

3D Data visualization with Mayavi

Prabhu Ramachandran

Department of Aerospace Engineering
IIT Bombay

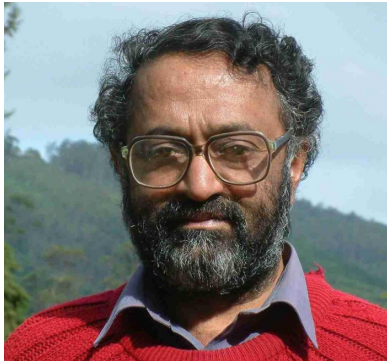
SciPy.in 2012,
December 27,
IIT Bombay.



In memory of John Hunter,



Kenneth Gonsalves,



and Raj Mathur.



Objectives

At the end of this session you will be able to:

- 1 Use `mplab` effectively to visualize numpy array data of various kinds
- 2 Apply some of mayavi's advanced features

Outline

1 Quick introduction to Mayavi

2 `m1ab`



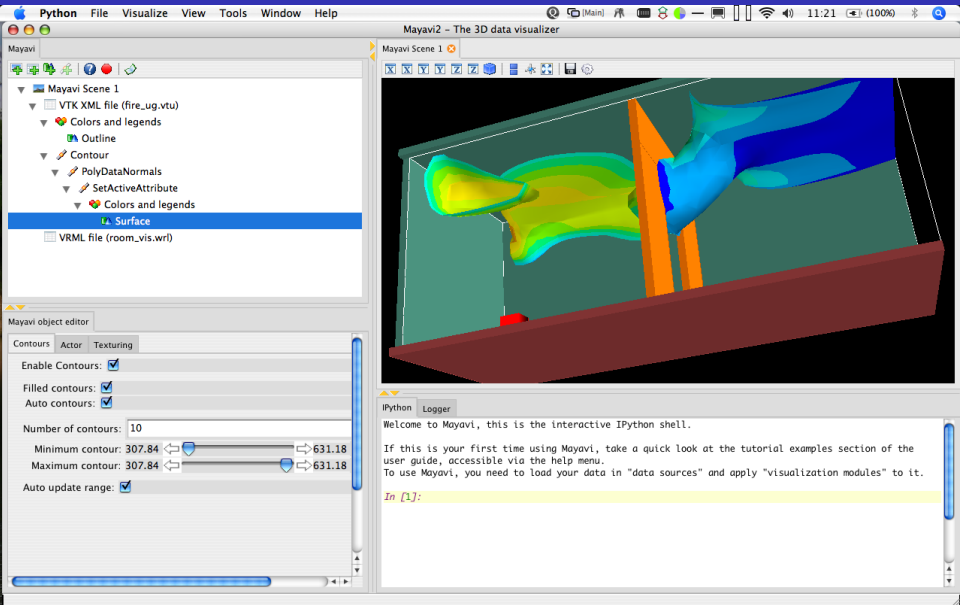
Outline

1 Quick introduction to Mayavi

2 `mlab`



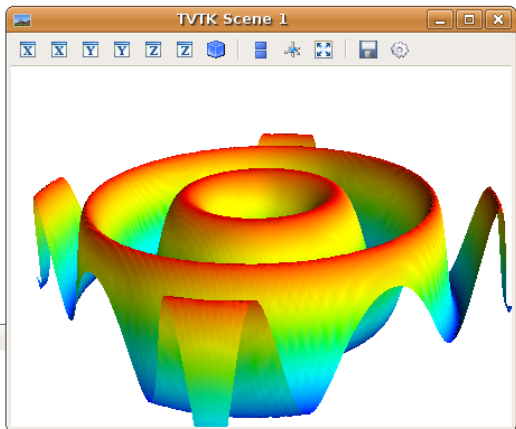
Overview of features



The screenshot displays the Mayavi2 application window, titled "Mayavi2 - The 3D data visualizer". The interface is divided into several panels:

