MayaVi Users Guide

Prabhu Ramachandran

August 25, 2005

Abstract

MayaVi is a scientific data visualizer. It is written in Python <http://www.python.org> and uses the Visualization Toolkit (VTK) <http://www.vtk.org/> for the visualization. An easy to use GUI using Tkinter <http://www.pythonware.com/library/tkinter/introduction/index.htm> is provided. MayaVi is free software and is distributed under the conditions of the BSD license <http://www.opensource.org/licenses/bsd-license.html>. It is also cross platform and should run on any platform where both Python and VTK are available (which is almost any *nix, Mac OSX or Windows).

Chapter 1

Introduction

MayaVi is a scientific data visualizer. It is written in Python <http://www.python.org> and uses the Visualization Toolkit (VTK) <http://www.vtk.org/> for the visualization. An easy to use GUI using Tkinter <http://www.pythonware.com/library/tkinter/introduction/index.htm> is provided. MayaVi is free software and is distributed under the conditions of the BSD license <http://www.opensource.org/licenses/bsd-license.html>. It is also cross platform and should run on any platform where both Python and VTK are available (which is almost any *nix, Mac OSX or Windows).

In Sanskrit "Mayavi" means magician. The name wasn't exactly chosen for its meaning but was the result of a long and hard search with the author pestering a lot of people for suggestions. My sincere thanks to all of those who offered suggestions.

MayaVi has a quite a few useful features:

- An easy to use GUI.
- MayaVi can be used as a Python module from other Python programs. MayaVi can also be used interactively from the Python interpreter.
- Provides modules to visualize grids, scalar and vector data fields. Rudimentary tensor support is also available.
- It provides support for *any* VTK dataset using the VTK data format <http://www.vtk.org/pdf/file-formats.pdf>. This includes rectilinear, structured and unstructured grid data and also polygonal data. Both the original VTK data formats and the new XML formats are supported.
- Support for PLOT3D data. Both ASCII and binary files work. Only the structured grid format works because of current limitations in the vtkPLOT3DReader. Simple support for multi-block data is also incorporated.
- Support for EnSight data. EnSight6 and EnSightGold formats are supported. Only single parts are supported at this time.
- Many datasets can be used simultaneously.
- Multiple visualization modules can be used simultaneously.
- Quite a few basic data filters are also provided.
- Supports volume visualization of data via texture and ray cast mappers.
- Support for importing a simple VRML scene or a 3D Studio file. Texturing is not yet supported due to limitations in VTK's vtkVRMLImporter.
- A pipeline browser with which one can browse and edit objects in the VTK pipeline. A segmented pipeline browser is used to make it easier to look at parts of the VTK pipeline.
- A modular design so one can add ones own modules and filters.
- A Lookup Table editor to customize lookup tables easily while visualizing data!

- The visualization (or a part of it) can be saved and reloaded in the future.
- Export the visualized scene to a Post Script file, PPM/BMP/TIFF/JPEG/PNG image, Open Inventor, Geomview OOGL, VRML files or RenderMan RIB files. It is also possible to save the scene to a vector graphic via GL2PS <http://www.geuz.org/gl2ps>. This is only available if VTK is built with GL2PS support.
- Support for picking data points or cells and also configuring the lighting of the visualization.

And a lot more. MayaVi is free software and hence can be modified to do things differently. The rest of this manual will provide information on how to use it.

Chapter 2

Getting started

MayaVi is a pretty powerful tool. This chapter describes the GUI that MayaVi provides and the way to use it. This chapter gets you started using MayaVi.

2.1 Starting MayaVi

Under *nix if your installation is setup such that the script **mayavi** is on the system wide path just run the executable **mayavi** anywhere. If not, change the current directory to the directory where MayaVi was installed and run:

```
% mayavi &
```

If you have an already saved MayaVi visualization in some file, say `saved_viz.mv` you can start MayaVi using that file like so:

```
% mayavi saved_viz.mv &
```

Under Windows visit the directory where MayaVi was installed and double click on the executable **mayavi.pyw**. If your installation went well this should start MayaVi.

If you have problems running MayaVi, consult the MayaVi home page and look at the Installation sections. You can also ask for help at the mailing list or ask the author.

2.2 Command line arguments

2.2.1 Basic options

This section lists some simple useful command line options

`mayavi --display DISPLAY` Use DISPLAY for the X display. This option makes sense only when running MayaVi under X.

`mayavi -g WIDTHxHEIGHT+XOFF+YOFF` Set the geometry of the main window when it is launched. The arguments that can be passed follow the standard X convention and include the width, height, x offset and y offset of the window. This option is also available through **`--geometry`**.

`mayavi -h` This prints all the available command line options and exits. Also available through **`--help`**.

`mayavi -V` This prints the MayaVi version on the command line and exits. Also available through **`--version`**.

`mayavi filename.mv` This loads a previously saved MayaVi visualization.

2.2.2 Advanced options

This section lists some advanced command line options. This section is intended for those who already understand how MayaVi works. If you are new to MayaVi it is recommended that you read the rest of the guide and then get back here when you need more advanced command line options.

`mayavi -d vtk_file.vtk` Opens a VTK file (even the new XML format is supported) passed as the argument. Also available through **`--vtk`**.

`mayavi -x plot3d_xyz_file` This opens a PLOT3D co-ordinate file passed as the argument. Also available through **`--plot3d-xyz`**.

`mayavi -q plot3d_q_file` This opens a PLOT3D solution file passed as the argument. Please note that this option must *always* follow a -q or –plot3d-xyz option. Also available through **`--plot3d-q`**.

`mayavi -e ensight_case_file` Opens an EnSight case file passed as the argument. Also available through **`--ensight`**.

`mayavi -m module-name` The passed module name is loaded in the current ModuleManager. The module name must be a valid one if not you will get an error message. Also available through **`--module`**.

`mayavi -f filter-name` The passed filter name is loaded in the current ModuleManager. The filter name must be a valid one if not you will get an error message. Also available through **`--filter`**. If the filter is the **`UserDefined`** filter then it could be specified as **`UserDefined:vtkSomeFilter`** where **`vtkSomeFilter`** is a valid VTK class. In this case the filter will not prompt you for the VTK filter to use.

`mayavi -z saved-visualization-file` Loads a previously saved MayaVi visualization file passed as the argument. Also available through **`--viz`** and **`--visualization`**.

`mayavi -M module-manager-file` Loads a module manager saved to a file. If a file that does not exist is given this will simply create a new module manager that can be populated with filters and modules. Also available through **`--module-mgr`**.

`mayavi -w vrml2-file` Imports a VRML2 scene given an appropriate file. Also available through **`--vrml`**.

`mayavi -3 3DStudio-file` Imports a 3D Studio scene given an appropriate file. Also available through **`--3ds`**.

`mayavi -n` Creates a new window. Any options passed after this will apply to this newly created window. Also available through **`--new-window`**.

2.2.3 Examples

Here are a few interesting examples.

```
% mayavi examples/heart.mv
```

This command loads an existing visualization.

```
% mayavi -z examples/heart.mv -n -z examples/other.mv
```

This command loads the `heart.mv` saved visualization in one window, creates a new window and loads the `other.mv` in the other.

