3D Data visualization with Mayavi and TVTK

Prabhu Ramachandran

Department of Aerospace Engineering
IIT Bombay

Advanced tutorials at SciPy10
Austin, Texas
June 29, 2010
Objectives

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

1. Use \texttt{mlab} effectively to visualize numpy array data of various kinds
2. Apply some of mayavi’s advanced features
3. Embed mayavi visualizations in your dialogs
4. Create TVTK datasets for more effective visualization (if time permits)
Outline

1. Quick introduction to Mayavi
2. mlab
3. Embedding mayavi
4. Creating and working with datasets
Outline

1. Quick introduction to Mayavi
2. mlab
3. Embedding mayavi
4. Creating and working with datasets
<table>
<thead>
<tr>
<th>Name</th>
<th>Role and Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prabhu Ramachandran</td>
<td>2001 – Creator and lead</td>
</tr>
<tr>
<td>Gaël Varoquaux</td>
<td>2007 – mlab, documentation, usability</td>
</tr>
<tr>
<td>Enthought Inc.</td>
<td>ETS, Hosting, support, sprints, initial funding, distribution</td>
</tr>
</tbody>
</table>
Mayavi-1.x: 2001
TVTK: 2004, Enthought
Mayavi2: 2005, Enthought, IITB
2008: Mayavi sprint at Austin
Overview of features

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]:

...
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

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.

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
Other features

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

- **Uses VTK** ([www.vtk.org](http://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. mlab
3. Embedding mayavi
4. Creating and working with datasets
Overview

- Simple
- Convenient
- Full-featured
Getting started

**Vanilla:**

$ ipython --wthread

**with Pylab:**

$ ipython --pylab --wthread
Using mlab:

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

**Try these:**

```python
>>> mlab.test_ <TAB>
>>> mlab.test_contour3d()
>>> mlab.test_contour3d??
```
Exploring the view

- Mouse
- Keyboard
- Toolbar
- Mayavi icon
>>> 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

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

IPython is your friend!

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

- Extra argument: Scalars
- Keyword arguments
- UI

```python
>>> 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

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

IPython is your friend!

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

- Extra argument: Scalars
- Keyword arguments
- UI

```python
>>> mlab.points3d(x, y, z, t,
        scale_mode=’none’)
```
1D data

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

Plots lines between the points
2D data

```python
>>> x = 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`

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

Points needn't be regular

```
$$3D \ data$$

>>> 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

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

obj = mlab.quiver3d(x, y, z, u, v, w)
```
Exercise: Lorenz equation

\[
\begin{align*}
\frac{dx}{dt} &= s(y - x) \\
\frac{dy}{dt} &= rx - y - xz \\
\frac{dz}{dt} &= xy - bz
\end{align*}
\]

Let \( s = 10, r = 28, b = 8/3. \)

Region of interest

\( x, y, z = \text{mgrid}[-50:50:20j, -50:50:20j, -10:60:20j] \)

Use \text{mlab.quiver3d}
Solution

```python
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
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
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
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
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
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
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 Scene 2
- Vector Scatter
- Colors and legends
- Vectors
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

```python
>>> 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

```python
>>> 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?
Datasets

Quiver v/s Flow

Get back to this later!
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
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

```python
>>> 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

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

- Embed Mayavi into a dialog box
- Use traits to wire up everything
- Full power of mayavi at your disposal
Simple example

120
Exercise: Lorenz trajectory

Use the provided skeleton script

1. Create a simple UI to show a trajectory
2. Create sliders to change the position of the initial condition
3. Create a UI element to change the integration time
Outline

1. Quick introduction to Mayavi
2. mlab
3. Embedding mayavi
4. Creating and working with datasets
Datasets are fundamental to doing visualization correctly

**Motivational problem**

Atmospheric data of temperature over the surface of the earth.

