partiview is a program that enables you to visualize and animate particle data. partiview runs on relatively simple desktops and laptops, but is mostly compatible with its big brother VirDir. This document helps you installing and running the development version of partiview.

Contents

1 Installation

This release has been tried on Linux (Fedora, Ubuntu, etc.), Mac OS X, Irix and Windows.

partiview needs two libraries to compile: OpenGL (or MESA) for the drawing operations, and FLTK for the graphical user interface. These libraries are known to work on MS-Windows as well as many Unix flavors.

1.1 OpenGL (possibly via Mesa)

Most platforms will have it installed already, whether as libMesaGL or libGL. Our configure script (see below) should take care of the two possible options.

Homepage: http://mesa3d.sourceforge.net/

Redhat packages: (part of powertools I believe)

1.2 FLTK

Also make sure FLTK is installed, from fltk.org. FLTK versions 1.1.x (e.g. 1.1.7, 1.1.9, ...) work. FLTK 2 will not work with partiview.

If you're not sure whether you already have it, try

```bash
% locate libfltk.a
% locate Fl_Slider.h

if they fail, then

% cd <where-ever>/fltk-1.1.9
% make install
```

(you only need it if you want to recompile partiview at some point, not if you just want to run it, since FLTK is built-in to partiview binaries.)

Homepage: http://www.fltk.org/

Find rpms: http://rpmfind.net
FLTK is under continuous development. Versions from 1.1.1 through 1.1.9 have been successfully tested with partview. Some problems with other versions exist, but 1.1.4 is also known to work.

1.3 partview

You can decide to use a branded version, usually available as a tar or zip file, or use the CVS (see below). Extract the tarball, and install the program from within the src directory:

% tar xzf partview-0.6.tar.gz
% cd partview-0.6/src
% make clean (if you really must compile a new executable)
% ./configure (GNU autoconf toolset to ease installation)
% make depend (might need to make new local dependencies)
% make partview (should not have to edit Makefile anymore)

If you encounter difficulties of locating either the FLTK or MESA/OpenGL libraries, configure script options can specify them: -with-fltk=dirname names the directory which contains the 1ib and FL subdirectories, -with-mesa=dirname can specify the Mesa installation directory [??], and -with-kira=dirname names the Starlab directory, whose default value is taken from environment variable STARLAB_PATH if that is set.

1.4 CVS

The current source code of partview is always available from CVS, with public anonymous read-only access. Occasionally we stamp out a packaged release, too, but looking to CVS is best.

(Partview developers can request a non-anonymous CVS account from Peter Teuben -teuben@astro.umd.edu.)

Currently the CVS repository machine is cvs.astro.umd.edu. Here’s a sample session with some commonly used CVS commands:

```
setenv CVSRoot :pserver:anonymous@cvs.astro.umd.edu:/home/cvuser
setenv CVSEDITOR emacs

cvs login (only needed once, and only for pserver type access)
cvs checkout partview # get a new local sandbox to work in, or
cd partview # goto the root directory of partview
cvs -n -q update # check if others had made any changes
cvs update # if so, update your sandbox and/or resolve conflicts
cd partview/src # goto the 'src' directory of partview
./configure --with-fltk=/some/where/fltk-1.1.something/ ...
emacs partibrains.c # edit some files
make all # compile the program
./partview # test the program
emacs kira_parti.cc # edit another file
```
1.5 Compiling under Windows

Partview can be compiled from the command line on Windows using either the Microsoft Visual C tools (cl, nmake, etc.) or using gcc/g++ with MinGW32, MSYS and w32api. The MinGW route is currently the only way to compile with kira/Starlab support. There’s no provision for building partview within the MS Visual Studio GUI.

To compile with Microsoft C:

1. Install FLTK using MS Visual C++ as described in its documentation.
2. Unpack the partview distribution from its tarball or via CVS.
3. Edit the file partview/src/partview.mak, changing FLTK_DIR as appropriate.
4. Run the vccvars32.bat script from the Developer Studio Bin directory; this will set the MSVCDIR environment variable, add the Bin directory to PATH, etc.
5. In the partview/src directory, compile with

    nmake -f partview.mak

Dependencies are not properly maintained by this Makefile, so use nmake -f partview.mak clean if you change anything.

To compile with MinGW and company, you’ll need to:

1. Install MinGW (gcc, etc.), its associated w32api libraries and header files, and the MSYS suite of UNIX-like tools. All three packages are available at: http://www.sourceforge.net/projects/mingw/

   Unpack the .zip or .tar archives of MinGW and w32api; both packages are intended to live in the same directory. The MSYS package comes as a self-extracting archive and can be extracted into a different directory as desired. (But don’t attempt to merge the MSYS bin directory contents into mingw/bin.)

2. Add both the MSYS bin subdirectory and MinGW bin subdirectory to the Windows PATH environment variable, with the MSYS directory coming earlier, e.g. in a command window

    set path=%path%;C:\util\msys\1.0\bin;C:\util\mingw\bin

   or the analogous setting of PATH using (on WinNT/2000 at least) My Computer -> Control Panel -> System -> Environment to make a permanent change to PATH.

3. Use MinGW to build FLTK. (FLTK 1.1.x, e.g. 1.1.9, works with partview. FLTK 2.0 won’t work. Not sure about versions in between.) Unpack the FLTK source distribution and say

    sh configure

    make
4. Build the Starlab libraries, if desired:

(a) You may need to install CVS for Windows. Binary packages are available; follow the Win32 link on http://www.cvs.home.org/downloads.html. Put the resulting cvs.exe file into the PATH somewhere.

(b) Use CVS to checkout the Starlab sources into some directory:

    cd C:\some\where
    set CVSROOT=:pserver:anonymous@cvs.astro.umd.edu:/home/cvsroot
    cvs login
    cvs checkout starlab
    cd starlab

(c) Copy templates\starlab_setup.bat to local\starlab_setup.bat, and edit it. Change the first two set commands: set STARLAB_PATH to the installation directory – in the above example, set STARLAB_PATH=C:\some\where\starlab. Also optionally update (or remove) set PATH=... to add MSYS and MinGW bin directories to it.

(d) From a Windows command window, type

    local\starlab_setup
    make libs

(e) If successful, you should find in the lib directory the files libdstar.a libdyn.a libnode.a librdc.a libstar.a libstd.a libtdyn.a

5. Now, back in the partiview/src directory, use configure and make as under Unix. The MSYS package imposes its own UNIX-like syntax for Windows pathnames, which you’ll need to use as arguments to configure and friends, with forward- instead of backslashes and a /drive-letter prefix. Also, if typing to a Windows command-window, shell scripts like configure must be explicitly fed to sh. Thus for example if FLTK is installed in C:\util\fltk-1.1.9 and Starlab is in F:\src\starlab, then you might build partiview by typing

    sh configure --with-fltk=c:/util/fltk-1.1.9 --with-kira=/f/src/starlab
    make

Note there’s no need to specify the location of the OpenGL or other libraries; the configure script and MinGW tools already know where to find those. Omit the "--with-kira=..." if you’re not using Starlab.

2 Directory structure

Here is the directory structure:

partiview/  root directory
partiview/src  source code
partiview/data  sample datafiles (e.g. Hipparcos Bright Star Catalogue)
partiview/doc  manual (sgml, and derived html, txt, ps/dvi)
partiview/scripts  various useful scripts (calculator, moviemaker, etc.)
partiview/nemo  NEMO specific converters/code
partiview/starlab  STARLAB specific converters/code
partiview/tutor  examples of tutorial type code (added in 0.2)
3 Running the program

First we describe a simple example how to run partiview with a supplied sample dataset. Then we describe the different windows that partiview is made up of, and the different commands and keystrokes it listens to.

3.1 Example 1: Hipparcos Bright Star Catalogue 3-D viewing

Start the program using one of the sample "speck" files in the data directory:

```
% cd partiview/data
% ./hipbright
or
% partiview hipbright
```

and this should come up with a display familiar to most of us who watch the skies. You should probably enlarge the window a bit. Mine comes up in roughly a 300 by 300 display window, which may be a bit small (certainly on my screen ;-) (Hint: the .partiviewrc file may contain commands like eval winsize 600 400.)