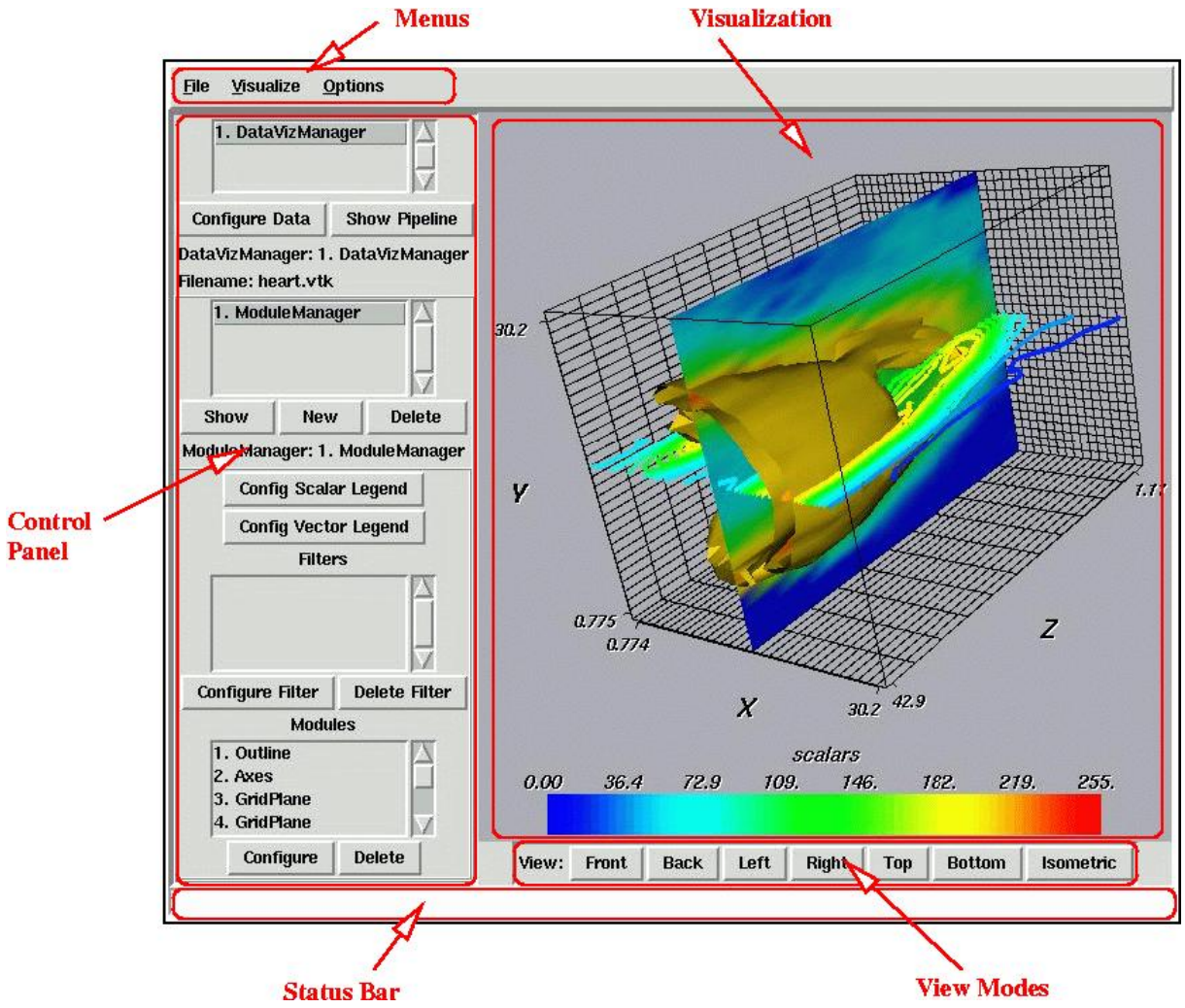
```
% mayavi -d examples/heart.vtk -m Axes -m GridPlane \
>  -M new -f Threshold -m IsoSurface \
>  -n -d examples/heart.vtk -m Outline -m ContourGridPlane
```

This command loads a VTK data file called `heart.vtk`, loads the `Axes`, `GridPlane` modules in one `ModuleManager`. Then creates a new `ModuleManager` and loads a `Threshold` filter and an `IsoSurface` module in it. It then opens a new visualization window, loads the VTK data file, `heart.vtk`, the modules `Outline` and `ContourGridPlane` in it.

The provided options make it possible to construct very useful visualizations from the command line.

2.3 The MayaVi Window

MayaVi provides an easy to use GUI. The picture shown below shows the basic GUI that MayaVi provides. The regions marked out in red are to be noted. The top left shows a set of menus. Below the menus is a control panel on the left and the actual visualization on the right. At the bottom of the application window is a status bar that turns red when MayaVi is busy doing something. In between the status bar and the visualization are provided a set of buttons that help control the visualization view.



Each section of the screen marked and described above provides important functionality.

Menu This provides a set of menus from which provide the user with bulk of the functionality.

Visualization This part of the screen is where the data is visualized using VTK.

Control Panel The control panel allows the user to configure and control the particular visualization. It provides various lists for the user's convenience. These are discussed in detail subsequently.

Status Bar This part of the screen indicates the status of MayaVi to the user. If MayaVi is busy doing something this part of the screen will turn red and the cursor will change to a *watch* indicating that MayaVi is busy.

View Modes These are a set of convenience buttons that help the user quickly see one particular view of the visualization.

The next chapter deals with using MayaVi.

Chapter 3

Using MayaVi

This chapter describes in detail the way to use MayaVi for your data visualization.

3.1 The Basic Design of MayaVi

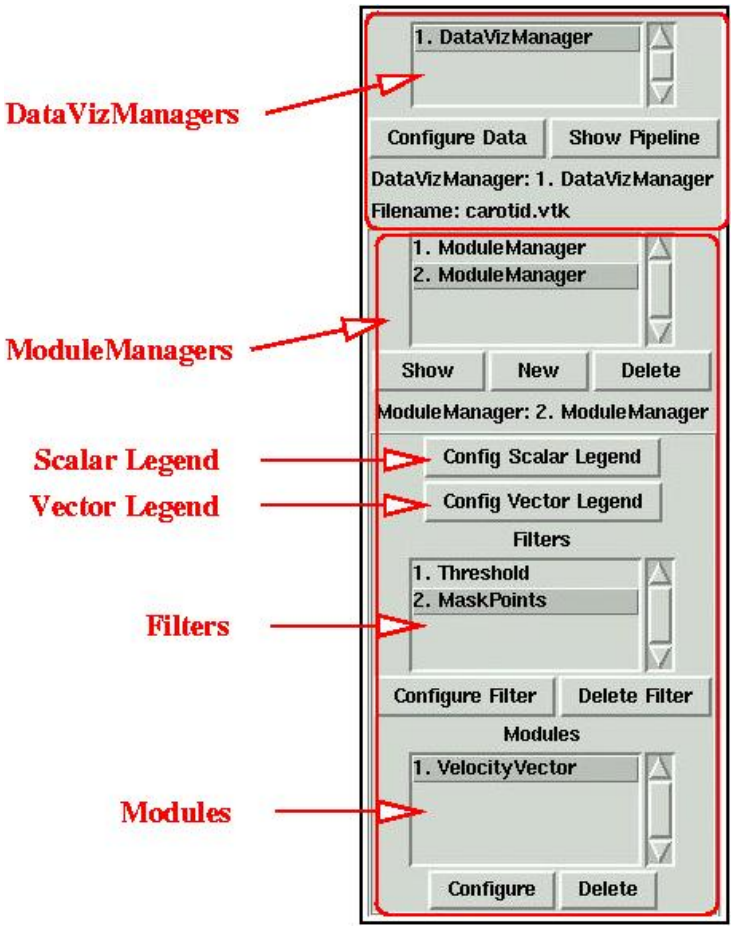
It is important to understand the basic design of MayaVi before you use it. MayaVi is a data visualizer and the design reflects this. The basic design is simple and is described in this section.