Let temperature \((T)\) vary linearly with height \((z)\):

\[ T = 288.15 - 6.5z \]
Simple solution

```python
lat = linspace(-89, 89, 37)
lon = linspace(0, 360, 37)
z = linspace(0, 100, 11)

x, y, z = mgrid[0:360:37j, -89:89:37j, 0:100:11j]
t = 288.15 - 6.5*z
mlab.contour3d(x, y, z, t)
mlab.outline()
mlab.colorbar()
```
Simple solution

```python
lat = linspace(-89, 89, 37)
lon = linspace(0, 360, 37)
z = linspace(0, 100, 11)

x, y, z = mgrid[0:360:37j, -89:89:37j, 0:100:11j]

t = 288.15 - 6.5*z
mlab.contour3d(x, y, z, t)
mlab.outline()
mlab.colorbar()
```
What happens underneath?

```python
P = mlab.pipeline
src = P.scalar_field(x, y, z, t)
iso = P.iso_surface(src)
# Try this.
print src
```
The underlying dataset

```python
from enthought.tvtk.api import tvtk
orig = (0, -90, 0)
spacing = (10, 5, 10)
dims = (37, 37, 11)
id = tvtk.ImageData(origin=orig,
                   spacing=spacing,
                   dimensions=dims)

id.point_data.scalars = t.T.flatten()
id.point_data.scalars.name = 'T'
# View it.
src = P.add_dataset(id)
iso = P.iso_surface(src)
```
The general idea

- Specify the points (explicitly or implicitly)
- Specify the connectivity between the points (explicit/implicit)
- The connectivity lets you build “cells” that break the space into pieces
- Specify “attribute” data at the points or cells
Types of datasets

- Implicit topology (structured):
  - Image data (structured points)
  - Rectilinear grids
  - Structured grids

- Explicit topology (unstructured):
  - Polygonal data (surfaces)
  - Unstructured grids
Implicit versus explicit topology

- Implicit topology associated with points:
  - The X co-ordinate increases first, Y next and Z last
- Easiest example: a rectangular mesh
- Non-rectangular mesh certainly possible
On a sphere?

lon, lat, ht = x*pi/180, (90+y)*pi/180, z
r = (1+0.005*ht)
tmp = r*sin(lat)

# Points on the sphere
x, y, z = tmp*cos(lon), tmp*sin(lon),
r*cos(lat)

pts = empty(x.shape + (3,), dtype=float)
pts[... ,0] = x
pts[... ,1] = y
pts[... ,2] = z

# Reorder the points/scalars for VTK
pts = pts.transpose(2, 1, 0, 3).copy()
pts.shape = pts.size/3, 3
t = t.T.copy()
sg = tvtk.StructuredGrid()
sg.dimensions = x.shape
sg.points = pts
sg.point_data.scalars = t.ravel()
sg.point_data.scalars.name = 'T'

P = mlab.pipeline
src = P.add_dataset(sg)
P.grid_plane(src)
P.iso_surface(src, contours=1)

mlab.show()
# Save a dataset to disk.

```python
from enthought.tvtk.api import write_data
write_data(dataset, fname)
```

# Open back the data.

```python
mlab.pipeline.open(fname)
```

Try right clicking a node!
Unstructured grids

- Explicit topology specification
- Specified via connectivity lists
- Different number of neighbors, different types of cells
from enthought.tvtk.api import tvtk

# The points in 3D.
points = array([[0, 0, 0], [1, 0, 0], [0, 1, 0], [0, 0, 1]], 'f')

# Connectivity via indices to the points.
triangles = array([[0, 1, 3], [0, 3, 2], [1, 2, 3], [0, 2, 1]])

# Creating the data object.

mesh = tvtk.PolyData()
mesh.points = points  # the points
mesh.polys = triangles  # triangles for connectivity

# For lines use: mesh.lines = lines

# Now create some point data.
temperature = array([10, 20, 20, 30], 'f')
mesh.point_data.scalars = temperature
mesh.point_data.scalars.name = 'temperature'
from enthought.tvtk.api import tvtk

# The points in 3D.
points = array([[0,0,0], [1,0,0], [0,1,0], [0,0,1]], 'f')

# Connectivity via indices to the points.
triangles = array([[0,1,3], [0,3,2], [1,2,3], [0,2,1]])

# Creating the data object.

mesh = tvtk.PolyData()
mesh.points = points  # the points
mesh.polys = triangles  # triangles for connectivity

# For lines use: mesh.lines = lines

# Now create some point data.
temperature = array([10, 20, 20, 30], 'f')

mesh.point_data.scalars = temperature
mesh.point_data.scalars.name = 'temperature'
Summary

Datasets

Points in 3D

Vector

Scalar

On the points

In the cells

Implicit

Explicit

Uconnected

Connected?
Advanced features

Command line arguments, timeseries, scripting
Thank you!