Hit the TAB key to bring focus to the (one line) command window inbetween the log screen (top) and viewing screen (bottom). Type the commands

```
fov 50
jump 0 0 0 80 70 60
```

(field of view 50 degrees)

(put yourself in the origin and look at euler angles

\[ RxRyRz (80,70,60) \]

and it should give another nice comfy view :-) If you ever get lost, and this is not hard, use the jump command to go back to a known position and/or viewing angle.

Note that spatial units for this dataset are parsecs, though angular units are degrees for any data in partiview.

Now play with the display, use the 't', 'r', 'f' and 'o' keys (keys are case sensitive) in the viewing window and use the left and mouse buttons down to (carefully) move around a bit, and make yourself comfortable with moving around. Using the 't' button you get some idea of the distance of the stars by moving back and forth a little (the parallax trick). In fact, if you 't' around a little bit, you may see a green line flashing through the display. This is one of the RGB (xyz) axes attached to the (0,0,0) [our sun] position. You should see Procyn and Sirius exhibit pretty large parallaxes, but Orion is pretty steady since it is several hundred parsecs away. If you move the right mouse button you will zoom in/out and should see our Sun flash by with the red-green-blue axes.

The RGB axes represent the XYZ axes in a (right-handed) cartesian system. For the Hipparcos data the X (red) axis points to RA=0h, Y (green) axis to RA=6h, both in the equatorial plane, and the Z (blue) axis points to the equatorial north pole.

Try and use the middle mouse button (or the 'p' key) to click on Sirius or Procyn, and see if you can get it to view its properties. Now use the 'P' key to switch center to rotation to that star. Sirius is probably a good choice. Move around a bit, and try and get the sun and orion in the same view :-)
[NOTE: these Hipparcos data do not have reliably distance above 100-200 pc, so Orion’s individual distances are probably uncertain to 30%]

A little bit on the types of motion, and what the mouse buttons do

<table>
<thead>
<tr>
<th></th>
<th>left Button-1</th>
<th>middle Button-2</th>
<th>right Button-3</th>
<th>Shift Button-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>f (fly)</td>
<td>fly</td>
<td>’pick’</td>
<td>zoom</td>
<td></td>
</tr>
<tr>
<td>o (orbit)</td>
<td>orbit</td>
<td>’pick’</td>
<td>zoom</td>
<td></td>
</tr>
<tr>
<td>r (rotate)</td>
<td>rotate X/Y</td>
<td>’pick’</td>
<td>rotate Z</td>
<td>translate</td>
</tr>
<tr>
<td>t (translate)</td>
<td>translate</td>
<td>’pick’</td>
<td>zoom</td>
<td></td>
</tr>
</tbody>
</table>

The point of origin for rotations can be changed with the ’P’ button. First you can try and pick (’p’ or Button-2) a point, and if found, hit ’P’ to make this point the new rotation center default.

red  = X axis  
green = Y axis  
blue  = Z axis

To choose an arbitrary center of rotation, use the center command.

3.2 Top Row

The top row contains some shortcuts to some frequently used commands. From left to right, it should show the following buttons:

More

Offers some mode switches as toggles: inertia for continues spin or motion, and an H-R Diagram to invoke a separate H-R diagram window for datasets that support stellar evolution.

[g1]

Pulldown g1, g2, ... (or whichever group) is the currently selected group. See object command to make aliases which group is defined to what object. If multiple groups are defined, the next row below this contains a list of all the groups, and their aliases, so you can toggle them to be displayed.

[f]ly

Pulldown to select fly/orbit/rot/tran, which can also be activate by pressing the f/o/r/t keys inside the viewing window.

point

Toggle to turn the points on/off. See also the points command.

poly

Toggle to turn polygons on/off. See also the polygon command.

lab

Toggle to turn labels on/off. See also the label command.
tex
Toggle to turn textures on/off. See also the texture command.

box
Toggle to turn boxes on/off. See also the boxes command.

# # # # #
The current displayed value of the logslum lum slider (see next)

logslum lum
Slider controlling the logarithm of the datavar variable selected as luminosity (with the lum command).

3.3 Group row (optional)
When more than one group has been activated (groups of particles or objects can have their own display properties, and be turned on and off at will), a new Group Row will appear as the 2nd row. Left-clicking (button 1) on a button toggles the display of that group; right-clicking (button 3) enables display of the group, and also selects it as the current group for GUI controls and text commands.

3.4 Time Animation rows (Optional)
For time-dependent data, the third and fourth row from the top control the currently displayed data-time. This time-control bar is only visible when the object has a nonzero time range.

T
Shows the current time (or offset from the tripmeter). The absolute time is the sum of the T and + fields. Both are editable. See also the step control command.

trip
Press to mark a reference point in time. The T field becomes zero, and the + field (below) is set to current time. As time passes, T shows the offset from this reference time.

back
Press to return to reference time (sets T to 0).

+
Current last time where tripmeter was set. You can reset to the first frame with the command step 0

dial
Drag to adjust the current time. Sensitivity depends on the speed setting; dragging by one dial-width corresponds to 0.1 wall-clock second of animation, i.e. 0.1 * speed in data time units.

<

>|
Step time backwards or forwards by 0.1 * speed data time units. See also the < and > keyboard shortcuts.
**3.5 Camera (path) Animation row**

The fifth (or 4th or 3rd, depending if Group and/or Time rows are present) row from the top controls loading and playing sequences of moving through space.

**Path...**

Brings up a filebrowser to load a .wf path file. This is a file with on each line 7 numbers: xyz location, RxRyRz viewing direction, and FOV (field of view). The rdata command loads such path files too.

**Play**

Play the viewpoint along the currently loaded path, as the play command does. Right-click for a menu of play-speed options.

\[<<< [###] >>>\]

Step through camera-path frames. See also frame control command.

**slider**

Slides through camera path, and displays current frame.

**3.6 Logfile window**

The third window from the top contains a logfile of past commands and responses to them, and can be resized by dragging the bar between command window and viewing window. The Logfile window also has a scroll bar on the left. You can direct the mouse to any previous command, and it will show up in the command window. Using the arrow keys this command can then be edited.

**3.7 Command window**

The Command window is a single line entry window, in which Control Commands can be given. Their responses appear in the Logfile window and on the originating console. (Unlike Data Commands, which show no feedback). You can still give Data Commands in this window by prefixing them with the add command. The Up- and Down-arrow keys (not those on the keypad) scroll through previous commands, and can be edited using the arrow keys and a subset of the emacs control characters.
3.8 Viewing window

The (OpenGL) Viewing window is where all the action occurs. Typically this is where you give single
keystroke commands and/or move the mouse for an interactive view of the data. It can be resized two
ways: either by resizing the master window, or by picking up the separator between Viewing window and
Command window above.

3.9 Example 2: a (starlab) animation

Setting up a small animation in for example Starlab can be done quite simply as follows: (see also the
primbim16.mk makefile to create a standard one):

```
% makeplummer -i -n 20 | makemass -l 0.5 -u 10.0 | scale -s | kira -d 2 -D x10 > run1
% partiview run1.cf
% cat run1.cf

kira run1
eval every
eval lum mass 0 0.01
eval psize 100
eval cmnt 1 1.7 .3
eval color clump exact
```

Alternatively, if you had started up partiview without any arguments, the following Control Command (see
below) would have done the same

```
read run1.cf
```

3.10 Example 3: stereo viewing

The 's' key within the viewing window toggles stereo viewing. By default each object is split in a blue and a
red part, that should be viewed with a pair of red(left)/blue(right) glasses. Red/green glasses will probably
work too. Cross-eyed viewing is also available if selected by stereo cross. See stereo and focallen in the
View Commands section.