3.1.1 The Control Panel

The control panel needs to be understood before one can do anything serious with MayaVi. This section describes the control panel in some detail.

- Associated with each data file that is to be visualized is an object called a `DataVizManager`. This object is responsible for the datafile and the entire visualization associated with that data file. Each `DataVizManager` instance is shown in the first list in the control panel.
- Each `DataVizManager` controls a set of `ModuleManager`s. These `ModuleManager`s are listed in the second list from the top.
- Each `ModuleManager` controls set of two legends (one for scalar visualization and one for vector visualization), a collection of `Filter`s and a collection of `Module`s. Any number of Filters and Modules can be used.
- A `Filter` is an object that filters out the data in some way or the other. A simple example is the `ExtractVectorNorm` filter. This extracts the magnitude of the selected vector data field attribute for the data file. All modules that use this as an input will receive the vector magnitude as the scalar field data. The filters can be chosen from the **Visualize** menu. Each `ModuleManager` can have as many filters as are required. When multiple filters are used, it is important to note that each filter sends its data to the next filter in sequence. This could be problematic in some situations. Lets say there is a structured grid object and that needs to be subsampled. We can use the `ExtractGrid` filter and then display a `GridPlane`. Now we want to show contours but this time we want to threshold the contours based on input scalars so we use the `Threshold` filter. If we put the `Threshold` filter in the same ModuleManager, the grid will disappear since the `Threshold` output is an unstructured grid. So to handle this we create a new `ModuleManager` (click the **New** button) and add the `Threshold` filter in that `ModuleManager`. Put all the modules that use this filter in that `ModuleManager`.
- A `Module` is an object that actually visualizes the data. There are a large number of Modules that can be used and these are also available from the **Visualize** menu. Each `ModuleManager` can have as many modules as are required.

Although the above sounds complex, it really is not. It may just take a little getting used to before you are completely comfortable with it. The following figure illustrates the above and clarifies matters.



If you have multiple `DataVizManagers` and want to see the `ModuleManagers` of one of them then you either double click on the appropriate manager or single click on the manager and click on the **Show Pipeline** for the `DataVizManagers` and on the **Show** for the `ModuleManagers`.

The other GUI buttons and features are all rather self explanatory. There are only a few points that need to be made in order to make this description complete:

- The creation and deletion of a `DataVizManager` is controlled via the **File** menu. The open menu items will create a new `DataVizManager` and the **Close** menu item will close the selected `DataVizManager` and all its `ModuleManagers`. The **Close All** menu item will close all the `DataVizManagers`.
- The addition and deletion of `ModuleManagers`, can be done using the **New** and **Delete** buttons in the control panel.
- `Filters` and `Modules` can be added from the **Visualize** menu and the corresponding **Filters** and **Modules** the sub menus. They can be deleted from the control panel.
- `Filters` and `Modules` can be configured by either double clicking on the corresponding item or by selecting the item and clicking on the **configure** button.

3.2 Data formats

MayaVi is a data visualizer and one first needs to have data to visualize anything. MayaVi supports the following data formats:

VTK data format <`http://www.vtk.org/pdf/file-formats.pdf`> MayaVi supports *any* VTK dataset using the VTK data format <`http://www.vtk.org/pdf/file-formats.pdf`>. This includes rectilinear, structured and unstructured grid data and also polygonal data. Please refer the linked document for details on the VTK data format. The new VTK XML data format is also supported by MayaVi.

PLOT3D data MayaVi supports PLOT3D file format with binary structured grid data. The other PLOT3D datasets will not work due to limitations in VTK's vtkPLOT3DReader. Simple support for multi-block data is also incorporated.

EnSight data MayaVi supports EnSight data. EnSight6 and EnSightGold formats are supported. Only single parts are supported at this time.

In addition to this MayaVi allows one to import VRML2 files and 3D Studio files. Texturing is not yet supported due to limitations in VTK's vtkVRMLImporter.

Those interested in converting data arrays under Python into VTK files should look at Pearu Peterson's pyVTK <`http://cens.ioc.ee/projects/pyvtk/`> Python module.

MayaVi ships with a simple example of a heart CT scan data in the VTK data format (this should be in the `examples/` directory). The dataset used is a structured points dataset with a scalar field. This can be used as a simple reference. More VTK, PLOT3D and EnSight data samples should be available from the VTK download page <`http://www.vtk.org/get-software.php`>.

3.3 Opening a data file

Once data suitable for MayaVi is available one can begin the visualization. If you don't have data then please note that MayaVi ships with a simple example of a heart CT scan data in the VTK data format (this should be in the `examples/` directory). The dataset used is a structured points dataset with a scalar field. More VTK, PLOT3D and EnSight data samples should be available from the VTK download page <`http://www.vtk.org/get-software.php`>.

The first thing to do is load an appropriate data file. Visit the **File** menu and choose the appropriate menu item for the data you have and select the file you want from the resulting GUI. Once you do this, you will notice that the control panel will show a *DataVizManager* at the topmost list and you will see a lot of controls below this. The control panel is described in some detail in this section. If you aren't familiar with that section yet, this might be a good time to review it.

Once the data file is opened a dialog box will pop up that allows one to configure the datafile. One can choose the appropriate data field one is interested in. This configuration window can be closed when unnecessary. To reconfigure the data use the **Configure Data** button provided in the control panel. Only one scalar attribute and one vector attribute is supported at a given time. Each *DataVizManager* that is seen represents a different file. One can open as many data files as one wants. Different data types can be loaded simultaneously too. Using a similar procedure one can import a simple VRML2 scene or even a 3D studio file. To close a VRML or 3D Studio file choose the appropriate file in the **Close** menu.

MayaVi also supports time series data. If the file name ends with an integer, MayaVi treats this integer as a time index. All files in the same directory as the loaded file are scanned. If any of them share the same pattern (without the last integer) as the name of the opened file, then the files are treated as part of the time series. These files are then sorted. The **Configure Data** GUI automatically addes a slider to switch between these time steps and an auto-sweep button to sweep through the time series.

Once the data file is read and a `DataVizManager` is created an instance of a `ModuleManager` is also created. At this point one has to load the `Filter`s and the `Module`s in order to do the visualization.

3.4 Visualizing the Data

Once the data file has been opened and the appropriate field attribute for both scalars and vectors has been chosen one can either filter the data or one can directly apply a module to the data and visualize it. MayaVi provides a large number of Modules and a few Filters. They are described in more detail subsequently.

In order to Filter the data one must use the **Visualize** menu and from the **Filters** sub-menu choose the appropriate filter. As soon as a filter is requested a popup window will appear that helps you configure the particular filter. Please note that it is not at all necessary to filter the data. If no filtering is required one does not need to load any filter. Even if a filter is used one can delete it at any time using the controls provided in the control panel.

In order to use a particular visualization module a procedure similar to the one for Filters is used. One merely uses the **Visualize** menu and from the **Modules** sub-menu chooses the appropriate Module. The module might take a little while to load. If there is some kind of error a warning dialog will attempt to describe the problem and hopefully the user can correct the situation.

3.4.1 Navigating the Visualization

It is important to be able to navigate the data and view it appropriately. In MayaVi this is achieved in one of two ways. Using the standard view mode buttons provided at the bottom of the visualization widget or by using the mouse to navigate through the visualization. The mouse based navigation is far more powerful and general purpose. The buttons however provide quick shortcuts to commonly desired views. The buttons and the visualization widget are shown in the illustration in an earlier chapter.

