3D visualization with TVTK and MayaVi2

Prabhu Ramachandran

Department of Aerospace Engineering
IIT Bombay

19, July 2007

Outline

Outline

What is visualization?

Visualize: dictionary meaning (Webster)

1. To make visual, or visible.
2. to see in the imagination; to form a mental image of.

Visualization: graphics

> *Making a visible presentation of numerical data, particularly a graphical one. This might include anything from a simple X-Y graph of one dependent variable against one independent variable to a virtual reality which allows you to fly around the data.*

– from the Free On-line Dictionary of Computing

Introduction

- Cross-platform 2D/3D visualization for scientists and engineers
- Almost all 2D plotting: matplotlib and Chaco
- More complex 2D/3D visualization
    - Unfortunately, not as easy as 2D (yet)
- Most scientists:
    - not interested in details of visualization
    - need to get the job done ASAP
    - have enough work as it is!
- Fair enough: *not often do we need to know internals of matplotlib!*

Outline

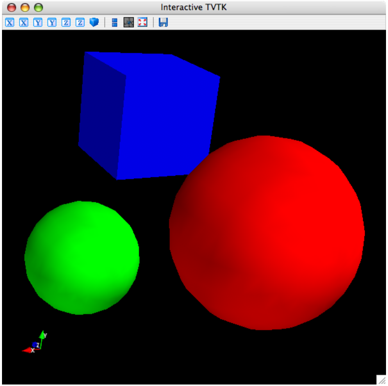
Introduction to `visual`

- `from enthought.tvtk.tools import visual`
- `visual`: based on VPython's visual but uses VTK
- Makes it very easy to create simple 3D visualizations
- API shamelessly stolen fron VPython (and for good reason)
- Implemented by Raashid Baig
- Differences from VPython:
    - Slight API differences from VPython
    - Does not require special interpreter
    - Works with `ipython -wthread`
    - Uses traits
    - Does not rely on special interpreter for looping
    - Much(?) Slower

Simple visual example

Example code

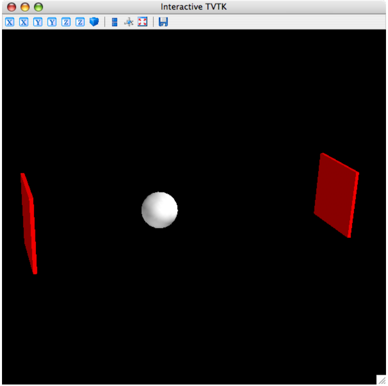
```
from enthought.tvtk.tools import visual
s1 = visual.sphere(pos=(0,0,0),
                   radius=0.5,
                   color=visual.color.red)
s2 = visual.sphere(pos=(1,0,0),
                   radius=0.25,
                   color=visual.color.green)
b = visual.box(pos=(0.5,0.5,0.5),
               size=(0.5, 0.5, 0.5),
               representation='s',
               color=visual.color.blue)
```



visual: bouncing ball

Example code

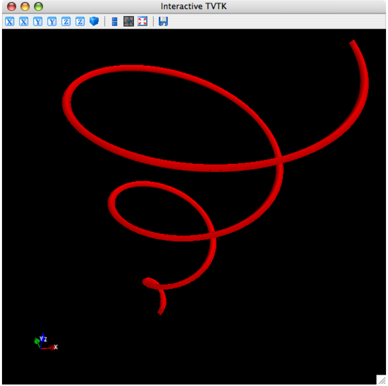
```
from enthought.tvtk.tools import visual
lwall = visual.box(pos=(-4.5, 0,0),
                   size=(0.1,2,2),
                   color=visual.color.red)
rwall = visual.box(pos=(4.5, 0,0),
                   size=(0.1,2,2),
                   color=visual.color.red)
ball = visual.sphere(pos=(0,0,0), radius=0.5,
                     t=0.0, dt=0.5)
ball.v = visual.vector(1.0, 0.0, 0.0)
def anim():
    ball.t = ball.t + ball.dt
    ball.pos = ball.pos + ball.v*ball.dt
    if not (4.0 > ball.x > -4.0):
        ball.v.x = -ball.v.x

# Iterate the function without blocking the GUI
# first arg: time period to wait in millisecs
iter = visual.iterate(100, anim)
# Stop, restart
iter.stop_animation = True
iter.start_animation = True
```