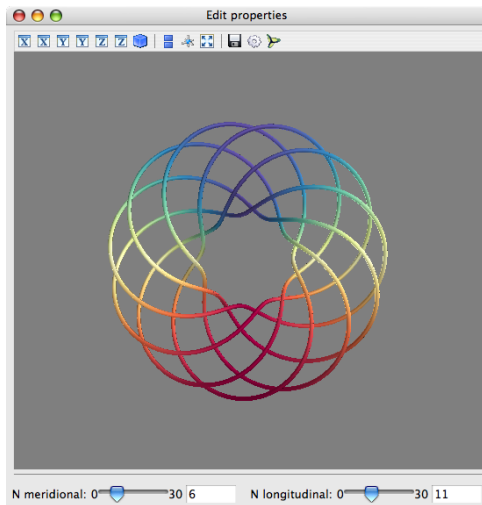
- Mayavi Scene 1:** The main 3D view area showing a ship's hull with a color-coded surface. The hull is primarily red, with a yellow and green area on the upper part, and a blue area on the right side. Two orange vertical structures are visible on the deck.
- Mayavi object editor:** A panel on the left side containing a tree view of the scene's objects. The "Surface" object is selected and highlighted in blue. Other objects include "VTK XML file (fire_ug.vtu)", "Colors and legends", "Outline", "Contour", "PolyDataNormals", "SetActiveAttribute", "Colors and legends", and "VRML file (room_vis.wrl)".
- Contours:** A panel below the object editor with tabs for "Contours", "Actor", and "Texturing". It includes several settings:
 - Enable Contours:
 - Filled contours:
 - Auto contours:
 - Number of contours: 10
 - Minimum contour: 307.84 (with a slider and arrow pointing right)
 - Maximum contour: 631.18 (with a slider and arrow pointing left)
 - Auto update range:
- iPython:** A panel at the bottom right containing a text area with the following text:

```
Welcome to Mayavi, this is the interactive IPython shell.  
  
If this is your first time using Mayavi, take a quick look at the tutorial examples section of the user guide, accessible via the help menu.  
To use Mayavi, you need to load your data in "data sources" and apply "visualization modules" to it.  
  
In [1]:
```

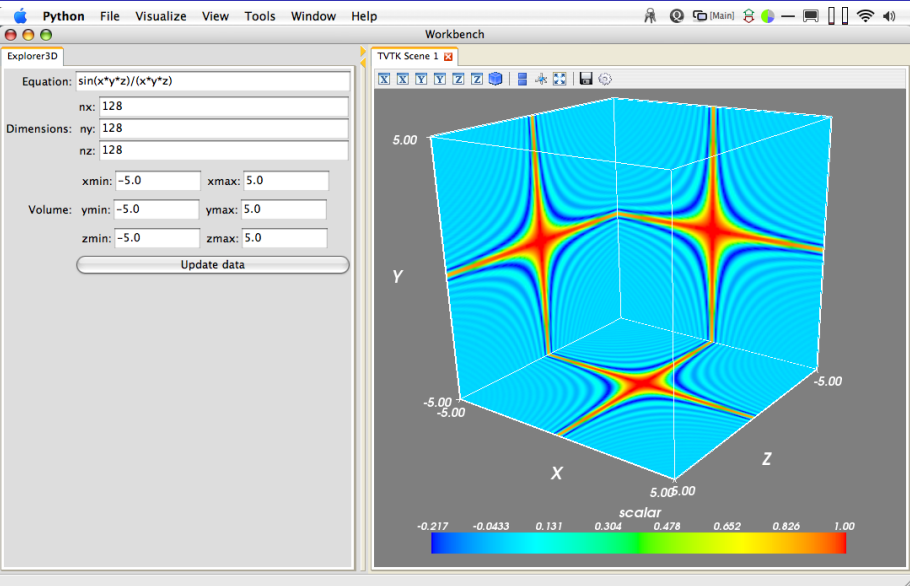



```
Terminal  
  
In [1]: from numpy import *  
In [2]: x, y = mgrid[-3:3:100j, -3:3:100j]  
In [3]: z = sin(x**2 + y**2)  
In [4]: from enthought.mayavi import mlab  
In [5]: mlab.surf(x, y, z)
```