3.4.1.1 Navigating using the mouse

Mouse navigation is powerful but takes a little getting used to. It is relatively simple and with experience can be used easily. This section briefly describes how one can use the mouse to navigate through the data.

Rotating the visualization With 3D visualization it is important to be be able to rotate the visualized scene. In MayaVi this is achieved by first placing the mouse pointer on top of the visualization window. Then one keeps the left mouse button pressed and drags the mouse pointer in the direction one needs to rotate the scene. This is very much like rotating an actual object.

Zooming in and out To zoom in and out of the scene first one places the mouse pointer inside the visualization window. To zoom into the scene one keeps the the right mouse button pressed and drags the mouse upwards. To zoom out of the scene one keeps the right mouse button pressed and drags the mouse downwards.

Panning the scene To pan a scene implies translating the center of the rendered scene. In MayaVi this is done in two ways.

1. By keeping the left mouse button pressed and simultaneously holding down the **Shift** key and dragging the mouse in the appropriate direction.
2. By keeping the middle mouse button pressed and dragging the mouse in the appropriate direction.

Just practice this a few times and you should get used to this pretty easily. This practically covers all that you need to know to be able to use MayaVi effectively. The best way to really learn about MayaVi is to explore the various options and try them out. Subsequent sections provide more details on the various menu's provided and the various modules and filters that are available.

3.5 Picking data

MayaVi supports data picking. While visualizing some data press the **p** or **P** key to pick the data point above the current mouse position. This will pop up a new window where the picked values will be displayed. The window also has a few controls that let you configure the type of picking you wish to perform. Please note that the picker will pick data on the actors that you have visualized on screen. It will attempt to find the nearest actor and pick the data on that. The picked point will be highlighted using an axes. The picker supports picking the following:

Picking the nearest point This option is the default and lets you pick the nearest point in the data. You could use this if you have data specified as point data. Note that when you pick using this option the picked location may not be exactly where you placed the mouse. The location will snap to the nearest available point. By changing the tolerance presented in the GUI you can control how near the point should be to the exact picked position.

Picking the nearest cell This option lets you pick the nearest cell in the data. You could use this if you have data specified as cell data. By changing the tolerance you can control how closely the picker finds the nearest cell.

Picking an arbitrary point This option lets you pick an arbitrary point in space that is not tied to the nearest cell or point. The motion of this picker is much smoother than the point or cell picker. However, the point that you pick will not be exactly at a point in the actual data. The results will therefore be interpolated using a probe filter.

3.6 Configuring the lights

MayaVi allows you to configure the lighting of the scene using a graphical utility. When the mouse is over the visualization frame of the window press the **l** or **L** key to open the light configuration kit. This will pop up a new window where you can configure as many as eight different lights. The default light is to have one light placed as a headlight. A headlight is a light that points directly ahead in the same direction as the camera. It is possible to change the position of the default light. It is also possible to turn on other lights, configure their elevation and azimuthal positions using the sliders provided, configure their intensity and color. To configure a particular light click on one of the conical glyphs that indicate a particular light. The GUI is fairly easy to follow. Experiment with it to become comfortable with it.

Note that if you save a visualization your light settings are also saved and when you reload the visualization these settings are restored.

3.7 The Menus

This section details the various menus that MayaVi provides. Almost all the menu items have hot keys associated with them. The underlined letter indicates the key sequence to be used. Consider the case of the **New Window** menu item (the letter *N* is underlined) that is in the **File** menu (the letter *F* is underlined). This can be reached by using the following key strokes. **Alt-F** followed by **N**. The Menu itself requires the **Alt** modifier but the menu item does not. The following are the various menus that MayaVi provides.

3.7.1 File Menu

The **File** menu provides the following menu items.

New Window This creates a new MayaVi visualization window that is completely independent of the first window. Any number of such windows can be created.

Open This provides a submenu containing two items.

- **VTK file** This provides a GUI dialog from which a VTK file can be selected for opening. Once the file is opened a new `DataVizManager` is created and a GUI dialog for configuring the data file is provided.
- **VTK XML file** This provides a GUI dialog from which a VTK XML file can be chosen. This is a new VTK format and is only available in VTK versions higher than 4.0.
- **PLOT3D file** This provides a sub menu from which either a PLOT3D file containing single block or multi-block binary structured grid data can be selected for opening. Once the file is opened a new `DataVizManager` is created and a GUI dialog for configuring the data file is provided.
- **EnSight case file** This provides a sub menu from which an EnSight case file can be opened.

Import This provides a submenu containing two items.

- **VRML2 scene** This loads a VRML2 scene into the current visualization.
- **3D Studio scene** This provides a menu containing all the VRML files that are already opened. The chosen VRML file is closed and the VRML files actors are removed from the rendered scene.

Load This provides a submenu containing three items.

- **Visualization** This loads a saved *complete* visualization.
- **ModuleManagers** This creates new `ModuleManager`s for the current `DataVizManager` and loads `ModuleManager`s that have been saved to a file into the newly created ones.
- **ModuleManagers (Append)** This is slightly different from the previous menu item and loads the first of the saved `ModuleManager`s into the current `ModuleManager` and for the subsequent saved `ModuleManager`s it creates new `ModuleManager`s and loads `ModuleManager`s from the saved file.

Save This provides a submenu containing four items.

Entire Visualization This saves the *entire* visualization configuration to a file such that it can be loaded by the **Load** menu's **Visualization** menu item.

Current DataVizManager This creates saves the current `DataVizManager` to a file. This can be loaded as a visualization.

Current ModuleManager This enables one to store the currently active `ModuleManager` into a file such that it can be loaded later.

All ModuleManagers This enables one to store the all the `ModuleManagers` for the currently active `DataVizManager` into a file such that it can be loaded later.

Save Scene to Provides a menu which in turn provides menu items to export the visualized scene to a Post Script file, PPM/BMP/TIFF/JPEG/PNG image, Open Inventor, Geomview OOGL, VRML and RenderMan RIB files. It is also possible to save the scene to a vector graphic via GL2PS <`http://www.geuz.org/gl2ps`>. This is only available if VTK is built with GL2PS support.

Close Provides submenu's to close the current `DataVizManager`. This means that all the `ModuleManagers` of that particular `DataVizManager` will also be deleted. It also provides menus to close the currently active VRML and 3D Studio scenes that have been imported.

Close All Close all the `DataVizManagers` and all the imported VRML2 and 3D Studio scenes.

Exit Close this particular MayaVi Window. If this is the only MayaVi window the application exits completely.

3.7.2 Visualize Menu

The **Visualize** menu provides the following menu items.

Modules Provides a sub-menu which contains a list of all available `Modules`. This list is dynamically generated based on the available modules.

Filters Provides a sub-menu which contains a list of all available `Filters`. This list is dynamically generated based on the available filters.

Pipeline browser This creates a GUI that shows the entire VTK visualization pipeline. The objects in the pipeline can be configured by double clicking on the items. If there are a large number of objects this can become confusing to use and it would be better to use the pipeline segment browser configuration provided with the configuration for each `Module`

3.7.3 Options Menu

The **Options** menu provides the following menu items.