visual: a curve

Example code

```
from enthought.tvtk.tools import visual
import numpy
t = numpy.linspace(0, 6*pi, 1000)
t = numpy.linspace(0, 6*numpy.pi, 1000)
pts = numpy.zeros((1000, 3))
pts[:,0] = 0.1*t*numpy.cos(t)
pts[:,1] = 0.1*t*numpy.sin(t)
pts[:,2] = 0.25*t
c = visual.curve(points=pts, color=(1,0,0),
                 radius=0.05)
# Append (or extend) points.
c.append([2,0, 0])
# Remove points.
c.points = c.points[:-1]
```



More `visual` demos

- More examples available in `tvtk/examples/visual` directory
- More demos!

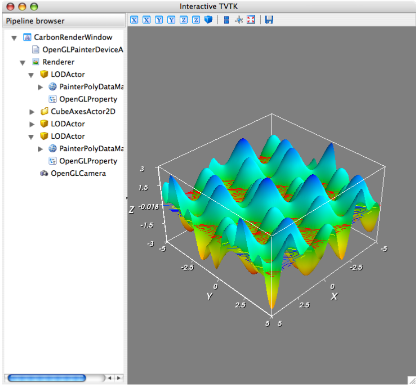
Introduction to `tvtk.tools.mlab`

- `enthought.tvtk.tools.mlab`
- Provides Matlab like 3d visualization conveniences
- Visual OTOH is handy for simpler graphics
- API mirrors that of Octaviz: `http://octaviz.sf.net`
- Place different Glyphs at points
- 3D lines, meshes and surfaces
- Titles, outline

tvtk.mlab: example

Example code

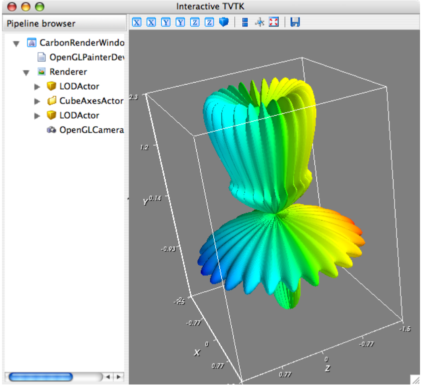
```
from enthought.tvtk.tools import mlab
from scipy import *
def f(x, y):
    return sin(x+y) + sin(2*x - y) + cos(3*x+4*y)

x = linspace(-5, 5, 200)
y = linspace(-5, 5, 200)
fig = mlab.figure()
s = mlab.SurfRegularC(x, y, f)
fig.add(s)
fig.pop()
x = linspace(-5, 5, 100)
y = linspace(-5, 5, 100)
s = mlab.SurfRegularC(x, y, f)
fig.add(s)
```



tvtk.mlab: example

Example code

```
from enthought.tvtk.tools import mlab
from scipy import pi, sin, cos, mgrid
# Make the data
dphi, dtheta = pi/250.0, pi/250.0
[phi,theta] = mgrid[0:pi+dphi*1.5:dphi,
                    0:2*pi+dtheta*1.5:dtheta]
m0 = 4; m1 = 3; m2 = 2; m3 = 3;
m4 = 6; m5 = 2; m6 = 6; m7 = 4;
r = sin(m0*phi)**m1 + cos(m2*phi)**m3 +
    sin(m4*theta)**m5 + cos(m6*theta)**m7
x = r*sin(phi)*cos(theta)
y = r*cos(phi)
z = r*sin(phi)*sin(theta);
# Plot it!
fig = mlab.figure()
s = mlab.Surf(x, y, z, z)
fig.add(s)
```



More `tvtk.tools.mlab` demos

- More examples available in `tvtk/tools/mlab.py` source
- Demos!