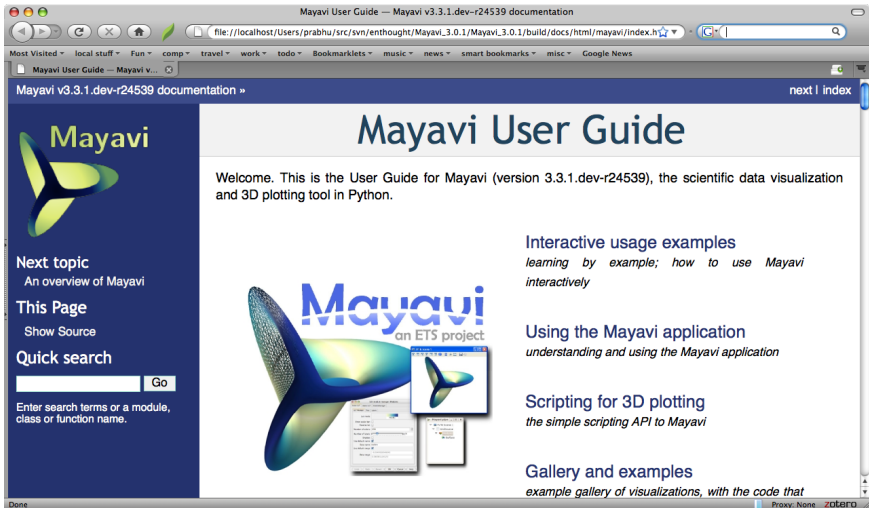
Live in your dialogs



Mayavi in applications



Exploring the documentation




The screenshot shows a web browser window with the title "Mayavi User Guide — Mayavi v3.3.1.dev-r24539 documentation". The address bar shows the file path: `file:///localhost/Users/prabhu/src/svn/enthought/Mayavi_3.0.1/Mayavi_3.0.1/build/docs/html/mayavi/index.html`. The browser's "Most Visited" list includes "local stuff", "Fun", "comp", "travel", "work", "todo", "Bookmarks", "music", "news", "smart bookmarks", "misc", and "Google News". The page content is as follows:

Mayavi v3.3.1.dev-r24539 documentation » next | index

Mayavi User Guide

Welcome. This is the User Guide for Mayavi (version 3.3.1.dev-r24539), the scientific data visualization and 3D plotting tool in Python.




Next topic
An overview of Mayavi

This Page
Show Source

Quick search

Enter search terms or a module, class or function name.



- [Interactive usage examples](#)
learning by example; how to use Mayavi interactively
- [Using the Mayavi application](#)
understanding and using the Mayavi application
- [Scripting for 3D plotting](#)
the simple scripting API to Mayavi
- [Gallery and examples](#)
example gallery of visualizations, with the code that

Done Proxy: None zotero

Other features

- Easy customization
- Offscreen animations
- Automatic script generation
- Powerful command line options

Summary

- `http://code.enthought.com/projects/mayavi`
- Uses VTK (`www.vtk.org`)
- BSD license
- Linux, win32 and Mac OS X
- Highly scriptable
- Embed in Traits UIs (wxPython and PyQt4)
- Envisage Plugins
- Debian/Ubuntu/Fedora
- **Pythonic**

Outline

- 1 Quick introduction to Mayavi
- 2 `m1ab`



Overview

- Simple
- Convenient
- Full-featured



Getting started

Vanilla:

```
$ ipython --gui=wx
```

with Pylab:

```
$ ipython --pylab=wx
```

Using mlab:

```
>>> from enthought.mayavi import mlab
```

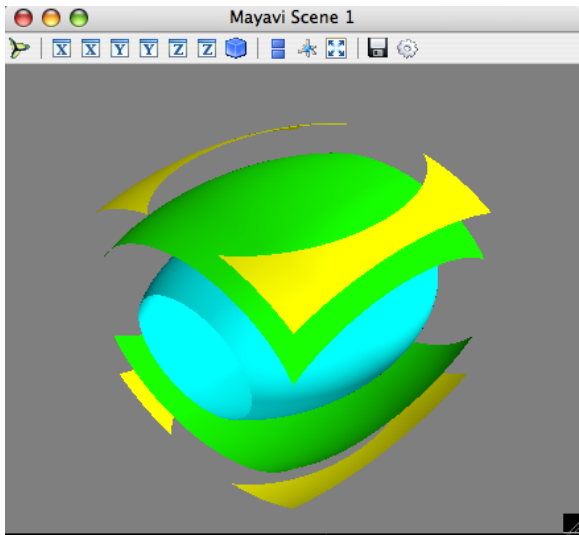
Try these:


```
>>> mlab.test_<TAB>
```

```
>>> mlab.test_contour3d()
```