3.11 Example 4: subsetting

In the data directory, run

```
partiview hip.cf
```

One of the data fields for these stars is the $B-V$ color, colorb_v, abbreviatable to just color. Look at just
the bluest stars: try

```
thresh color < -.1
```
Back off to a large distance (drag with right mouse button, and drag the \texttt{logslum} slider to brighten) and look at the distribution of these blue stars. The Orion spiral-arm spur, extending generally along the \(+Y\) (green) axis, has lots of them. Now look at more reddish stars, those with \(0.5 \leq B-V \leq 1.5\), with:

\begin{verbatim}
thresh color \texttt{.5 1.5}
\end{verbatim}

These are much more uniformly distributed in the galactic plane. Return to seeing all stars, with:

\begin{verbatim}
see all
\end{verbatim}

or re-view the threshold-selected subset (reddish stars) with

\begin{verbatim}
see thresh
\end{verbatim}

or its complement with

\begin{verbatim}
see -thresh
\end{verbatim}

4 Commands

There are two types of commands in \texttt{partview}: Control Commands and Data Commands. Probably the most visible difference between the two is that every Control Command returns feedback to the user, whereas Data Commands are interpreted without comment unless an error occurs.

Some situations, e.g. the command-entry text box, expect to receive Control Commands; others, e.g. files (cf, speck, etc.) named on the command line or specified by \texttt{read} or \texttt{include} commands, are expected to contain Data Commands.

However, it is always possible to enter a Data Command where a Control Command is expected, using the \texttt{add} command prefix, e.g. you could type in the text box:

\begin{verbatim}
add 0 0 0 text The Origin
\end{verbatim}

. Likewise, a Control Command may be given where data is expected, as in a data or .cf file, using the \texttt{eval} prefix:

\begin{verbatim}
1 0 0 text X=1
    eval bgcolor 0.3 0.2 0.1
\end{verbatim}

See also the previous \texttt{starlab} example.

4.1 Control Commands

(see \texttt{partibrains.c::specks\_parse\_args})

Control Commands are accepted in the Command window, and in some other contexts. Generally, \texttt{partview} gives a response to every Control Command, reporting the (possibly changed) status.

Typically, if parameters are omitted, the current state is reported.

Some commands apply to particles in the current group (see Object group commands); others affect global things, such as time or display settings.

Data Commands can also be given, if prefixed with \texttt{add}. 


4.2 I/O Control Commands

**read specks-file**

Read a file containing Data Commands (typical suffix .cf or .speck).

**async unix-command**

Run an arbitrary unix command (invoked via /bin/sh) as a subprocess of partiview. Its standard output is interpreted as a stream of control commands. Thus partiview can be driven externally, e.g. to record an animation (using the snapshot command), or to provide additional GUI controls. Several async commands can run concurrently. Examples are given later. Warning: you cannot interrupt a started command, short of hitting ESC to exit partiview.

**add data-command**

Enter a Data Command where a Control Command is expected, e.g. in the text input box. For example,

```
add 10 15 -1 text blah
```

adds a new label "blah" at 10 15 -1, or

```
add kira myrun.out
```

loads a kira (starlab) output file.

**eval control-command**

Processes that control command just as if the eval prefix weren’t there. Provided for symmetry: wherever either a control command or a data command is expected, entering eval control-command ensures that it’s taken as a control command.

**add filepath (data-command)**

Determines the list of directories where all data files, color maps, etc. are sought. See the filepath entry under Data Commands.

4.3 Object Group Control Commands

Partiview can load multiple groups of particles, each with independent display settings, colormaps, etc. When more than one group is loaded, the Group Row appears on the GUI, with one toggle-button for each group. Toggling the button turns display of that group on or off. Right-clicking turns the group unconditionally on, and selects that group as the current one for other GUI controls.

Many Control Commands apply to the currently selected group.

Groups always have names of the form gN for some small positive N; each group may also have an alias.

**gN**

Select group gN. Create a new group if it doesn’t already exist.

**gN=alias**

Assign name alias to group gN. Note there must be no blanks around the = sign.
object objectname

Likewise, select object objectname, which may be either an alias name or gN.

gN control-command

type: example

object objectname control-command

Either form may be used as a prefix to any control command to act on the specified group, e.g. object fred poly on

gall control-command

Invoke the given control-command in all groups. For example, to turn display of group 3 on and all others off, use:

    gall off
    g3 on

on

either one will enable the display of the currently selected group (as it is by default).
	off

disable

Either one will turn off the display of the current group.

4.4 View Control commands

View commands affect the view; they aren’t specific to data groups.

fov float

Angular field of view (in degrees) in Y-direction.

cen[ter] X Y Z [RADIUS]

Set point of interest. This is the center of rotation in [o]rbit and [r]otate modes. Also, in [o]rbit mode, translation speed is proportional to the viewer’s distance from this point. The optional RADIUS (also set by censize) determines the size of the marker crosshair, initially 1 unit.

cen[ter] [X Y Z [RADIUS]] int[erest] [X Y Z [RADIUS]]

Set point of interest. This is the center of rotation in [o]rbit and [r]otate modes. And, in [o]rbit mode, translation speed is proportional to the viewer’s distance from this point. The optional RADIUS (also set by censize) determines the size of the marker crosshair, initially 1 unit.

    * * * * why is center/interest commented out in the first example. Originally this command was documented twice, the first one has /interest commented out.

censize [RADIUS]

Set size of point-of-interest marker.
where *(also)* \texttt{w}

Report the 3-D camera position and forward direction vector.

**clip NEAR FAR**

Clipping distances. The computer graphics setup always requires drawing only objects in some finite range of distances in front of the viewpoint. Both values must be strictly positive, and their ratio is limited; depending on the graphics system in use, distant objects may appear to blink if the \texttt{FAR/NEAR} ratio exceeds 10000 or so.

To set the far clip range without changing the near, use a non-numeric near clip value, e.g. \texttt{clip -1000}.

**jump \[X \ Y \ Z\] \[R_x \ R_y \ R_z\]**

Get or set the current position (XYZ) and/or viewing (RxRyRz) angle.

**readpath**

Read a Wavefront (.\texttt{.w}) file describing a path through space.

**rdata**

Synonym for readpath.

**play speed[f]**

Play the currently loaded (from readpath/rdata) camera animation path, at \texttt{speed} times normal speed, skipping frames as needed to keep up with wall-clock time. (Normal speed is 30 frames per second.) With \texttt{"f"} suffix, displays every \texttt{speed-th} frame, without regard to real time.

**frame \[frameno\]**

Get or set the current frame the \texttt{frameno-th}.

**update**

Ensures the display is updated, as before taking a snapshot. Probably only useful in a stream of control commands from an \texttt{async} subprocess.

**winsize \[XSIZE \ [YSIZE]\]**

**winsize XSIZE\texttt{YSIZE+XPOS+YPOS}**

Resize graphics window. With no arguments, reports current size. With one argument, resizes to given width, preserving aspect ratio. With two arguments, resizes window to that height and width. With complete X geometry specification (no embedded spaces), e.g. \texttt{winsize 400x350+20-10}, also sets position of graphics window, with +X and +Y measured from left/top, -X and -Y measured from right/bottom of screen.

**detach [full|hide] \[+XPOS+YPOS\]**

Detach graphics window from GUI control strip and optionally specify position of control strip. With \texttt{full} or \texttt{hide}, makes graphics window full-screen with GUI visible or hidden, respectively. With neither \texttt{full} nor \texttt{hide}, the graphics window is detached but left at its current size.

The \texttt{+XPOS+YPOS} is a window position in X window geometry style, so e.g. \texttt{detach full -10+5} places the GUI near the upper right corner of the screen, 10 pixels in from the right and 5 pixels down from the top edge.
If you don’t mind typing blindly, it’s still possible to enter text-box commands even with the controls hidden; press the Tab key before each command to ensure that input focus is in the text box. Use Tab detach full Enter to un-hide a hidden control strip.

bgcolor R G B

Set window background color (three R G B numbers or one grayscale value).

focallen distance

Focal length: distance from viewer to a typical object of interest. This affects stereo display (see below) and navigation: the speed of motion in ttranslate and ffly modes is proportional to this distance.

stereo [on|off|red|cyan|glasses|cross|left|right] [separation]

Stereo display. Also toggled on/off by typing 's' key in graphics window. Where hardware allows it, stereo glasses selects CrystalEyes-style quad-buffered stereo. All systems should be capable of stereo red/cyan, which requires wearing red/green or red/blue glasses, and of cross (crosseyed), which splits the window horizontally. left and right show just that eye’s view, and may be handy for taking stereo snapshots.