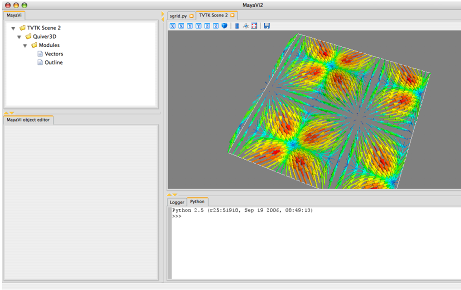
Introduction to `mayavi.tools.mlab`

- `enthought.mayavi.tools.mlab`
- Provides Matlab like 3d visualization conveniences
- This one uses MayaVi
- Makes it more powerful (easier to use and extend) than the tvtk version
- Current API somewhat similar to `tvtk.mlab`
- API still under development: will change for the better
- Gael Varoquaux helping with current development/design

mayavi.tools.mlab example

Example code

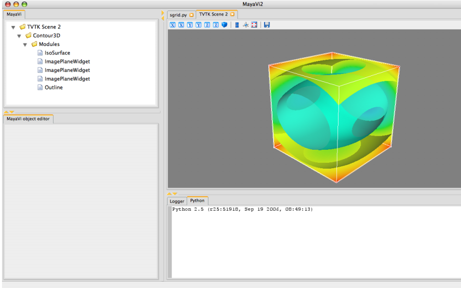
```
dims = [32, 32]
xmin, xmax, ymin, ymax = [-5,5,-5,5]
x, y = scipy.mgrid[xmin:xmax:dims[0]*1j,
                   ymin:ymax:dims[1]*1j]
x = x.astype('f')
y = y.astype('f')
u = cos(x)
v = sin(y)
w = scipy.zeros_like(x)
quiver3d(x, y, w, u, v, w)
# Show an outline.
outline()
```



mayavi.tools.mlab example

Example code

```
dims = [64, 64, 64]
xmin, xmax, ymin, ymax, zmin, zmax = \
                        [-5,5,-5,5,-5,5]
x, y, z = numpy.ogrid[xmin:xmax:dims[0]*1j,
                      ymin:ymax:dims[1]*1j,
                      zmin:zmax:dims[2]*1j]
x, y, z = [t.astype('f') for t in (x, y, z)]

scalars = x*x*0.5 + y*y + z*z*2.0
# Contour the data.
contour3d(scalars, contours=4,
          show_slices=True)
# Show an outline.
outline()
```



More `mayavi.tools.mlab` demos

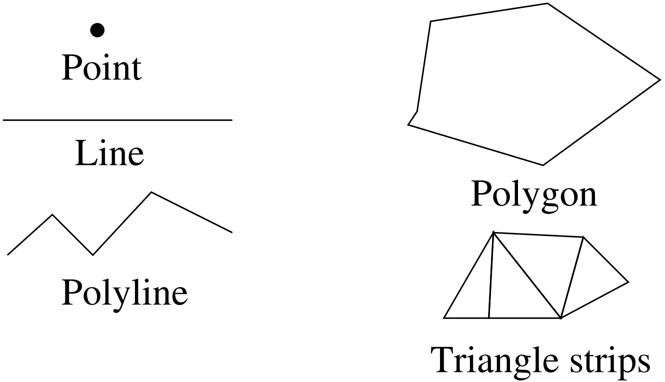
- More examples available in `mayavi/tools/mlab.py` source
- Demos!