Preferences Provides a GUI using which one can edit the default preferences. The preferences allow one to set various default settings including foreground color, background color, default directory where the file related dialogs will open in initially. These options can be saved so that the next time MayaVi is started it will use these defaults. If you set the default directory to an empty one the directory that the file open/save dialogs will use will be intelligently chosen. The stereo rendering option enables stereo rendering in the MayaVi window. If the save current lighting option is set then the current lighting is saved as the default and used in all subsequent MayaVi visualizations.

The search path setting allows the user to specify a list of directories where user defined sources, modules and filters are made available. The search path is a ':'-separated string and is specified like the PYTHONPATH. '~', '~user' and '$VAR' are all expanded. Each of the directories specified in this string can have a `Sources/`, a `Modules/` and a `Filters/` directory inside where user defined sources, modules and filters can be stored. These modules and filters will be made available inside the **User** sub-menu of the **File/Open**, **Module** and **Filter** menus respectively. These sources, modules and filters can be used from the command line or from a Python interpreter session by using 'User.SourceName' (sources cannot be specified from the command line), 'User.ModuleName' or 'User.FilterName'. When creating user defined sources or modules or filters make sure that the name of the module is the same as the name of the class that defines the particular object.

Configure RenderWindow Provides a simple GUI to configure the Visualization RenderWindow. This is also where one can change the stereo rendering options.

Change Foreground Allows one to change the default foreground color.

Change Background Allows one to change the default background color.

Show Debug window Pops up a small window where debug messages are printed. This is very useful if you run into trouble and want to know what happened. It also shows the function call sequence. The messages are also printed to stderr.

Show Control Panel Toggles the visibility of the control panel. This can be very useful when you want to do a full screen visualization.

Reload Modules This function reloads all the currently loaded Python modules. This is *very* useful while debugging a new feature for some module that one is creating. One may also see funny behavior for already instantiated objects.

3.7.4 Help Menu

The **Help** menu provides the following menu items.

About Displays a few details about MayaVi.

Users Guide Opens a web browser and displays this MayaVi users guide.

Home page Opens a web browser and displays the MayaVi home page.

3.8 Module Documentation

The following are the list of provided Modules along with a brief description.

Axes This module creates and manages a set of three axes for your data. The class uses a vtkCubeAxesActor2D.

BandedSurfaceMap Displays a surface map with special contouring using the vtkBandedPolyDataContourFilter. This contour filter produces filled contours of the same color between two contour lines rather than either a continuous color distribution or just individual contour lines. It should work for any input dataset. It is best used for 2d surfaces. Note that one can either specify a total number of contours between the minimum and maximum values by entering a single integer or specify the individual contours by specifying a Python list/tuple in the GUI.

ContourGridPlane This module shows a grid plane of the given input structured, rectilinear or structured points grid with the scalar data either as a color map or as contour lines. This works only for structured grid, structured point and rectilinear grid data. Note that one can either specify a total number of contours between the minimum and maximum values by entering a single integer or specify the individual contours by specifying a Python list/tuple in the GUI.

CustomGridPlane This module shows a grid plane of the given input grid. The plane can be shown as a wireframe or coloured surface with or without scalar visibility and contour lines. The module basically wraps around the vtk*GeometryFilters. This module enables one to completely configure the grid plane. It works only for structured grid, structured point and rectilinear grid datasets. Note that one can either specify a total number of contours between the minimum and maximum values by entering a single integer or specify the individual contours by specifying a Python list/tuple in the GUI.

Glyph This module displays glyphs scaled and colored as per the input data. This will work for any dataset and can be used for both scalar and vector data.

GridPlane This module shows a grid plane of the given input grid. The plane can be shown as a wireframe or coloured surface with or without scalar visibility. This works only for structured grid, structured point and rectilinear grid data. Useful for debugging and displaying your created grid.

HedgeHog This module shows the given vector data as a 'hedge hog' plot. The lines can be colored based on the input scalar data. This class should work with any dataset.

IsoSurface This module shows an iso-surface of scalar data. This will work for any dataset.

Labels Displays text labels of input data. When instantiated, the class can be given a module name (the same name as listed in the Modules GUI) or an index of the module (starting from 0) in the current module manager. If this is not provided the module will ask the user to choose a particular module or choose filtered data. The module will then generate text lables for the data in the chosen module and display it. The module provides many configuration options. It also lets one turn on and off the use of a vtkSelectVisiblePoints filter. Using this filter will cause the module to only display visible points. Note that if the module that is being labeled has changed significantly or is deleted this Labels module will have to be updated by changing one of the settings (like the RandomModeOn check button) to a different value and then back to the original one. Alternatively, choose the module to be labeled again.

Locator This module creates a 'Locator' axis, that can be used to mark a three dimensional point in your data.

Outline Displays an Outline for any data input.

PolyData Displays any input polydata, nothing fancy.

ScalarCutPlane This module plots scalar data on a cut plane either as a color map or with contour lines. This will work for any dataset. Note that one can either specify a total number of contours between the minimum and maximum values by entering a single integer or specify the individual contours by specifying a Python list/tuple in the GUI.

Streamlines This module makes it possible to view streamlines, streamtubes, and stream ribbons for any type of vector data. Any number of point sources can be added and deleted. A fairly powerful UI is provided. This module should work with any dataset.

StructuredGridOutline Displays an Outline for a structured grid.

SurfaceMap Displays a surface map of any data. It should work for any dataset but is best if used for 2d surfaces (polydata and unstructured surfaces). Note that one can either specify a total number of contours between the minimum and maximum values by entering a single integer or specify the individual contours by specifying a Python list/tuple in the GUI.

TensorGlyphs This module displays glyphs, scaled and colored as per the tensor data. This will work for any dataset.

Text Displays simple text on the screen. The text properties and position are configurable. The text can also be multi-line if newlines are embedded in it.

VectorCutPlane This module displays cone glyphs scaled and colored as per the vector or scalar data on cut plane. This will work for any dataset.

VelocityVector This module displays cone or arrow glyphs scaled and colored as per the vector data. This will work for any dataset.

Volume This Volume module allows one to view a structured points dataset with either unsigned char or short data as a volume. The module also provides a powerful GUI to edit the Color Transfer Function (CTF). You can drag the mouse with different buttons to change the colors. The following are the mouse buttons and key combinations that can be used to edit the CTF – red curve: Button-1, green curve: Button-2/Control-Button-1, blue curve: Button-3/Control-Button-2, alpha/opacity: Shift-Button-1. It is possible to use either the vtkVolumeRayCastMapper or the vtkVolumeTextureMapper2D. It is also possible to choose among various ray cast functions.

WarpVectorCutPlane This module takes a cut plane and warps it using a vtkWarpVector as per the vector times a scale factor. This will work for any dataset.

3.9 Filter Documentation

The following are the list of provided Filters along with a brief description.

CellToPointData This class produces PointData given an input that contains CellData. This is useful because many of VTK's algorithms work best with PointData. The filter basically wraps the vtkCellDataToPointData class.

CutPlane This filter takes a cut plane of any given input data set. It interpolates the attributes onto a plane. The position and orientation of the plane are configurable using a GUI.

Delaunay2D This filter wraps around the vtkDelaunay2D filter and lets you do 2D triangulation of a collection of points. The key parameters are Tolerance and the Alpha value. Tolerance gives the criteria for joining neighbouring data points and alpha is the threshold for the circumference of a caluculated triangulated polygon.

Delaunay3D This filter wraps around the vtkDelaunay3D filter and lets you do 3D triangulation of a collection of points. The key parameters are Tolerance and the Alpha value. Tolerance gives the criteria for joining neighbouring data points and alpha is the threshold for the circumference of a caluculated triangulated polygon.