Useful separation values might be 0.02 to 0.1, or -0.02 to -0.1 to swap eyes. See also focallen command, which gives 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 distance 2 * focallen * separation, with convergence angle 2 * arctan(separation).

See also the winsize and detach commands for control over graphics window size and placement.

Beware: some systems which support hardware ("glasses") stereo also require that the display be set to a stereo-capable video mode. Partiview does not do this automatically. For example, on stereo-capable SGI Irix systems, you may need to type (to a unix shell) /usr/gfx/setmon -n 1024x768_96s to allow stereo viewing and something like /usr/gfx/setmon -n 72 to revert. Otherwise, turning partiview’s stereo on will just show the left eye’s view – displacing the viewpoint but nothing else.

snapshot [-n FRAMENO] FILESTEM [FRAMENO]

Set parameters for future snapshot commands. FILESTEM may be a printf format string with frame number as argument, e.g. snapshot pix/%04d.ppm, generating image names of pix/0000.ppm, pix/0001.ppm, etc. If FILESTEM contains no % sign, then .%03d.ppm.gz is appended to it, so snapshot ./pix/fred yields snapshot images named ./pix/fred.000.ppm.gz etc.

Frame number FRAMENO (default 0) increments with each snapshot taken.

snapshot [FRAMENO | FILENAME]

Capture a snapshot image of the current view.

Either give snapshot an explicit filename, or else specify a file format string with snapshot and then let snapshot fill in the frame number. With neither FRAMENO nor FILENAME, snapshot adds one to the previous frame number.

Partiview generally invokes the ImageMagick program convert(1), which must be installed and be on the user’s $PATH. Convert determines the type of image (jpeg, sgi, bmp, etc.) based on the file suffix.

Convert is not needed if the snapshot FILESTEM ends in .ppm.gz (invokes gzip rather than convert) or .ppm (no external program required).
4.5 Particle Display Control Commands

These commands affect how particles (in the current group) are displayed.

**psize scalefactor**

All particle luminosities (as specified by `lum` command) are scaled by the product of two factors: a `lumvar`-specific factor given by `slum`, and a global factor given by `psize`. So the intrinsic brightness of a particle is \( \text{value-specified-by-lum} \times \text{slum-for-current-lumvar} \times \text{psize-scalefactor} \).

**slum slumfactor**

Data-field specific luminosity scale factor, for current choice of `lumvar` as given by the `lum` command. A `slumfactor` is recorded independently for each data field, so if data fields `mass` and `energy` were defined, one might say

```
  lum mass
  slum 1000
  lum energy
  slum 0.25
```

having chosen each variable’s `slumfactor` for useful display, and then freely switch between `lum mass` and `lum energy` without having to readjust particle brightness each time.

**ptsz minpixels maxpixels**

Specifies the range of apparent sizes of points, in pixels. Typical values might be `ptsz 0.1 5`. The graphics system may silently impose an upper limit of about 10 pixels.

**poly [on|off]**

Display polygons, or don’t.

**polysize [scalefactor]**

Multiplier for polygon size. Default is zero (!), so you must set polysize to something else before polygons will show up.

**polylumvar [attrname | point-size] [area | radius]**

Choose which attribute determines the radius of a particle’s polygon. By default, it is `point-size`, a pseudo-attribute which varies with the brightness of points (so adjusting the slum slider scales polygons too).

Each polygon’s 3-D radius is the `polysize scalefactor` times its particle’s given attribute (whether an actual particle attribute or `point-size`). Or, if the `area` keyword is specified, then the radius is the square root of attribute \( \times \) scalefactor. `area` is useful if the attribute represents a luminosity; in that case, the polygon total brightness (which is proportional to its screen area) becomes proportional to the attribute / distance\(^2\).

**polymin minradius [maxradius]**

Specify a minimum screen radius for polygons, in pixels. If smaller than this, they are not drawn.

**color**

Specify how particles are colored. Generally, a linear function of some data field of each particle becomes an index into a colormap (see `cmap`, `cment`).


color colorvar [minval maxval]
    Use data field colorvar (either a name as set by datavar or a 0-based integer column number) to determine color. Map minval to color index 1, and maxval to the next-to-last entry in the colormap (Nmap-2). The 0th and last (Nmap-1) colormap entry are used for out-of-range data values.
    If minval and maxval are omitted, the actual range of values is used.

color colorvar exact [baseval]
    Don’t consider field colorvar as a continuous variable; instead, it’s integer-valued, and mapped one-to-one with color table slots. Data value N is mapped to color index N+baseval.

color colorvar -exact
    Once the exact tag is set (for a particular data-field), it’s sticky. To interpret that data field as a continuous, scalable variable again, use -exact.

color const R G B
    Show all particles as color R G B, each value in range 0 to 1, independent of any data fields.

lum
    Specify how particles’ intrinsic luminosity is computed: a linear function of some data field of each particle.

lum lumvar [minval maxval]
    Map values of data field lumvar (datavar name or field number) to luminosity. The (linear) mapping takes field value minval to luminosity 0 and maxval to luminosity 1.0.
    If minval and maxval are omitted, the actual range of values is mapped to the luminosity range 0 to 1.
    Note that the resulting luminosities are then scaled by the psise and slum scale factors, and further scaled according to distance as specified by fade, to compute apparent brightness of points.

lum const L
    Specify constant particle luminosity L independent of any data field values.

fade [planar|spherical|linear refdist|const refdist]
    Determines how distance affects particles’ apparent brightness (or "size"). The default fade planar gives 1/r^2 light falloff, with r measured as distance from the view plane. fade spherical is also 1/r^2, but with r measured as true distance from the viewpoint. fade linear refdist gives 1/r light falloff - not physically accurate, but useful to get a limited sense of depth. fade const refdist gives constant apparent brightness independent of distance, and may be appropriate for orthographic views.
    The refdist for linear and const modes is that distance r at which apparent brightness should match that in the 1/r^2 modes - a distance to a "typical" particle.

labelmin minpixels
    Labels computed to be smaller than this screen size (pixels) are suppressed.

labels size scalefactor

lsize scalefactor
    lsize (alias labelsize) sets the 3-D height of labels. If the text was created with a text -size textsize option, the scalefactor is multiplied by that to determine the 3-D size.
point[s] [on|off]
Turn display of points on or off. With no argument, toggles display.

poly[gons] [on|off]
Turn display of points on or off. With no argument, toggles display.

texture [on|off]
Turn display of textures on or off. With no argument, toggles.

label[s] [on|off]
Turn display of label text on or off. With no argument, toggles.

taxscale  scale factor
Scale size of all textures relative to their polygons. A scale factor of 0.5 (default) make the texture square just fill its polygon, if polysides is 4.

polyorivar
Report setting of polyorivar data-command, which see.

texturevar
Report setting of texturevar data-command, which see.

laxes [on|off]
Toggle label axes. When on, and when labels are displayed, shows a set of red/green/blue (X/Y/Z) axes to indicate orientation.

polysides(s)
Number of sides a polygon should have. Default 11, for fairly round polygons. For textured polygons, polysides 4 might do as well, and be slightly speedier.

fast [on|off]
see also psize

ptsize  minpixels [maxpixels]
Specifies range of apparent (pixel) size of points. Those with computed sizes (based on luminosity and distance) smaller than minpixels are randomly (but repeatably) subsampled - i.e. some fraction of them are not drawn. Those computed to be larger than maxpixels are drawn at size maxpixels.

gamma displaygamma
Tells the particle renderer how the display + OpenGL relates image values to visible lightness. You don't need to change this, but may adjust it to minimize the brightness glitches when particles change size. Typical values are gamma 1 through gamma 2.5 or so. Larger values raise the apparent brightness of dim things.

alpha alpha
Get or set the alpha value, in the range 0 to 1; it determines the opacity of polygons.

speed
For time-dependent data, advance datatime by this many time units per wall-clock second.
step [timestep]

For time-varying data, sets current timestep number. Real-valued times are meaningful for some kinds of data including those from Starlab/kira; for others, times are rounded to nearest integer. If running, step also stops datetime animation. (See run.)

step [+-]deltatimestep

If preceded with a plus or minus sign, adds that amount to current time.