Outline

Introduction to graphics

- Visually represent things
- Fundamental elements: graphics primitives
- Two kinds: vector and raster graphics
- Vector: lines, circles, ellipses, splines, etc.
- Raster: pixels – dots
- Rendering: conversion of data into graphics primitives
- Representation on screen: 3D object on 2D screen

Graphics primitives



Physical Vision

- Objects, their properties (color, opacity, texture etc.)
- Light source
- Light rays
- Eye
- Brain: interpretation, vision

Rendering

- Ray tracing
  - Image-order rendering
  - Slow (done in software)
  - More realistic
- Object order rendering
  - Faster
  - Hardware support
  - Less realistic but interactive
- Surface rendering: render surfaces (lines, triangles, polygons)
- Volume rendering: rendering “volumes” (fog, X-ray intensity, MRI data etc.)

Lights, camera, action!

- Colors: RGB (Red-Green-Blue), HSV (Hue-Saturation-Value)
- Opacity: 0.0 - transparent, 1.0 - opaque
- Lights:
  - Ambient light: light from surroundings
  - Diffuse lighting: reflection of light from object
  - Specular lighting: direct reflection of light source from shiny object
- Camera: position, orientation, focal point and clipping plane
- Projection:
  - Parallel/orthographic: light rays enter parallel to camera
  - Perspective: Rays pass through a point – thus objects farther away appear smaller
- Homogeneous coordinate systems are used (x_h, y_h, z_h, w_h)

Practical aspects

- Bulk of functionality implemented in graphics libraries
- Various 2D graphics libraries (quite easy)
- Hardware acceleration for high performance and interactive usage
- OpenGL: powerful, hardware accelerated, 3D graphics library

Outline

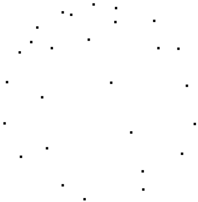
Scientific data

- Numbers from experiments and simulation
- Numbers of different kinds
- Space (1D, 2D, 3D, . . . , n-D)
- Surfaces and volumes
- Topology
- Geometry: CAD
- Functions of various dimensions (what are functions?)
- Scalar, Vector and Tensor fields

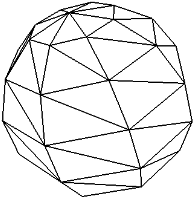
Space

- Linear vector spaces
- The notion of dimensionality
- *Not* the number of components in a vector of the space!
- Maximum number of linearly independent vectors
- Some familiar examples:
  - Point - 0D
  - Line - 1D
  - 2D Surface (embedded in a 3D surface) - 2D
  - Volumes - 3D

Dimensionality example



Points



Wireframe

Surface

Topology

Topology: mathematics

> *A branch of mathematics which studies the properties of geometrical forms which retain their identity under certain transformations, such as stretching or twisting, which are homeomorphic.*

– from the Collaborative International Dictionary of English

Topology: networking

> *Which hosts are directly connected to which other hosts in a network. Network layer processes need to consider the current network topology to be able to route packets to their final destination reliably and efficiently.*

– from the free On-line Dictionary of Computing

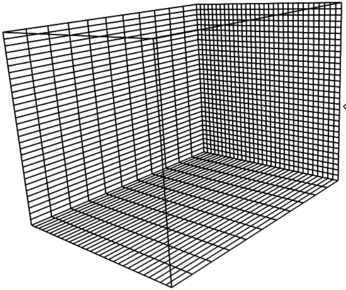
Topology and grids

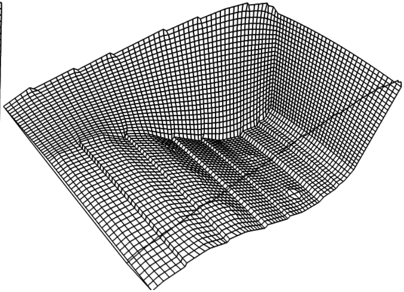
- Layman definition: how are points of the space connected up to form a line/surface/volume
- Concept is of direct use in “grids”
- *Grid* in scientific computing = points + topology
- Space is broken into small “pieces” called
  - Cells
  - Elements
- Data can be associated with the points or cells