ExtractGrid Wraps vtkExtractGrid (structured grid), vtkExtractVOI (imagedata/structured points) and vtkExtractRectilinearGrid (rectilinear grids). These filters enable one to select a portion of, or subsample an input dataset. Depending on the input data the appropriate filter is used.

ExtractTensorComponents This wraps the vtkExtractTensorComponents filter and allows one to select any of the nine components or the effective stress or the determinant from an input tensor data set. This will work for any dataset.

ExtractUnstructuredGrid This wraps the vtkExtractUnstructuredGrid filter. From the VTK docs: vtkExtractUnstructuredGrid is a general-purpose filter to extract geometry (and associated data) from an unstructured grid dataset. The extraction process is controlled by specifying a range of point ids, cell ids, or a bounding box (referred to as 'Extent'). Those cells lying within these regions are sent to the output. The user has the choice of merging coincident points (Merging is on) or using the original point set (Merging is off).

ExtractVectorComponents This wraps the vtkExtractVectorComponents filter and allows one to select any of the three components of an input vector data attribute.

ExtractVectorNorm This wraps the vtkVectorNorm filter and produces an output scalar data with the magnitude of the vector.

MaskPoints This wraps the vtkMaskPoints filter. The problem with this filter is that its output is Polygonal data. This means that if you add this filter to a ModuleManager with visualizations apart from HedgeHog or other velocity vector data you won't see anything! If that happens create another ModuleManager and show the other visualizations there. Also, this means that this filter should be typically inserted at the end of the list of filters.

PolyDataNormals This wraps the vtkPolyDataNormals filter. vtkPolyDataNormals is a filter that computes point normals for a polygonal mesh. This filter tries its best to massage the input data to a suitable form. Its output is a vtkPolyData object. Computing the normals is very useful when one wants a smoother looking surface.

StructuredPointsProbe A useful filter that can be used to probe any dataset using a Structured Points dataset. The filter also allows one to convert the scalar data to an unsigned short array so that the scalars can be used for volume visualization.

Threshold This wraps the vtkThreshold filter. The problem with this filter is that its output is an Unstructured Grid. This means that if you add this filter to a ModuleManager of a gridded dataset and you have a few grid planes, then your grid planes won't show anymore. If that happens create another ModuleManager and show the grid planes there acting on unfiltered data.

UserDefined This filter wraps around a user specified filter and lets one experiment with VTK filters that are not yet part of MayaVi. By default if the class is instantiated it will ask the user for the VTK class to wrap around. If passed a valid VTK class name it will try to use that particular class.

Vorticity This filter computes the vorticity of an input vector field. For convenience, the filter allows one to optionally pass-through the given input vector field. The filter also allows the user to show the component of the vorticity along a particular cartesian co-ordinate axes.

WarpScalar This wraps the vtkWarpScalar filter. vtkWarpScalar is a filter that modifies point coordinates by moving points along point normals by the scalar amount times the scale factor. Useful for creating carpet or x-y-z plots.

WarpVector Warps the geometry using the vtkWarpVector filter. vtkWarpVector is a filter that modifies point coordinates by moving points along vector times the scale factor. Useful for showing flow profiles or mechanical deformation.

Chapter 4

Using MayaVi from Python

If you have installed MayaVi from the sources and are not using a binary release, then you can use MayaVi as a Python module. This chapter details how you can use MayaVi as a Python module. If you are looking for a powerful, interactive, cross-platform Python interpreter you might be interested in IPython. <http://ipython.scipy.org>

4.1 An example

Its very easy using MayaVi as a Python module. Thanks to Tkinter, it is also possible to use MayaVi from the Python interpreter. This means that one can script MayaVi! This is a pretty powerful and useful feature. To illustrate using MayaVi as a module and its scriptability, we will consider a few simple examples where the user generates some data and a VTK file and then uses MayaVi to visualize the data.

4.1.1 Generating some data

```
>>> # generate the data.
>>> from Numeric import *
>>> import scipy
>>> x = (arange(50.0)-25)/2.0
>>> y = (arange(50.0)-25)/2.0
>>> r = sqrt(x[:,NewAxis]**2+y**2)
>>> z = 5.0*scipy.special.j0(r)  # Bessel function of order 0
>>> # now dump the data to a VTK file.
>>> import pyvtk
>>> # Flatten the 2D array data as per VTK's requirements.
>>> z1 = reshape(transpose(z), (-1,))
>>> point_data = pyvtk.PointData(pyvtk.Scalars(z1))
>>> grid = pyvtk.StructuredPoints((50,50, 1), (-12.5, -12.5, 0), (0.5, 0.5, 1))
>>> data = pyvtk.VtkData(grid, point_data)
>>> data.tofile('/tmp/test.vtk')
```

The above example uses the Numeric <http://numpy.sourceforge.net>, SciPy <http://www.scipy.org> and pyVtk <http://cens.ioc.ee/projects/pyvtk/> modules. Please note the step where `z1` is obtained from `z`. This step is done to correctly flatten the two dimensional array `z`. The problem with Numeric arrays and VTK data is that you have to be careful of the order of the data points. The way VTK reads data (for all the data formats that have a structure) is something like this:

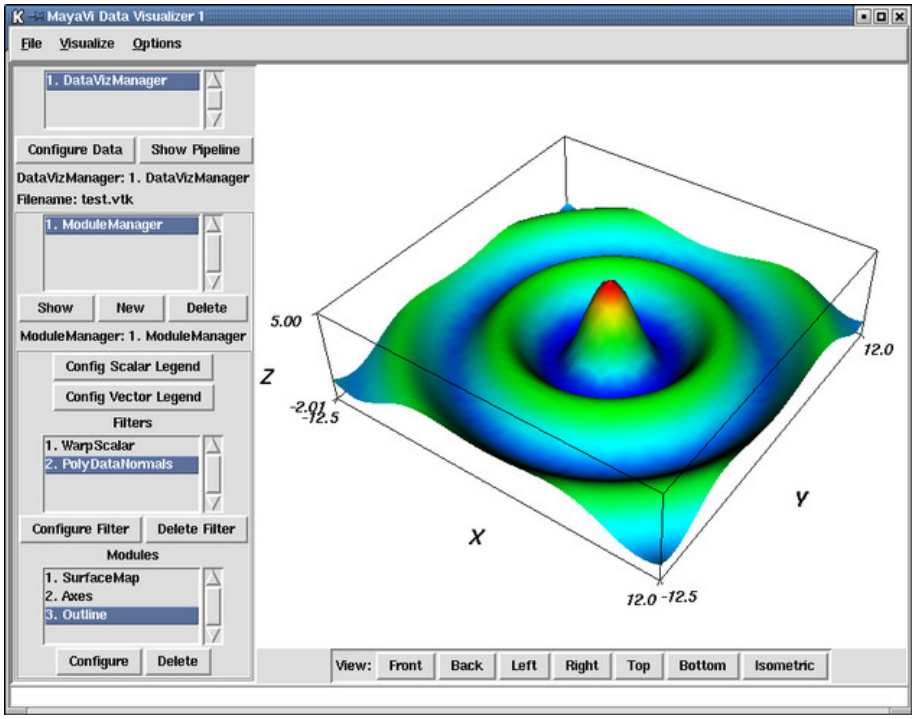
```
>>> for k in range(n_z):
>>>     for j in range(n_y):
>>>         for i in range(n_x):
```

```
>>>             read_line()
```

This means that the x values must be iterated over first, the y values next and the z values last. If you simply flatten the 2D numeric array then this will not happen properly. By using `reshape(transpose(z), (-1,))` we ensure that the data points are specified in the correct order. The next step is to visualize the generated data.