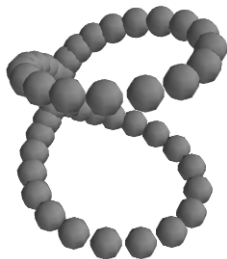
```
>>> mlab.test_contour3d??
```

Exploring the view



- Mouse
- Keyboard
- Toolbar
- Mayavi icon 

0D data



```
>>> from numpy import *
>>> t = linspace(0, 2*pi, 50)
>>> u = cos(t)*pi
>>> x, y, z = sin(u), cos(u), sin(t)

>>> mlab.points3d(x, y, z)
```

Changing how things look

Clearing the view

```
>>> mlab.clf()
```

IPython is your friend!

```
>>> mlab.points3d?
```

- Extra argument: Scalars
- Keyword arguments
- UI

```
>>> mlab.points3d(x, y, z, t,  
                  scale_mode='none')
```

Changing how things look

Clearing the view

```
>>> mlab.clf()
```

IPython is your friend!

```
>>> mlab.points3d?
```

- Extra argument: Scalars
- Keyword arguments
- UI

```
>>> mlab.points3d(x, y, z, t,  
                  scale_mode='none')
```

Changing how things look

Clearing the view

```
>>> mlab.clf()
```

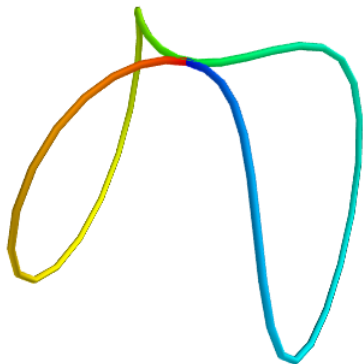
IPython is your friend!

```
>>> mlab.points3d?
```

- Extra argument: Scalars
- Keyword arguments
- UI

```
>>> mlab.points3d(x, y, z, t,  
                  scale_mode='none')
```

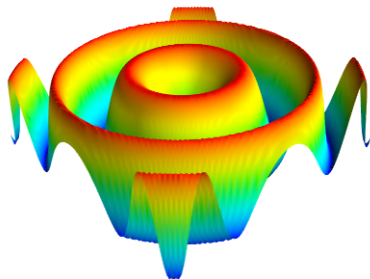
1D data



```
>>> mlab.plot3d(x, y, z, t)
```

Plots lines between the points

2D data



```
>>> x, y = mgrid[-3:3:100j, -3:3:100j]  
>>> z = sin(x*x + y*y)
```

```
>>> mlab.surf(x, y, z)
```

Assumes the points are rectilinear

2D data: `mlab.mesh`

```
>>> mlab.mesh(x, y, z)
```

Points needn't be regular

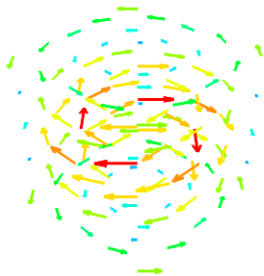
```
>>> phi, theta = numpy.mgrid[0:pi:20j,
...                           0:2*pi:20j]
>>> x = sin(phi)*cos(theta)
>>> y = sin(phi)*sin(theta)
>>> z = cos(phi)
>>> mlab.mesh(x, y, z,
...           representation='wireframe')
```

3D data



```
>>> x, y, z = ogrid[-5:5:64j ,  
...               -5:5:64j ,  
...               -5:5:64j ]  
>>> mlab.contour3d(x*x*0.5 + y*y +  
                   z*z*2)
```

3D vector data: mlab.quiver3d



```
>>> mlab.test_quiver3d()
```

```
obj = mlab.quiver3d(x, y, z, u, v, w)
```

3D vector data: *mlab.flow*

```
>>> x, y, z = mgrid[-2:3, -2:3, -2:3]
>>> r = sqrt(x**2 + y**2 + z**4)
>>> u = y*sin(r)/(r+0.001)
>>> v = -x*sin(r)/(r+0.001)
>>> w = zeros_like(z)
>>> obj = mlab.flow(x, y, z, u, v, w,
                    seedtype='plane')
>>> obj.stream_tracer.integrator_type = \
    'runge_kutta45'
```

Exercise: Lorenz equation

$$\begin{aligned}\frac{dx}{dt} &= s(y - x) \\ \frac{dy}{dt} &= rx - y - xz \\ \frac{dz}{dt} &= xy - bz\end{aligned}$$

Let $s = 10, r = 28, b = 8./3$.