Structured versus unstructured grids

Structured grids

- Implicit topology associated with points
- Easiest example: a rectangular mesh
- Non-rectangular mesh certainly possible

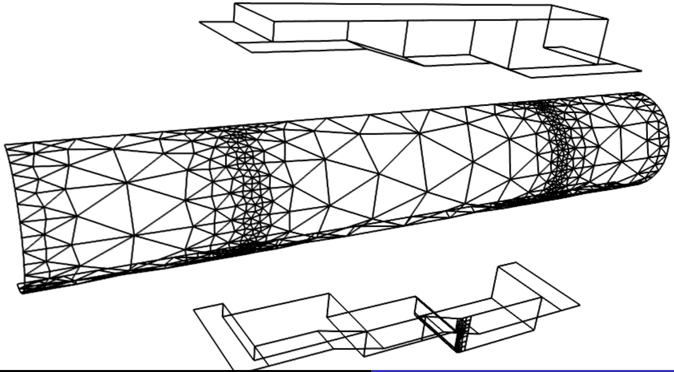




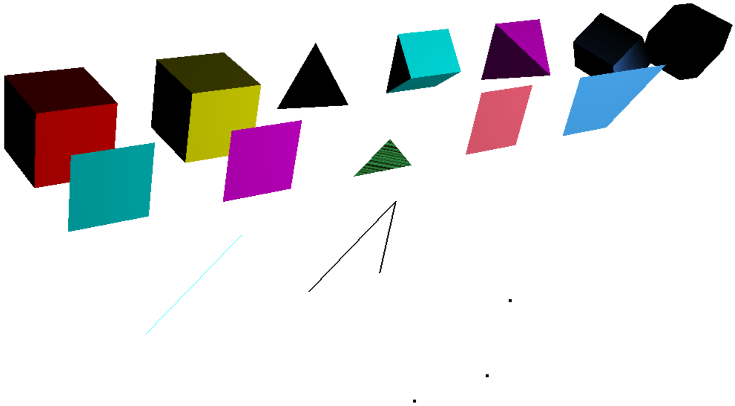
Structured versus unstructured grids

Unstructured grids

- Explicit topology specification
- Specified via connectivity lists
- Different number of neighbors, different types of cells



Different types of cells



Scalar, vector and tensor fields

- Associate a scalar/vector/tensor with every point of the space
- Scalar field: $f(\mathcal{R}^n) \longrightarrow \mathcal{R}$
- Vector field: $f(\mathcal{R}^n) \longrightarrow \mathcal{R}^m$
- Some examples:
    - Temperature distribution on a rod
    - Pressure distribution in room
    - Velocity field in room
    - Vorticity field in room
    - Stress tensor field on a surface
- Two aspects of field data, representation and visualization

Outline

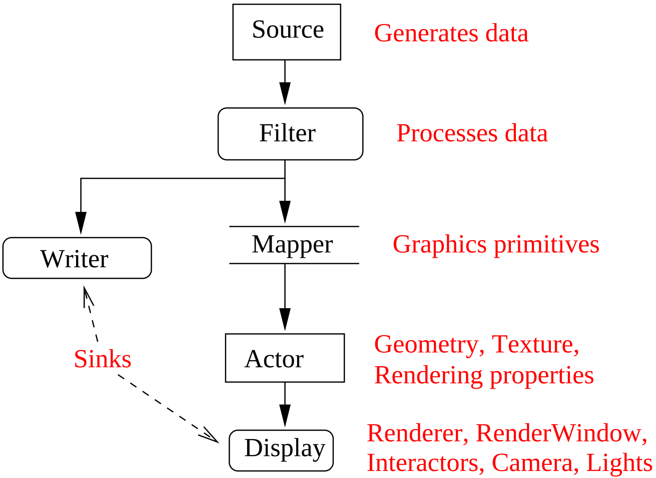
Introduction

- Open source, BSD style license
- High level library
- 3D graphics, imaging and visualization
- Core implemented in C++ for speed
- Uses OpenGL for rendering
- Wrappers for Python, Tcl and Java
- Cross platform: *nix, Windows, and Mac OSX
- Around 40 developers worldwide
- Very powerful with lots of features/functionality