(note that fspread has been deprecated)

run

Continue a stopped animation (see also step).

tfm [-v] [numbers...]

Object-to-world transformation. May take 1, 6, 7, 9 or 16 parameters: either scalefactor, or tx ty tz rx ry rz scalefactor>, or 16 numbers for 4x4 matrix, or 9 numbers for 3x3 matrix. See Coordinates and Coordinate Transformations.

With no numeric parameters, reports the current object-to-world transform. Use tfm -v to see the transform and its inverse in several forms.

move [gN] {on/off}

Normally, navigation modes [r]otate and [t]ranslate just adjust the viewpoint (camera). However, if you turn move on, then [r]otate and [t]ranslate move the currently-selected object group instead, e.g. to adjust its alignment relative to other groups. ([o]rbit and [f]ly modes always move the camera.)

To indicate that move mode is enabled, the control strip shows the selected group's name in bold italics, as [g3]. Use move off to revert to normal. The tfm command reports the current object-group-to-global-world transformation.

fwd

datawait on|off

For asynchronously-loaded data (currently only iee data command), say whether wait for current data step to be loaded. (If not, then keep displaying previous data while loading new.)

cmap filename

Load (ascii) filename with RGB values, for coloring particles. The color command selects which data field is mapped to color index and how.

Colormaps are text files, beginning with a number-of-entries line and followed by R G B or R G B A entries one per line; see the Colormaps section.

vcmap -v fieldname filename

Load colormap as with cmap command. But use this colormap only when the given data field is selected for coloring. Thus the cmap color map applies to all data fields for which no vcmap has ever been specified.

cmzent colorindex [R G B]

Report or set that colormap entry.
rawdump dump-filename

All particle attributes (not positions though) are written to a dump-filename. Useful for debugging. Warning: it will happily overwrite an existing file with that name.

warp [on/off]

Enable, disable, or report the status of any warp data-command set up for the current group. If it exists, particles’s positions can change with time, in a handful of canned ways built into the warp command. See the warp entry under Data Commands.

4.6 Particle subsetting & statistics

clipbox ...

see cb below.

cb ....

Display only a 3D subregion of the data – the part lying within the clipbox.

   cb xmin ymin zmin xmax ymax zmax

   Specified by coordinate ranges. Note only spaces are used to separate the 6 numbers.

   cb xcen, ycen, zcen xrad, yrad, zrad

   Specified by center and "radius" of the box. Note no spaces after the commas!

   cb xmin, xmax ymin, ymax zmin, zmax

   Specified by coordinate ranges.

   cb off

   Disable clipping. The entire dataset is again visible.

   cb on

   Re-enable a previously defined clipbox setting. It will also display the clipbox again

   cb hide

   Hide the clipbox, but still discard objects whose centers lie outside it.

Note this command does not toggle clipping if no arguments given (that would be handy and more in line with similar commands). If no arguments given, it reports the current clipbox.

thresh

Display a subset of particles, chosen by the value of some data field. Each thresh command overrides settings from previous commands, so it cannot be used to show unions or intersections of multiple criteria. For that, see the only command. However, unlike only, the thresh criterion applies to time-varying data.

thresh field minval maxval

   Display only those particles where minval <= field <= maxval. The field may be given by name (as from datavar) or by field number.

thresh field < maxval

thresh field > minval

   Show only particles where field is <= or >= the given threshold.
thresh [off/on]
Disable or re-enable a previously specified threshold.

only= datafield value minvalue-maxvalue <value >value ...

only+ datafield value minvalue-maxvalue <value >value ...

only- datafield value minvalue-maxvalue <value >value ...
Scans particles (in the current timestep only!), finding those where datafield has value value, or has a value in range minvalue <= value <= maxvalue, or whatever. Multiple value-ranges may be specified to select the union of several sets. The resulting set of particles is assigned to (only=), added to (only+) or subtracted from (only-) the thresh selection-set. Also display just particles in that selection-set, as if see thresh had been typed.

The net effect is illustrated by these examples:

only= type 1-3 5
Show only particles of type 1, 2, 3 or 5.

only- mass <2.3 >3.5
After the above command, shows only the subset of type 1/2/3/5 particles AND have mass between 2.3 and 3.5. (Note that to take the intersection of two conditions, you must subtract the complement of the latter one. Maybe some day there’ll be an only&.

see selexpr
Show just those particles in the selection-set selexpr. Predefined set names are all, none, thresh and pick, and other names may be defined by the sel command. The default is see all. Using the thresh or only commands automatically switch to displaying see thresh.

Note that you can see the complement of a named set, e.g. all except the thresh-selected objects, with see -thresh.

sel selname = selexpr
Compute a logical combination of selection-sets and assign them to another such set. The set membership is originally assigned by thresh or only commands. Yeah, I know this doesn’t make sense. Need a separate section to document selection-sets.

sel selexpr
Count the number of particles in the selection-set selexpr.

clearobj
Erase all particles in this group. Useful for reloading on the fly.

every N
Display a random subset (every N-th) of all particles. E.g. every 1 shows all particles, every 2 shows about half of them. Reports current subsampling factor, and the current total number of particles.

hist datafield [-n nbuckets] [-l] [-c] [-t] [minval] [maxval]
Generates a (numerical) histogram of values of datafield, which may be a named field (as from datavar) or a field index. Divides the value range (either minval..maxval or the actual range of values for that field) into nbuckets equal buckets (11 by default). Uses logarithmically-spaced intervals if -l (so long
as the data range doesn't include zero). If a clipbox is defined, use -c to count only particles within it. If a threshold or only subset is defined, use -t to count only the chosen subset.

bound [w]
Reports 3D extent of the data. With w, reports it in world coordinates, otherwise in object coordinates.

datavar
dv
Report names and value ranges (over all particles in current group) of all named data fields.

4.7 Boxes

showbox list of integer box level numbers...
hidebox list of integer box level numbers...
box[es] [off|on|only]
Turn box display off or on; or display boxes but hide all particles.

boxcolor map filename
Color boxes using that colormap. Each box's level number (set by -1 option of box data-command, default 0) is the color index.

boxcm [colorindex] [R G B]
Get or set the given box-colormap index. E.g. boxcm 0 reports the color of boxes created with no -1 specified.

boxlabel [on|off]
Label boxes by id number (set by -n option of box data-command).

boxaxes [on|off]
Toggle or set box axes display mode.

boxscale [float] [on|off]
gobox boxnumber
goboxscale
menu finenu

BEGIN CAVMENU
pos P1 P2
wall P1
hid [P1]
show [P1]
h [P1]
demandfps [P1]
font
4.8 Data commands

(see also partibrains.c::specks_read)

Data Commands can be placed in a data file. Lines starting with # will be skipped.

Control Commands can also be given, if prefixed with the eval command.

read file

read a speck formatted file. Recursive, commands can nest. (strtok ok??) Note that read is also a
Control Command, doing exactly the same thing.

include file

read a speck formatted file.

ieee [-t time] file

Read a IEEEIO formatted file, with optional timestep number (0 based). Support for this type of data
must be explicitly compiled into the program.

kira file

read a kira formatted file. See the kiract1 Control Command to modify the looks of the objects.
Only present if Starlab is compiled into partview.

setenv name value

Add (or change) a named variable of the environment variables space of partview. Enviroment vari-
ables, like in the normal unix shell, can be referred to by prepending their name with a $. Note there
probably is not an unsetenv command.

object gN=ALIAS

Defines/Selects a particular group number (N=1,2,3,...) to an ALIAS. In command mode you can use
gN=ALIAS. Any data following this command will now belong to this group.

object ObjectName

Select an existing group. Following data will now belong to this group.

sdbvars var

Choose which data fields to extract from binary sdb files (any of: mMcrogtxyzSn) for subsequent sdb
commands.

sdb [-t time] file

Read an SDB (binary) formatted file, with optional timestep number. (Default time is latest datetime,
or 0.)
pb [-t time] file

Read a .pb (binary) particle file, with optional timestep number. (Default time is latest datetime, or 0.) A .pb file contains (all values 32-bit integer or 32-bit IEEE float):

1. magic number, 0xFFFFFFFF98 (int32)
2. byte offset of first particle (int32)
3. number of attributes (int32)
4. sequence of null-terminated attribute name strings, attributename0 \0 attributename1 \0 ...
5. possibly some pad bytes, enough to reach the specified first-particle file offset
6. sequence of particle records, each (number-of-attributes + 4)*4 bytes long:
   (a) particle-id (int32)
   (b) particle X, Y, Z (3 float32's)
   (c) particle attributes (number-of-attributes float32's)
   ending at the end of the file (i.e. there's no particle-count field).