Region of interest

```
x, y, z = mgrid[-50:50:20j, -50:50:20j, -10:60:20j]
```

Use `mlab.quiver3d`

Solution

```
def lorenz(x, y, z, s=10., r=28., b=8./3.):  
    u = s*(y-x)  
    v = r*x - y - x*z  
    w = x*y - b*z  
    return u, v, w
```

```
x, y, z = mgrid[-50:50:20j, -50:50:20j,  
                -10:60:20j]  
u, v, w = lorenz(x, y, z)
```

```
mlab.quiver3d(x, y, z, u, v, w,  
              scale_factor=0.01,  
              mask_points=5)
```

```
mlab.show()
```

Issues and solutions

- Basic visualization: not very useful
- Tweak parameters:
mask_points, scale_factor
- Explore parameters on UI
- mlab.flow is a lot better!

Good visualization involves work

Other utility functions

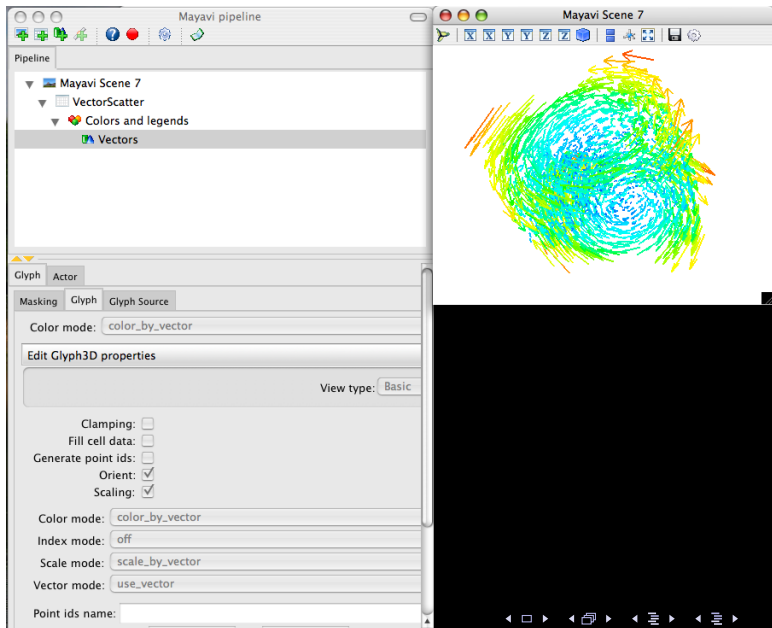
- `gcf`: get current figure
- `savefig`, `figure`
- `axes`, `outline`
- `title` , `xlabel` , `ylabel` , `zlabel`
- `colorbar`, `scalarbar`, `vectorbar`
- `show`: Standalone mlab scripts
- Others, see UG

Can we do more?

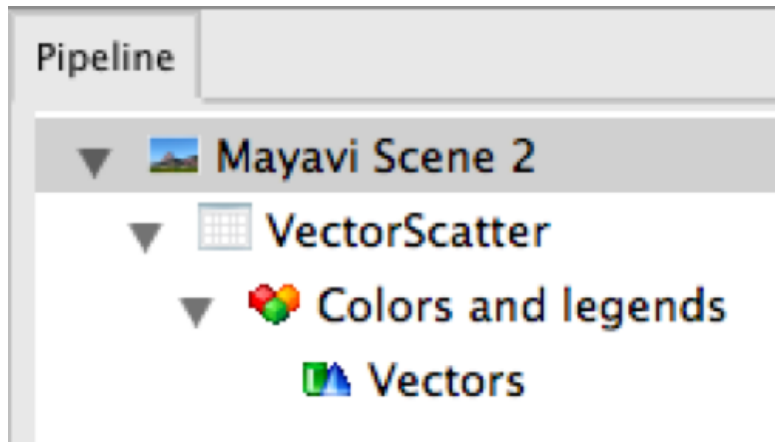
Yes!

```
quiver3d(x, y, z,  
         u, v, w,  
         scale_factor=0.01,  
         mask_points=5)
```