VTK: an overview

- Pipeline architecture
- Huge with over 900 classes
- Not trivial to learn
- Need to get the VTK book
- Reasonable learning curve

VTK / TVTK pipeline



Example VTK script

```python
import vtk
# Source object.
cone = vtk.vtkConeSource()
cone.SetHeight(3.0)
cone.SetRadius(1.0)
cone.SetResolution(10)
# The mapper.
coneMapper = vtk.vtkPolyDataMapper()
coneMapper.SetInput(cone.GetOutput())
# The actor.
coneActor = vtk.vtkActor()
coneActor.SetMapper(coneMapper)
# Set it to render in wireframe
coneActor.GetProperty().SetRepresentationToWireframe()
```

See TVTK example

Outline

Legacy VTK Data files

- Detailed documentation on this is available here: http://www.vtk.org/pdf/file-formats.pdf.
- VTK data files support the following Datasets.
  1. Structured points.
  2. Rectilinear grid.
  3. Structured grid.
  4. Unstructured grid.
  5. Polygonal data.
- Binary and ASCII files are supported.

General structure

```
# vtk DataFile Version 2.0
A long string describing the file (256 chars)
ASCII | BINARY
DATASET [type]
...

POINT_DATA n
...

CELL_DATA n
...
```

- Point and cell data can be supplied together.
- `n` is the number of points or cells.

Structured Points

```
# vtk DataFile Version 2.0
Structured points example.
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS nx ny nz
ORIGIN x0 y0 z0
SPACING sx sy sz
```

- Important: There is an implicit ordering of points and cells. The X co-ordinate increases first, Y next and Z last.
- $nx \geq 1, ny \geq 1, nz \geq 1$

Rectilinear Grid

```
# vtk DataFile Version 2.0
Rectilinear grid example.
ASCII
DATASET RECTILINEAR_GRID
DIMENSIONS nx ny nz
X_COORDINATES nx [dataType]
x0 x1 ... x(nx-1)
Y_COORDINATES ny [dataType]
y0 y1 ... y(ny-1)
Z_COORDINATES nz [dataType]
z0 z1 ... z(nz-1)
```

- Important: Implicit ordering as in structured points. The *X* co-ordinate increases first, *Y* next and *Z* last.

Structured Grid

```
# vtk DataFile Version 2.0
Structured grid example.
ASCII
DATASET STRUCTURED_GRID
DIMENSIONS nx ny nz
POINTS N [dataType]
x0 y0 z0
x1 y0 z0
x0 y1 z0
x1 y1 z0
x0 y0 z1
...
```

- Important: The *X* co-ordinate increases first, *Y* next and *Z* last.
- N = nx*ny*nz

Polygonal data

```
[ HEADER ]
DATASET POLYDATA
POINTS n dataType
x0 y0 z0
x1 y1 z1
...
x(n-1) y(n-1) z(n-1)

POLYGONS numPolygons size
numPoints0 i0 j0 k0 ...
numPoints1 i1 j1 k1 ...
...
```

- size = total number of connectivity indices.

Unstructured grids

```
[ HEADER ]
DATASET UNSTRUCTURED_GRID
POINTS n dataType
x0 y0 z0
...
x(n-1) y(n-1) z(n-1)

CELLS n size
numPoints0 i j k l ...
numPoints1 i j k l ...
...

CELL_TYPES n
type0
type1
...
```

- size = total number of connectivity indices.

Dataset attributes

- Associated with each point/cell one may specify an attribute.
- VTK data files support scalar, vector and tensor attributes.
- Cell and point data attributes.
- Multiple attributes per same file.