Either big- or little-endian formats are accepted; the value of the magic number determines endianness of all values in that file.

box[es] ....

Draw a box, using any of the following formats:

xmin ymin zmin xmax ymax zmax
xmin,xmax ymin,ymax zmin,zmax
xcen,ycen,zcen xrad,yrad,zrad

[-t time] [-n boxno] [-l level] xcen,ycen,zcen xrad,yrad,zrad

level determines color.

mesh [-t tzone] [-c colorindex] [-s style]

Draw a quadrilateral mesh, optionally colored or textured. Following the mesh line, provide a line with the mesh dimensions: nu nv

Following this comes the list of nu*nv mesh vertices, one vertex (specified by several blank-separated numbers) per line. (Blank lines and comments may be interspersed among them.) Note that the mesh connections are implicit: vertex number i*nu+j is adjacent to (i-1)*nu+j, (i+1)*nu+j, i*nu+(j-1), and i*nu+(j+1). Each vertex line has three or five numbers: the first three give its 3-D position, and if a -t texture was specified, then two more fields give its u and v texture coordinates.

Options:

-t tzone
  Apply texture number tzone to surface. In this case, each mesh vertex should also include u and v texture coordinates.

-c colorindex
  Color surface with color from integer cmap entry colorindex.

-s style
  Drawing style:
solid
filled polygonal surface (default)

wire
just edges

point
just points (one per mesh vertex)

Xcen Ycen Zcen ellipsoid [options]... [transformation]
Draw an ellipsoid, specified by:

Xcen Ycen Zcen
Center position in world coordinates

-c colorindex
Integer color index (default -1 => white)

-s style
Drawing style:
solid
filled polygonal surface (default)
plane
3 ellipses: XY, XZ, YZ planes
wire
latitude/longitude ellipses
point
point cloud: one per lat/lon intersection

-r Xradius, Yradius, Zradius
Radius (for sphere) or semimajor axes (for ellipsoid)

-n nlat, nlon
Number of latitude and longitude divisions. Relevant even for plane style, where they determine how finely the polygonal curves approximate circles. Default nlon = nlat/2 + 1.

transformation
Sets the spatial orientation of the ellipsoid. May take any of three forms:

(nothing)
If absent, the ellipsoid’s coordinate axes are the same as the world axes for the group it belongs to.

9 blank-separated numbers
A 3x3 transformation matrix T from ellipsoid coordinates to world coordinates, in the sense
Pworld = Pellipsoid * T + [Xcen, Ycen, Zcen].

16 blank-separated numbers
A 4x4 transformation matrix, as above but for the obvious changes.

Load a Wavefront-style .obj model. Material properties are ignored; the surface is drawn in white unless -c colorindex in which case it’s drawn using that color-table color. Also if -texture (alias -tx) is supplied, the surface is textured using whatever texture coordinates are supplied in the .obj file. The
model is displayed at all times only if marked -static; otherwise it’s displayed only at the time given by -time timestep or by the most recent datetime.

A subset of the .obj format is accepted:

\[ \text{v X Y Z} \]
- vertex position

\[ \text{vt U V} \]
- vertex texture coordinates

\[ \text{vn NX NY NZ} \]
- vertex normal

\[ \text{f V1 V2 V3 ...} \]
- face, listing just position indices for each vertex. The first v line in the .obj file has index 1, etc.

\[ \text{f V1/T1 V2/T2 V3/T3 ...} \]
- face, listing position and texture coordinates for each vertex of the face.

\[ \text{f V1/T1/N1 V2/T2/N2 V3/T3/N3 ...} \]
- face, listing position, texture-coordinate, and normal indices for each vertex.

Note that material properties (mtl) are ignored. Waveobj models are colored according to the -c colorindex option (integer index into the current cmap colormap), or white if no -c is used. If texturing is enabled – if the .obj model contains vt entries, and the -texture option appears, and that numbered texture exists – then the given texture color multiplies or replaces the -c color, according to the texture options.

tfm |camera| numbers...

Object-to-world transformation. May take 1, 6, 7, 9 or 16 numbers: either scalefactor or tx ty tz rx ry rz [s/t/scalefactor s/t/1] or 16 numbers for 4x4 matrix, or 9 numbers for 3x3 matrix. See Coordinates and Coordinate Transformations.

Normally the transform is to world coordinates; but with optional camera prefix, the object’s position is specified relative to the camera, useful to place legends in a fixed position on the screen. In camera coordinates, (0,0,0) is the viewpoint, x=y=0 at screen center, and negative z extends forward. Try for example

\[ \text{tfm camera -3 -3 -20 0 0 0} \]
\[ 0 0 0 \text{ text -size 20 Legend} \]

eval command

execute a Control Command.

feed command

Synonym for eval.

VIRDIR command

Synonym for eval.

filepath path

A colon-separated list of directories in which datafiles, color maps, etc. will be searched for. If preceded with the + symbol, this list will be appended to the current filepath.
**polyorivar inxerno**

By default, when polygons are drawn, they're parallel to the screen plane - simple markers for the points. It's sometimes useful to give each polygon a fixed 3-D orientation (as for disk galaxies). To do this, provide 6 consecutive data fields, representing two 3-D orthogonal unit vectors which span the plane of the disk. Then use **polyorivar inxerno** giving the data field number of the first of the 6 fields. The vectors define the X and Y directions on the disk, respectively - relevant if texturing is enabled.

Actually, unit vectors aren't essential; making them different lengths yields non-circular polygonal disks.

If **polyorivar** is specified for the group, but some polygons should still lie in the screen plane, use values 9 9 9 9 9 9 for those polygons.

**texture [-aiAOImnMDB] txno file.sgi**

-**a(lpha)**
  A single-channel image would normally be used as luminance data. With -a, the image is taken as opacity data instead (GL_ALPHA texture format).

-**i(ntensity)**
  For 1- or 3-channel images, compute the intensity of each pixel and use it to form an alpha (opacity) channel.

-**A(dd)**
  Use additive blending. This texture will add to, not obscure, the brightness of whatever lies behind it (i.e. whatever is drawn later).

-**O(ver)**
  Use "over" compositing. This texture will obscure features lying behind it according to alpha values at each point.

-**M(odulate)**
  Multiply texture brightness/color values by the colormap-determined color of each particle.

-**D(ecal)**
  The textured polygon's color is determined entirely by the texture, suppressing any colormapped color.

-**B(lend)**
  Probably not very useful.

**texturevar field**

If polygon-drawing and texturing are turned on, use the given field (datavar name or number) in each particle to select which texture (if any) to draw on its polygon.

**coord name ... 16 world-to-coord tfm floats (GL order)**

**dataset indexno datasetname**

Give names to multiple datasets in IEEEO files (read with iee command). indexno is an integer, 0 being the first dataset.
datavar indexno name [minval maxval]

Name the variable in data field indexno. The first data field has indexno 0. If provided, minval maxval supply the nominal range of that data variable; some control commands (lum, color) need to know the range of data values, and will use this instead of measuring the actual range.

datetime time

Label subsequent data with this time (a non-negative integer).

warp

When 'warp' has been defined for a group, all its particles get their positions (re)computed according to (a) the warp data-command’s parameters, (b) the current time, (c) the particle’s initial position, and (d) maybe some attributes of each particle.

