data file, a label can be added interactively via the Command Line by prefacing it with **add**. Just supply the x, y, z location of the label and the text of the label in place of the *yourlabel* argument. For example, to interactively place a label that reads "Star Number One" (without the quotes) at $(x, y, z) = (400, 200, 200)$, type

```
add 400 200 200 text Star Number One
```

in the Command Line. Or, simply include the line `400 200 200 text Star Number One` inside a data or config file. Labels can be sized relative to one another using the **-size** option, increasing or decreasing the size of the label by the factor *value*. For example, to set a label to half its value, use `-size 0.5` option.

Example Pages: 35, **68**

See Also: **add**, **labels**, **labelsize**, **labelminpixels**, **laxes**, **textcment**, **textcolor**

textcment *colorindex* [*R G B*]

Define a text color *R G B* to the color index *colorindex* for the active data group. Use this in conjunction with the **textcolor** command.

Example Pages: **67**, 68

See Also: **textcolor**, **labels**, **labelsize**, **labelminpixels**, **laxes**, **text**

[add] textcolor *colorindex*

Associate the text color for a given data group or a portion of a data group with the *colorindex* assigned via the **textcment** command. Note: this command does not appear to work from the Command Line or from within a config file. It must appear in the `.speck` file where the labels are listed. (so the **[add]** preface is listed here for consistency but has no effect on the usage of **textcolor**.)

Example Pages: **67**, 68

See Also: **textcment**, **labels**, **labelsize**, **labelminpixels**, **laxes**, **text**

texture [**on** | **off**]

Toggles the textures on and off. The Texture toggle button also turns the textures on and off.

Example Pages: *21*, **58**

See Also: **texture** Data Command

[add] texture [**-a** | **-i** | **-A** | **-O** | **-M** | **-D**] *texnum filename*

Associates a texture file *filename* with a texture number *texnum* for a particular particle or many particles. Using the **datavar** command, a data variable can be defined in your `.speck` file to hold the texture number for each particle. Then, using **texture**, the same texture numbers are assigned to the desired files. Display options include:

-a Alpha: Image is taken as opacity data, rather than luminance data (the GL_ALPHA texture format). A single-channel image is normally used as luminance data.

-i Intensity: Computes the intensity of each pixel and uses it to form an alpha channel for 1 or 3 channel images.

-A Additive blending: Texture will add to, not obscure, the brightness of textures that are behind it.

-O Over compositing: Texture will obscure features behind it according to the alpha values at each point.

-M Modulate: Multiply the texture brightness/color values by the color map-determined color of each particle.

-D Decal: The texture's polygon color is determined entirely by the texture, suppressing any color map information.

To turn textures on and off, see the **texture** Control Command.

Example Pages: **57**

See Also: **texture** Control Command, **texturevar**, **txscale**, **alpha**, **polygons**, **polyminpixels**, **polyorivar**, **polysides**, **polysize**, **polylumvar**, **color**

[add] texturevar *datavar*

Sets which *datavar* (column in a file) to be read as texture numbers. Texture numbers are defined in the *texnum* argument of the **texture** Data Command. For example, if column 6 in a given data file is the texture number assigned to each particle, then `texturevar 2` would tell Partiview that column 6 is where the texture information is located for each

particle (recall the first three columns of any data file are always x, y, z, and Partiview begins counting from zero for columns thereafter).

Example Pages: 57

See Also: **texture** Data Command, **texture** Control Command, **datavar**

tfm [*matrix*]

Apply a transform to the active data group. The *matrix* takes the form $tx\ ty\ tz\ R_x\ R_y\ R_z$, where $tx\ ty\ tz$ are the x, y, z translations and $R_x\ R_y\ R_z$ are the rotations about the x, y, z axes, respectively. For example, if you would like to move a data group 10 units of distance in the y direction, then you would issue the command: `tfm 0 10 0 0 0 0` (making sure that the group you want to transform is the active data group).

Example Pages: –

See Also: –

thresh [**on** | **off** | *datavar* $\langle$ *minval maxval* | $<$*maxval* | $>$*minval* $\rangle$]

Threshold data in the active data group. Without an argument, **thresh** reports the current threshold settings. **on** and **off** toggle the thresholding on and off. *datavar minval maxval* selects data that only have *datavar* values between *minval* and *maxval*. Similarly, the thresholding range can be extended from the minimum value in the *datavar* up to some *maxval* in *datavar* $<$*maxval* or from some *minval* up to the maximum value in the *datavar* using *datavar* $>$*minval*. *minval* and *maxval* are always included in the resulting subset of data. Threshold settings can be saved in selection expressions that are defined using the **sel** command.