4.1.2 Visualize the generated data

```
>>> import mayavi
>>> v = mayavi.mayavi() # create a MayaVi window.
>>> d = v.open_vtk('/tmp/test.vtk', config=0) # open the data file.
>>> # The config option turns on/off showing a GUI control for the data/filter/module.
>>> # load the filters.
>>> f = v.load_filter('WarpScalar', config=0)
>>> n = v.load_filter('PolyDataNormals', 0)
>>> n.fil.SetFeatureAngle (45) # configure the normals.
>>> # Load the necessary modules.
>>> m = v.load_module('SurfaceMap', 0)
>>> a = v.load_module('Axes', 0)
>>> a.axes.SetCornerOffset(0.0) # configure the axes module.
>>> o = v.load_module('Outline', 0)
>>> v.Render() # Re-render the scene.
```



The result of this is seen in the above figure. It is important to note that the Python interpreter will continue to remain interactive when MayaVi is running. In fact, it is possible to create an animation from the interpreter as done in the following.

```
>>> # now do some animation.
>>> import time
>>> for i in range (0, 10):
...  f.fil.SetScaleFactor(i*0.1)
...  v.Render()
```

```
...  v.renwin.save_png('/tmp/anim%d.png'%i) # save the image to a PNG file
...  time.sleep(1)
>>>
```

The above example saves the screen each iteration to a PNG image. One will need VTK 4.0 for PNG support. These images can be later used by some other utility to create a movie. It is therefore possible to create very useful visualizations from within the Python interpreter.

4.1.3 Using VTK data objects

There are times when the user has created a VTK data object that needs to be visualized. MayaVi has a special data handler for such cases. The following shows how this can be used. The example itself uses a VTK file but the data could have also been generated using other means.

```
>>> # import VTK
>>> import vtk
>>> # create some data.
>>> reader = vtk.vtkStructuredPointsReader()
>>> reader.SetFileName('/tmp/test.vtk')
>>> reader.Update()
>>> data = reader.GetOutput()  # this is a vtkStructuredPoints object.
>>> import mayavi
>>> v = mayavi.mayavi() # create a MayaVi window
>>> v.open_vtk_data(data) # load the data from the vtkStructuredPoints object.
>>> f = v.load_filter('WarpScalar', 0)
>>> # Load other filters and modules...
```

The above example uses a `vtkStructuredPoints` as the input. Other types can also be used as the input. The other valid types are: `vtkRectilinearGrid`, `vtkStructuredGrid`, `vtkUnstructuredGrid` and `vtkPolyData`. Any of these objects can be used as an input and then visualized using MayaVi.

Right now the best way to find out what functions are available etc. would be to read the sources or use pydoc to browse through the code. Experimenting with MayaVi from the interpreter is also a good idea and will be highly educative.

4.1.4 Standalone MayaVi scripts

After interactively exploring MayaVi from the interpreter one usually would like to run these in a non-interactive fashion. That is you'd like to create a Python script that invokes MayaVi. The easiest way to do it is as shown in the following simple example

```
import mayavi
v = mayavi.mayavi()
v.load_visualization('heart.mv')
# Do whatever you please with the MayaVi window.

# To make the MayaVi window interact with the user and wait
# till it is closed to proceed, do the following:
v.master.wait_window()

# Now once the previous window is closed if you need
# to open another do this:
v = mayavi.mayavi()
d = v.open_vtk('file.vtk')
# etc.
v.master.wait_window()
```

```
# Once the MayaVi window is closed the program will exit.
```

As can be seen above, it is easy to use code from an interactive session in a standalone Python script. It is also possible to script MayaVi in the following manner.

```
import Tkinter
r = Tkinter.Tk()
r.withdraw()
import mayavi
v = mayavi.Main.MayaViTkGUI(r)
v.load_visualization('heart.mv')
# Do whatever you please with the MayaVi window.

# now do this to start the Tk event loop.
root.mainloop()
# Once the MayaVi window is closed the program will exit.
```

This is an alternative way to do use MayaVi from Python scripts. This might be helpful if you have used `Tkinter` and know how to use it. However, the first approach is a lot easier.

4.2 Useful Python Modules

This section describes some other useful modules that are released as part of MayaVi but are not necessarily part of the core MayaVi module/application. The module `ivtk` is described in the next section. The MayaVi package also contains a sub-package called `tools`. This directory contains miscellaneous but useful tools that use or are related to MayaVi. This is described subsequently.

4.2.1 The Interactive VTK module

It is very nice to be able to use and experiment with VTK from the Python interpreter. In order to make this easier I've written a simple module that uses some of the MayaVi classes. This makes using VTK from Python very pleasant. The module is called `ivtk` which stands for interactive VTK. `ivtk` provides the following features.

- An easy to use VTK actor viewer that has menus to save the scene, change background, show a help browser, show a pipeline browser etc.
- A simple class documentation search tool/browser that lets you search for arbitrary strings in the VTK class documentation and lets you browse the VTK class documentation.
- An easy to use GUI to configure VTK objects using the `vtkPipeline.ConfigVtkObj` module.
- An integrated picker that can be activated by pressing the **p** or **P** keys. This picker functions the same way as the MayaVi picker.
- An integrated light configuration kit that can be activated by pressing the **l** or **L** keys. This light configuration functions the same way as the MayaVi light kit.

The help browser allows one to search for arbitrary strings in the VTK class documentation. 'and' and 'or' keywords are supported and this makes searching for specific things easier. If a search is successful a list of matching classes is returned. Clicking on a class will pop up a window with the particular class documentation. It is also possible to search for a particular class name. All classes matching the searched name will be shown. The searching is case insensitive.

Here is a sample session that illustrates how `ivtk` can be used. A simple cone example is shown.

```
>>> from mayavi import ivtk
>>> from vtk import *
>>> c = vtkConeSource()
>>> m = vtkPolyDataMapper()
>>> m.SetInput(c.GetOutput())
>>> a = vtkActor()
>>> a.SetMapper(m)
>>> v = ivtk.create_viewer() # or ivtk.viewer()
# this creates the easy to use render window that can be used from
# the interpreter.  It has several useful menus.

>>> v.AddActors(a)    # add actor(s) to viewer
>>> v.config(c)       # pops up a GUI configuration for object.
>>> v.doc(c)          # pops up class documentation for object.
>>> v.help_browser()  # pops up a help browser where you can search!
>>> v.RemoveActors(a) # remove actor(s) from viewer.
```

The `AddActors/RemoveActors` method can be passed a list/tuple or a single actor. All of the passed actors will be added/removed to the `vtkRenderWindow`. The `config` method provides an easy to use GUI to configure the passed VTK object. The viewer also provides menus to save the rendered scene and also provides a menu to open a VTK Pipeline browser that can be used to browse the VTK pipeline and configure objects in it.

Even without creating the actor viewer it is possible to use the help browser and the configure code as shown below.

```
>>> from mayavi import ivtk
>>> d = ivtk.doc_browser()
# pops up a standalone searcheable VTK class help browser.
>>> from vtk import *
>>> c = vtkConeSource()
>>> ivtk.doc(c)            # pops up class documentation for c
>>> ivtk.doc('vtkObject')  # class documentation for vtkObject.
>>> ivtk.config(c)         # configure object with GUI.
```

The module is fairly well documented and one should look at the module for more information. However, the above information should suffice if one wants to start using the module.