There are several (mutually exclusive) kinds of warping available:

- for doing a sort of differential rotation for disk-like galaxies;
- for doing N-D to 3-D projection, where particle positions are replaced with (time-independent) linear combinations of attribute values;
- linear or polynomial extrapolation of the particle position with time, with coefficients specified as triples of attributes

Options to warp data command:

-p period0[f|s]

"rotation period". Sets timescale of motion, in frames (f) or seconds (s).

-extrap coef0[.degree]

Extrapolate position with time. Velocity is given by attribute coef0 and the two attributes following it (coef0 .. coef0+2), in the sense \( p = p_0 + [\text{coef0} .. \text{coef0+2}] \times (t / \text{period0}) \). If degree given (default 1), uses \( 3^\text{degree} \) attributes as polynomial coefficients, as \( p = p_0 + (t / \text{period0}) \times \text{field} \times [\text{coef0} .. \text{coef0+2}] + (t / \text{period0})^2 \times [\text{coef0+3} .. \text{coef0+5}] + \ldots \)

-sheet ampl,xlength,zlength

For disk galaxy style. Applies exponential sheet warp for disk lying in the X-Z plane. Scale set by xlength and zlength, Y-displacement set by ampl.

-f fin,fout

For disk-galaxy style: gives time range over which warp applies.

-z zerotime

For disk-galaxy style: sets time at which particles are in their original positions.

-R rot[,drot]

Disk galaxy style: Add constant to rotation angle.

-T o2d

Provide object-to-disk coordinate transform (in "disk" coordinates, the disk lies in X-Z plane). 9 or 16 numbers.

-F d2o

Provide disk-to-object transform. 9 or 16 numbers.
-r rcore[,transition][w]
  Disk galaxy style: set radius of rigidly-rotating inner region, and transition to constant-velocity region

-fix x,y,z[w][radius[w]]
  Disk galaxy style: Keep the given 3-D point, or a point at the given disk radius, fixed. E.g. track the sun.

-galaxy gorbcoef0
  Special disk galaxy style. Each star is on its own disk-galaxy-like orbit, with 8 orbital parameters given by 8 consecutive attributes starting with gorbcoef0. See galaxyorbit.h (read the source).

-ride speckno
  Ride along with speckno'th particle in first loaded group (displace particles by the difference between their computed orbit position and the ridden-on particle).

Xpos Ypos Zpos Var0 ....
  These lines, with XYZ positions in the first 3 columns, will make up the bulk of a typical dataset. The 4th and subsequent columns contain the values of the datavariabes as named with the datavar commands. Note that data variable (field) numbers are 0-based.

4.9 Kira/Starlab

To read Kira output, in human-readable or binary tdyn form, use the "kira kirafilename" data-command.

4.9.1 Kira particle attributes

The particles read in have the following attributes:

id
  positive integer worldline index for single stars (matching the id in the kira stream). For non-leaf (center-of-mass) tree nodes, id is a negative integer.

mass
  Mass, in solar mass units (see “kira mscale” control command).

nclump
  Number of stars in this particle’s subtree. 1 for isolated stars, 2 for binaries, etc.

Tlog
  base-10 log of temperature (K)

Lum
  Luminosity in solar-mass units. (Note this is linear, not log luminosity.)

stype
  Stellar type code (small integer). The [bracketed] message reported when picking (button-2 or p key) on a star gives the corresponding human-readable stellar type too.
ismember
Is this star still a member of (bound to) the cluster?

rootid
id of root of subtree. For single stars, rootid = id.

treeaddr
bit-encoded location of star in subtree.
ringsize
0 for stars. For nonleaf nodes, this is the semimajor axis or instantaneous separation (according to "kira sep"). This field isn't multiplied by the scale factor given in kira sep; it gives the actual distance in kira units.
sqrtmass
Square root of mass/Msun. Might be useful for luminosity scaling.

mu

4.9.2 Hertzsprung-Russell diagram

The H-R diagram can be invoked via the More... menu (upper left) or by the kira hrdiag on control command. Axes for this plot are log temperature (initial range from 5 to 3) and log luminosity (initial range -4 to 6). Ranges may be changed with the kira hrdiag range command or with keystrokes.

Keystroke commands in the H-R window:

b/B
Adjust the (b)rightness (dot size) of the dots plotted for each star. Small b brightens (enlarges); capital B shrinks.
a/A
Adjust (a)lpha (opacity) of dots plotted for each star. If many stars coincide in H-R, their brightnesses add. Thus reducing opacity may help clarify the relative L-T space densities, if there are many stars.
v/V
Zoom out (v) or in (V) by 33%. The point under the cursor becomes the center of the view.

4.9.3 kira control commands

Viewing control options for kira/Starlab formatted data that have been read in with the kira Data Command. All control commands begin with kira too.

kira node {on|off|root}
Show or hide center-of-mass nodes for multiple stars. With on, show CM nodes for each level in a binary tree. With root, show only the top-level CM node for each multiple.
kira ring \{on\|off\|root\}
    Show circles around multiple stars; on and root as above.

kira tree \{on\|off\|cross\|tick\} \{tickscale\}
    Show lines connecting pairs of stars at each binary-tree level in a multiple group. With cross, also show a perpendicular line—a tick mark—which crosses at the CM point, and whose length is tickscale (default 0.5) times the true separation of the pair. With tick, just show the tick-mark with no connecting line.

kira size \{sep\|semi\} \{rangescalefactor\}
    Determines 3-D size of circles when kira ring on. With kira size sep, ring diameter is scalefactor * instantaneous separation. With kira size semi, ring radius is scalefactor * a (the semimajor axis of the two-body system, or |a| for hyperbolic orbits). Using semi gives typically more stable-looking rings, though they will pop if they become marginally (un-)bound. Default: kira size semi 1.5.

kira scale \{rangescalefactor\}
    Synonym for kira size above.

kira span minpix maxpix
    Sets screen-space (pixel) size limits on rings. They'll never get smaller than radius minpix nor larger than maxpix, regardless of true 3-D size. Thus even vanishingly tight binaries can always be visibly marked. Default: kira span 2 50.

kira track \id\|on\|off\|
    As particle \id\ moves through time, move the viewpoint in the same way, so that (if you don't move the view by navigation) the particle remains fixed in apparent position. kira track off disables tracking, and kira track on re-enables it. Use the p key or mouse button 2 to pick a particle (or CM node if kira node on) to see its numeric \id\. Transient center-of-mass nodes (shown if kira node on) can be tracked while they exist.

kira mscale massscalefactor[!]\]
    Set/check the mass scale factor. Starlab dynamical mass values are multiplied by this factor for reporting to the user. Normally massscalefactor should equal the initial cluster mass in solar-mass units. For some input files, starlab can determine what was specified in the original kira run. If so, “kira mscale number” will be ignored unless number ends with an exclamation point (!). So with no !, the user (or .cf script) provides a default value; use ! to override the original mass scale.

kira int seldest \[= selsrc\]
    Track interactions between particles. As the cluster evolves, whenever any star matching selection-expression selsrc encounters (is a member of the same kira tree as) another particle, then the other particle is added to the seldest set. If seldest and selsrc are the same (or if “= selsrc” is omitted), then kira int computes the transitive closure of the interaction set. Otherwise, only stars that encounter members of the initial selsrc set become members of the seldest set. Example:

    click on some star
        The clicked-on star(s) become members of the pick set.

    sel x = pick
        Save a copy in the new set named x.
kira int x
Accumulate encounters in the set x.

emph x
Increase brightness of members of x.

kira trail x
Extend trails from these set members.

kira trail seexpression|off
Leave trails behind particles selected by seexpression (see the sel command). As (dynamical) time
passes, for each display update, one sample point is added to the trail for each selected particle. (If
you reverse the direction of time, the trails will fold back on themselves.) Some examples:

kira trail all
Makes trails grow behind all particles (including CM nodes, if they're displayed)

kira trail pick
Clicking on a star will make a trail grow behind it. If several stars are within picking range (under
the cursor), trails will grow behind each of them.

thresh -s big mass > 1.5
threshold when masses are larger than 1.5

kira trail big
These two commands (a) select all stars exceeding 1.5 solar masses and (b) extend trails behind
them.

kira trail clear
Erase current trails, but let them continue to accumulate as time passes.

kira maxtrail nsamples
Set how many time-points are kept for each particle's trail, initially 50.

kira hrdiag on|off
toggle to turn HD Diagram on or off. Initially off.

kira hrdiag range logLeft logRight logLbottom logLtop
set limits on the HD Diagram axes.