Example Pages: 80

See Also: **only=**, **only+**, **only-**, **datavar**, **see**, **sel**

txscale [*size*]

Set the size of the texture which sits on the polygon. A scale factor of 0.5 sets the texture to fit perfectly on the polygon if **polysides** is 4. If *size* is set too low, the texture will overflow the polygon size, causing a loss of part of the image. If *size* is set too large, then the texture will disappear. The default value is 0.5, which scales the texture equal to the polygon size.

Example Pages: 58

See Also: **texture** Data Command, **texture** Control Command, **polysize**, **polysides**

update

Use **update** to manually update the display. This is probably only useful from subprocesses invoked via the **async** command.

Example Pages: –

See Also: –

version

Reports the version number for Partiview, as well as copyright information.

Example Pages: –

See Also: –

w[here]

Use the command **where** to generate a report on the current position and viewing angles. The report takes the form:

camera at x_w y_w z_w (w) x_g y_g z_g (gN)
looking to $\hat{x}_w$ $\hat{y}_w$ $\hat{z}_w$ (w) $\hat{x}_g$ $\hat{y}_g$ $\hat{z}_g$ (gN)
jump x_w y_w z_w R_x R_y R_z *scale*
c2w: a_{xx} a_{xy} a_{xz} 0 a_{yx} a_{yy} a_{yz} 0 a_{zx} a_{zy} a_{zz} 0 x_w y_w z_w 1
c2obj: b_{xx} b_{xy} b_{xz} 0 b_{yx} b_{yy} b_{yz} 0 b_{zx} b_{zy} b_{zz} 0 x_g y_g z_g 1

where x, y, z are the values of your current position, $\hat{x}$, $\hat{y}$, $\hat{z}$ are unit vectors that describe the forward direction vector relative to the world coordinates and the group coordinates, R_x, R_y, R_z are the rotation angles about the x, y, and z axes, and the `c2w` a_{ij} and `c2w` a_{ij} coefficients describe a transformation matrix from camera (your view) to world coordinates and the object coordinates.

Example Pages: 38

See Also: jump

winsize [*xsize* [*ysize*]] [$\pm$*xpos*$\pm$*ypos*]

Sets the size of the Viewing Window. Without any arguments, **winsize** reports the current size. With only one argument, the window size is set to *xsize*, preserving the current aspect ratio. Specifying *xsize* and *ysize* sets the window size explicitly. The $\pm$*xpos* and $\pm$*ypos* arguments set the location of the window on your screen. Positive values are measured from the top/left corner, negative values are measured from the bottom/right corner. Positive and negative values can be mixed; for example, `winsize -0+0` will place the window in the top/right corner. Note that specifying a *ypos* of +0 will place the top of the window off screen—the top of the GUI will be placed in the top/left corner. Use this command with **detach** to customize your display.

Example Pages: 45

See Also: detach

Index

active data group, 22, 23, 33
add, 28, 91
alpha, 55, 91
Alpha Slider, 22
async, 91
att Button, 25

Back Button, 24
bgcolor, 37, 92
bound, 79, 85, 92
Box Toggle Button, 21
boxaxes, 68, 93
boxcmap, 67, 93
boxcment, 67, 73, 93
boxes, 70–73
    changing focus to, 73
    coloring, 67, 73
    hiding, 72
    labeling, 71
    scaling, 73
    showing, 72
boxes, 71, 93
boxes Data Command, 71, 94
boxlabel, 71, 94
boxscale, 73, 95
cb, 79, 95
censize, 44, 95
Censize Slider, 22
center, 44, 95
clearobj, 96
clip, 42, 96
clipbox, 79, 96
cmap, 64, 97
cment, 66, 97
color, 98
color const, 63, 98
color *datavar*, 63, 98
color *datavar* **exact**, 99
color *datavar* **exact**, 64
color maps, 64
coloring particles, 62–68
Command Line, 25
comments in files, 26
config files, 35
Console Window, 25

data, *see also* particles
    formatting, 26, 34
    importing, 34
    labeling, 27, 35

data groups, 31–33
activating, 33
defining, 32
referring to, 32
datavar, 26, 34, 80, 99
detach, 46, 100
disable, 100
dv, 100

ellipsoid, 41, 74, 100
ellipsoid Data Command, 40, 74, 101
ellipsoids, 73–76
enable, 102
eval, 102
every, 78, 102
exit, 102
exporting
graphics display, 47