4.2.2 The MayaVi `tools` sub-package

MayaVi has a `tools` sub-package that contains useful modules that use or are related to MayaVi. The following modules are present currently.

4.2.2.1 The `imv` package

The `imv` module provides Matlab-like one liners that make it easy to visualize data from the Python interpreter. It currently provides three useful functions. These are partially described below. A simple example is also provided below that. The `imv` module is well documented so please read the documentation strings in the module for more details.

`surf(x, y, f)` This samples the function or 2D array *`f`* along *`x`* and *`y`* and plots a 3D surface.

`view(arr)` Views 2D arrays as a structured points dataset. The view is set to the way we usually think of matrices with (0,0) at the top left of the screen.

`viewi(arr)` Views 2D arrays as a structured points dataset. The data is viewed as an image. This function is meant to be used with large arrays. For smaller arrays one should use the more powerful `view()` function. The implementation of this function is a bit of a hack and many of MayaVi's features cannot be used. For instance you cannot change the lookup table color and expect the color of the image to change.

`sampler(xa, ya, func)` Samples a function (`func`) at an array of ordered points (with equal spacing) and returns an array of scalars as per VTK's requirements for a structured points data set, i.e. x varying fastest and y varying next.

Here is a simple example of what can be done with the `imv` module.

```
>>> from Numeric import *
>>> from mayavi.tools import imv

>>> # surf example.
>>> def f(x, y):
...   return sin(x*y)/(x*y)
>>> x = arange(-5., 5.05, 0.05)
>>> y = arange(-5., 5.05, 0.05)
>>> v = imv.surf(x, y, f)

>>> # view/viewi example.
>>> z1 = fromfunction(lambda i,j:i+j, (128,256))
>>> v1 = imv.view(z1)

>>> z2 = fromfunction(lambda i,j:i+j, (512, 512))
>>> v2 = imv.viewi(z2)
```

Chapter 5

Information for Developers

This chapter covers a few points that developers should note while creating their own filters and modules. The word component in the following discussion refers to either a `Source` or `Module` or a `Filter` or any other class relevant to MayaVi.

5.1 Source Organization

The MayaVi sources are organized in the following fashion.

Base/ The `Base/` directory contains classes that define the basic objects that populate MayaVi (in `Objects.py`). It also contains files that define the `ModuleManager` and the `DataVizManager`.

Common.py This contains useful functions and classes that are commonly used. It also contains some variables that are used by all the other source.

Filters/ This directory contains files that define the various filters.

Modules/ This directory contains files that define the various modules.

Main.py This defines the main GUI class for MayaVi.

doc/ This directory contains documentation for MayaVi.

examples/ This directory contains a few simple examples for VTK. Currently a simple VTK data file, a visualization based on this data file and a customized lookup table are included.

tools/ This directory is actually a sub-package of MayaVi. It contains useful modules that use MayaVi or are useful along with MayaVi. The packages in the tools are described in the tools section.

vtkPipeline/ This directory contains files for the vtkPipeline browser and associated tools. These include a Tkinter based pipeline browser for VTK, a segmented pipeline browser, a simple pickler for VTK objects and a Tkinter based VTK object GUI configurator. The pipeline browser can be used independently of MayaVi.

5.2 Extending MayaVi

5.2.1 User defined sources, modules and filters

The easiest way to extend MayaVi is to create a directory that contains three sub-directories called `Sources/`, `Modules/` and a `Filters/` directory inside where user defined sources, modules and filters can be stored. You don't need to create an `__init__.py` file in these directories. This directory should then be added to the MayaVi "search path" which may be set by using the **Options/Preferences** menu. The search path is a ':'-separated string and is specified like the PYTHONPATH. '~', '~user' and '$VAR' are all expanded. Note that after changing the search path you'll have to restart MayaVi.

These user defined modules and filters will be made available inside the **User** sub-menu of the **File/Open**, **Module** and **Filter** menus respectively. These sources, modules and filters can be used from the command line or from a Python interpreter session by using 'User.SourceName' (sources cannot

be specified from the command line but may be specified in a Python script), 'User.ModuleName' or 'User.FilterName'. When creating user defined sources or modules or filters make sure that the name of the module is the same as the name of the class that defines the particular object.

It is likely that the first thing you will do to test out this functionality is to copy a standard source, module or filter into your directory, rename it and test with it. Beware of some of the following points before you do that.

1. Make certain that the name of the class that defines the source/module/filter is identical with the name of the module that contains it. So if you have a module called `MySuperModule.py` the class that defines the module *must* be called `MySuperModule`. The same applies to the sources and filters as well.

2. The standard Modules/filters and sources are a little special and will typically contain code that looks like this.

```
import Base.Objects, Common
# ...
class SomeModule(Base.Objects.Module):
# ...
```

This must be changed to the following.

```
from mayavi import Common
from mayavi.Base import Objects
# ...
class SomeModule(Objects.Module):
# ...
```

Essentially, the Python module defining your object should be "import-able" from the Python interpreter even without MayaVi running.

5.2.2 General guidelines and information

This section lists a set of things that developers should know before extending MayaVi by implementing new `Module`s, `Source`s and `Filter`s.

1. The `print_err` function takes a string and prints it out on a GUI message box.

2. The `get_relative_file_name(base_f_name, f_name)` function returns a file name for the given `f_name` relative to the `base_f_name`. This is useful when one stores file names in the configuration.

3. The `get_abs_file_name(base_f_name, rel_f_name)` function returns an absolute file name given a base file name and another file name relative to the base name. This is useful to load a stored file name.

4. The `debug` function prints a string on the debug window and on `sys.stdout`. This is used to print debug messages in all the MayaVi code.

5. The `config` variable is an instance of class that contains the configuration preferences of the user. This is used to initialize default colors etc.

6. The `state` variable is instance of the `AppState` class. The `state.busy()` `state.idle()` members are used to change the state of the MayaVi application.

1. As with any code the best way to learn how the classes and functions are organized and work is to look through the source. What follows are a few hints that might be useful when you develop code for MayaVi.

2. Before writing a new component please look at the the `Module`, `Filter` and `Source` classes defined in `Base/Objects.py`. Also look at their parent classes. A lot of useful functionality is provided there and reading this may prevent unnecessary code replication.

3. The Module and Filter menus are created dynamically from the files in the `Modules` and `Filters` directories. All the files there except the `__init__.py` *will* be recognized as modules or filters respectively. So do not put any files there that are neither modules nor filters.

4. Make sure the name of the new Module/Filter or Source subclass and the name of the file that defines the Module subclass are the same. For instance the `Axes` module is defined in a file called `Axes.py`.

5. As far as possible try to follow my style of coding. The naming convention that I have followed is to name all VTK like methods in the usual VTK way (like GetOutput). All other methods are named like so get_vtk_object. This has been done to differentiate between the two. All class names have the first letter of each word capitalized for example as in `VtkDataReader` and `ExtractVectorComponents`.

6. Please try to document your classes and functions.

7. While writing any component please try to make the particular component "import safe" by not doing either of the first two and using the last way of importing modules. Doing this makes it possible to reload modules on the fly.

```
# Don't do this:
from SomeThing import *
# or the following:
from SomeThing import SomeClass
# Use this instead.
import SomeThing
```

8. While developing a component it is possible to test the component by running MayaVi and then retest it by reloading the modules from the **Options**.

9. The `Common.py` module provides a few useful classes and functions. Some of them are as follows:

10. Every component has to be able to save its state to an ASCII file and be able to reload its state from the saved file. The `save_config` and `load_config` functions should implement this. This should be tested properly.

Happy hacking!