4.10 Textures

To make polygons be textured:

- Use a series of texture data-commands to provide a table of textures, each named by a small integer
texture-index;
- Create a data field in each particle whose value is the texture-index for that particle’s polygon
- Use data-command texturevar fieldno to specify which data field that is.
• Use control commands \texttt{(poly, polylumvar, polysize)} to enable drawing polygons and textures, and to give the polygons nonzero size.

• Possibly use control command \texttt{polysides} to specify 4-sided polygons – a bit faster to draw than default 11-gons.

It doesn’t matter whether the texture-index data field is given a datavar name.

For each particle, if the value of its \texttt{texturevar}'th field either (a) doesn’t match the value in some \texttt{texture} command or (b) the file named in that \texttt{texture} command couldn’t be read, then its polygon is drawn as if texturing were disabled.

4.11 Coordinates and Coordinate Transformations

Matrices as for the \texttt{tfm} command are intended to be multiplied by an object-coordinate row vector on the left, so that 4x4 matrices specify a translation in their 13th through 15th entries. Generally they’re in the sense of an object-or-camera-to-world transform.

The six- or seven-number transforms \texttt{(tx ty tz rx ry rz [it/scalefactor/], as accepted by the tfm and jump commands) are interpreted as}

\[ P_{\text{world}} = P_{\text{object}} \ast \text{scalefactor} \ast \text{rotY}(ry) \ast \text{rotX}(rx) \ast \text{rotZ}(rz) \ast \text{translate}(tx, ty, tz) \]

4.12 Colormap Files

Colormap files, as read by the \texttt{cmap} and \texttt{vcmmap} commands, are line-oriented text files. Blank lines are ignored, as are \# comments. The first nonblank, non-comment line gives the colormap \texttt{size} (number of entries). Later lines may have the form

\<it/R G B/>

... giving red, green, and blue, each in the range 0 .. 1. Typically there will be \texttt{size} of these lines. However the colormap need not be written sequentially; a line like

\<it/colorindex\:<it/R G B/>

places that RGB value at that \texttt{colorindex}, in the range 0 .. \texttt{size}-1. Later \texttt{R G B} lines are assigned to \texttt{colorindex+1, colorindex+2} and so on. Also,

\<it/colorindex\:=<it/oldcolorindex/>

copies the (previously-assigned) RGB value from \texttt{oldcolorindex} and assigns it to \texttt{colorindex}.

5 Viewing Window Keyboard Shortcuts

Commands that you can give from within the viewing window are all single keystroke commands, often combined with moving the mouse.
TAB change focus to command window for Control Commands
S/s toggle STEREO mode (need blue/red glasses :-(
  modes: mono red/cyan crosseyed glasses
  See also the 'stereo' View Command
> single frame forward stepping, in time animation mode
< single frame backward stepping, in time animation mode

Button-N various translation/rotation/zoom, depending on mode (fly/orbit/rot/tra)
SHIFT + Button-N modifier to the usual Button-N action, to have more fine control
CTRL + Button-N modifier to orbit-mode, e.g. to do translations instead of rotations

playmodes:
s playnow
l loop (rock)
f,e playevery=1
r,t playevery=0

Gview.cpp : F1_Gview::handle()
cw reset camera position
p identify nearest object under mouse cursor
P pick that object as the new origin
o ORBIT mode
f FLY mode
r ROTATE mode
t TRANSLATE mode
O toggle perspective mode
v make field of view larger
V make field of view smaller
"v toggle debug output
@ report viewpoint position
= show object-to-world, world-to-object 4x4 matrices
  (precede by object name, e.g. "c=", "g3=")
ESC exit

PrintScreen Take image snapshot of current view
< > Step backwards/forwards in dynamical time
  (numeric prefix sets time step)
{ } Animate backwards/forwards in dynamical time
  Fermonic dynamical-time animation toggle:
    cycle between stop/forward/stop/backward/...
Z Halve/double animation speed (dyn units/sec)
  (numeric prefix sets animation speed)
6 Partiview and NEMO

The program `snapspecks` converts a NEMO snapshot to specks format that can be read in directly by partiview. The default viewing variables are x, y, z, m, but you can add and changed them by using the `options=` keyword. In fact, arbitrary `bodytrans` expressions can be used for output. In the following example a 32-body Plummer sphere is created, which is then given a power-law mass spectrum (with slope -2) between 0.5 and 10 mass units, and animated:

```
% mkplummer - 32 \
  snapmass -- massname='n(m)' masspars=p,-2 massrange=0.5,10 \ 
  hackcode1 - run1.dat \
% snapspecks run1.dat > run1.tab
% partiview run1.cf
% cat run1.cf

read run1.tab
eval labels off
eval lum lum 0 1
eval polylumvar point-size .1 area
texturevar 4
eval psize 5000
eval slum 5
eval every 1
```

7 Tips

During animation the trip/back buttons can effectively be used to return to a point in time where you want to return back to if you wanted to browse around some specific point in time.

You can spend most of the time moving in orbit mode. Left-button moves around chosen center; control-left pans around the sky. As opposed to switching to 't' mode to zoom and translate, you can also use SHIFT-Mouse-1 and SHIFT-Mouse-3 to achieve the same from the other ('o', 'f') modes.

To make an animation, create an executable shell script `movie1` with for example the following commands:

```
#!/bin/csh -f
#
# update
# snapshot
# step 0.01
# update
# snapshot
# step 0.02
# update
# snapshot
# step 0.03
# update
# snapshot
```
the Control Command async movie1, and it will create files snap.000.sgi, snap.001.sgi, .... and already with xv a movie can be shown:

```
xv -wait 0 snap.???.sgi
```

To make animated GIFs, here are some examples with common software, all with a default 0.1 sec delay between frames. Some animation software (e.g. xanim) can change these:

```
convert -delay 10 -loop 0 snap.???.sgi try1.gif
gifsicle -d 10 snap.???.gif > try2.gif
```

The script will run asynchronously within partview, so if you then use the mouse to change orientation or zoom, these actions (minus the location of the mouse of course) will be nicely recorded in the snapshots.

8 Bugs, Features and Limitations

Here is a list of known peculiarities, some of them bugs, others just features and others limitations, and there is always that class of things I simply have not understood how it works.

8.1 Limitations w.r.t. VirDir:

1. cannot set an auto-motion, as we can in the dome, although one could of course load a path and move through the dataset ;-) I was able to make a path (*.wf) file and load that though. Now mostly solved via the Inertia toggle under the More button from the Top Row Window.

8.2 Some notes for newcomers to VirDir

Although starting virdir is very similar to partview,

```
% parti gal2.cf
```
or,

```
% virir gal2.cf
```

the seasoned partview user will need to relearn a few modes to get used to virdir. In particular, at AMNH starting virdir will probably cause your console screen (which is normally panel #1 on the dome) to go dark with no visible command prompt. To regain control, type the commands (blindly)

```
raise
fly
idle
```

which will put virdir in fly and animation mode.

Here are some important modes, make sure you keep the mouse in the console window. It is easy to get it lost in any of the other 6 displays which are only visible on the dome.

1. Pushing the Left and Right mouse buttons simultaneously will send the display to the HOME position.
2. Left mouse button will toggle the Pause mode in animate/fly mode.
3. Holding the Ctrl-button down while moving the mouse will bring your point of interest into view.
4. Holding the Alt-button down while moving the mouse will rotate around your point of interest.
5. The 'p' key.
6. The middle mouse button toggles Head display vs. Center display.
7. Holding the Shift-button down while moving the mouse will change the available screen-space (works like a zoom).

9 Glossary

1. group: particles can be grouped with the object command. If multiple groups exist, a separate Group row will be activated automatically.
2. data command, not to be confused with control command
3. control command, not to be confused with data command
4.