Looking inside



The pipeline



Mayavi Engine

TVTK Scene

Source

Filter

ModuleManager

Lookup tables

List of Modules

Changing the pipeline

On UI

- Right click on node
- drag drop

Script

- Or use **mlab.pipeline**
- Example: `mlab.pipeline.outline()`
- `obj.remove()`

Exercise

```
>>> mlab.test_quiver3d()
```

Hide vectors, add a Vector Cut Plane

```
>>> mlab.test_flow()
```

Add a Vector Cut Plane

Can also use the Lorenz example

Exercise

```
>>> mlab.test_quiver3d()
```

Hide vectors, add a Vector Cut Plane

```
>>> mlab.test_flow()
```

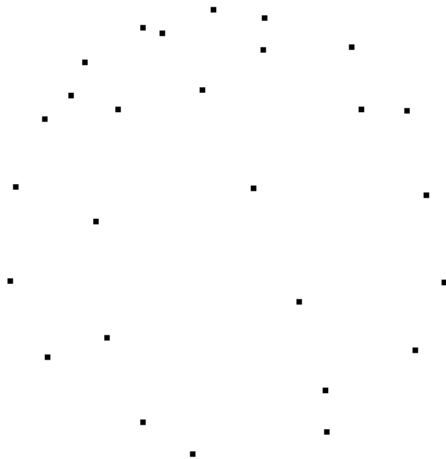
Add a Vector Cut Plane

Can also use the Lorenz example

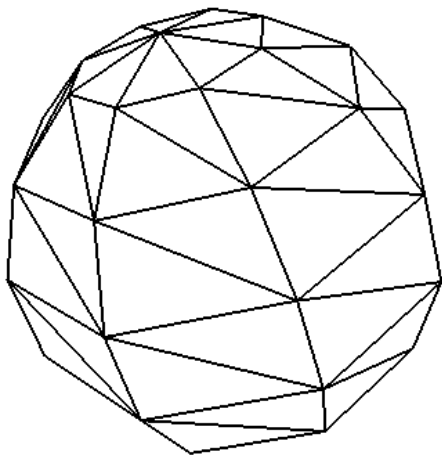
Surprised?

So what is the problem?

Points?



Curve?



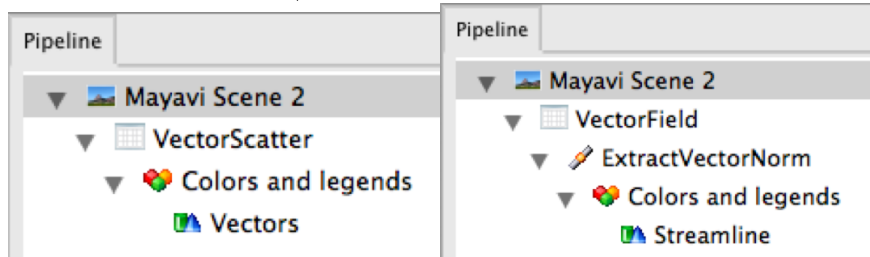
Surface?



Interior of sphere?



Quiver v/s Flow



Recap

- `mlab` gets you started
- Pipeline and data flow
- Datasets are important

Changing the pipeline

On UI

- Right click on node
- drag drop

Script

- Or use **mlab.pipeline**
- Example: `mlab.pipeline.outline()`
- `obj.remove()`

mlab and Mayavi2?

- `mlab` is just a thin layer over the Mayavi OO API
- `mlab` commands return mayavi objects

Exercise

- 1 Start with flow for the Lorenz system
- 2 Now extract the vector norm (use a filter)
- 3 Plot iso-contours of this
- 4 Figure out how to do this from the UI and `mlab.pipeline`

85



So how do you make a fancier script?

Use script recording

Demo

So how do you make a fancier script?

Use script recording

Demo

Animating data

```
>>> s = mlab.flow(x, y, z, u, v, w)
>>> s.mlab_source.u = u*z
```

- **mlab_source.set**: multiple attributes
- If you change the shape of the arrays use the **reset** method

Setting the view

```
>>> print mlab.view()  
>>> mlab.view(azimuth=None,  
              elevation=None,  
              distance=None,  
              focalpoint=None)
```


Thank you!