fade, 61, 103
fast, 51, 103
Feed Button, 24
field of view, 40–42
file types
`.bat`, 35
`.cf`, 35
`.label`, 35
`.partiviewrc`, 17, 47
`.speck`, 34
filepath Data Command, 29, 104
flight mode
changing, 18
Fly, 18
Orbit, 18
Rotate, 18
Translate, 18
Flight Mode Menu, 21
flight path
control
att Button, 25
Flight Path Slider, 25
Frame Controls, 25
Path Button, 25
Play Button, 25
loading, 25
playing, 25
Flight Path Slider, 25
flight scale
linear, 18
logarithmic, 18
Fly Flight Mode, 18
focallen, 104
fov, 40, 104
FOV Slider, 22

gall, 33, 105
g*N*, 32, 104
g*N command*, 105
gobox, 73, 105
Group Buttons, 23
Groups Menu, 21

hidebox, 72, 106
hist, 86, 106
Home Button, 21

Importing data, 34–36
include Data Command, 107
interest, 44, 107

jump, 107

label colors, 67
label format, 27, 35
Label Toggle Button, 21
Labelmin Slider, 22
labelminpixels, 69, 108
labels, 68, 107
labelsize, 30, 69, 108
Labelsize Slider, 22
laxes, 30, 68, 108
linear flight scale, 18
logarithmic flight scale, 18
lsize, 30, 69
lum, 109
lum const, 31, 59, 109
lum *datavar*, 59, 109
luminosity of particles, 58–62

menu
    Flight Mode, 21
    Groups, 21
    More, 21
    Slider, 22
mesh Data Command, 76, 110
More Menu, 21
mouse controls
    Group Buttons, 23
    navigation, 18

navigation, 18

object, 32, 111
object Data Command, 111
off, 111
on, 111
only+, 83, 112
only-, 83, 112
only=, 82, 112
Orbit Flight Mode, 18

particle
    colors, 62–68
    labels, 68–69
    luminosity, 58–62
    statistical information, 85–88
    subsets, 78
        clipboxes, 79
        on specific values, 82
        random, 78
        saving, 83
        selection expressions, 83
        thresholding, 80–82
Partiview
    detaching the GUI, 46
    entering commands, 25
    flight modes, 18
    help, 10
    license, 9
    navigation, 17
    quitting, 17
    requirements, 11
    resizing the Viewing Window, 45

running full screen, 46
starting, 16
support, 10
testing, 13–15
Path Button, 25
Play Button, 25
Point of Interest, 44
changing, 19, 44
size of (**censize**), 44
Point Toggle Button, 21, 49
points, 49, 113
Polygon Toggle Button, 21
polygons, 52, 113
polylumvar, 54, 113
polyminpixels, 55, 114
polyorivar, 56, 114
polysides, 55, 114
Polysides Slider, 22
polysize, 52, 115
Polysize Slider, 22
psize, 60, 115
ptsize, 50, 115

read, 30, 116
Reference Time Display, 24
Rotate Flight Mode, 18

see, 81, 116
sel, 83, 116
selecting data points, 19
showbox, 72, 117
slider
Alpha, 22
Censize, 22
Flight Path, 25
FOV, 22
Labelmin, 22
Labelsize, 22
Polysides, 22
Polysize, 22
range
linear, 23
logarithmic, 23
Slum, 22
Speed, 24
Slider Menu, 22
Slider Scale Button, 22
slum, 60, 117
Slum Slider, 22
snapset, 47, 118
snapshot, 47, 119
Speed Slider, 24
Speed Toggle Button, 24
start files, 35
stereo, 119

text Data Command, 27, 35, 68, 119
textcment, 67, 68, 120
textcolor Data Command, 67, 68, 120
texture, 58, 120
texture Data Command, 57, 121
Texture Toggle Button, 21
textures, 57
texturevar Data Command, 27, 35, 57, 121
tfm, 122

thresh, 80, 122
Time Control Buttons, 24
time controls
    Back Button, 24
    Feed Button, 24
    Reference Time Display, 24
    Speed Slider, 24
    Speed Toggle Button, 24
    Time Control Buttons, 24
    Time Dial, 24
    Time Display, 24
    Trip Button, 24
Time Dial, 24
Time Display, 24
toggle button
    Box, 21
    Label, 21
    Point, 21
    Polygon, 21
    Speed, 24
    Texture, 21
Translate Flight Mode, 18
Trip Button, 24
txscale, 58, 123

update, 123
user interface, 20–25

version, 123
Virtual Director, 6

where, 38, 123
winsize, 45, 124