Scalar attributes

```
SCALARS dataName dataType numComp
LOOKUP_TABLE tableName
s0
s1
...
```

- `dataName`: any string with no whitespace (case sensitive!).
- `dataType`: usually `float` or `double`.
- `numComp`: optional and can be left as empty.
- `tableName`: use the value `default`.

Vector attributes

```
VECTORS dataName dataType
v0x v0y v0z
v1x v1y v1z
...
```

- `dataName`: any string with no whitespace (case sensitive!).
- `dataType`: usually `float` or `double`.

Simple example

```
# vtk DataFile Version 2.0
Structured points example.
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 2 2 1
ORIGIN 0.0 0.0 0.0
SPACING 1.0 1.0 1.0

POINT_DATA 4
SCALARS Temperature float
LOOKUP_TABLE default
100 200
300 400

VECTORS velocity float
0.0 0.0 0.0
1.0 0.0 0.0
0.0 1.0 0.0
1.0 1.0 0.0
```

Outline

Issues with VTK

- API is not Pythonic for complex scripts
- Native array interface
- Using NumPy arrays with VTK: non-trivial and inelegant
- Native iterator interface
- Can't be pickled
- GUI editors need to be "hand-made" ($>$ 800 classes!)

Introduction to TVTK

- “Traitified” and Pythonic wrapper atop VTK
- Elementary pickle support
- `Get/SetAttribute()` replaced with an `attribute` trait
- Handles numpy arrays/Python lists transparently
- Utility modules: pipeline browser, `ivtk`, `mlab`
- Envisage plugins for `tvtk` scene and pipeline browser
- BSD license
- Linux, Win32 and Mac OS X
- Unit tested

Example TVTK script

```
from enthought.tvtk.api import tvtk
cone = tvtk.ConeSource(height=3.0, radius=1.0,
                       resolution=10)
coneMapper = tvtk.PolyDataMapper(input=cone.output)
p = tvtk.Property(representation='w')
coneActor = tvtk.Actor(mapper=coneMapper, property=p)
```

See VTK example

The differences

VTK	TVTK
`import vtk`	`from enthought.tvtk.api import tvtk`
`vtk.vtkConeSource`	`tvtk.ConeSource`
no constructor args	traits set on creation
`cone.GetHeight()`	cone.height
`cone.SetRepresentation()`	`cone.representation='w'`

- `vtk3DWidget` → `ThreeDWidget`
- Method names: consistent with ETS (`lower_case_with_underscores`)
- VTK class properties (Set/Get pairs or Getters): traits

TVTK and traits

- Attributes may be set on object creation
- Multiple properties may be set via `set`
- Handy access to properties
- Usual trait features (validation/notification)
- Visualization via automatic GUI
- `tvtk` objects have strict traits
- `pickle` and `cPickle` can be used

Collections behave like sequences

```
>>> ac = tvtk.ActorCollection()
>>> print len(ac)
0
>>> ac.append(tvtk.Actor())
>>> print len(ac)
1
>>> for i in ac:
...     print i
...
# [Snip output]
>>> ac[-1] = tvtk.Actor()
>>> del ac[0]
>>> print len(ac)
0
```

Array handling

- All `DataArray` subclasses behave like Pythonic arrays:
  - support iteration over sequences, append, extend etc.
- Can set array using `vtk_array.from_array(numpy_array)`
  - Works with Python lists or a NumPy array
- Can get the array into a NumPy array via `numpy_arr = vtk_array.to_array()`
- `Points` and `IdList`: support these
- `CellArray` does not provide a sequence like protocol
- All methods and properties that accept a `DataArray, Points` etc. transparently accepts a NumPy array or a Python list!
- Most often these use views of the NumPy array!

Array example

Any method accepting `DataArray`, `Points`, `IdList` or `CellArray` instances can be passed a numpy array or a Python list!

```
>>> from enthought.tvtk.api import tvtk
>>> from numpy import array
>>> points = array([[0,0,0], [1,0,0], [0,1,0], [0,0,1]], 'f')
>>> triangles = array([[0,1,3], [0,3,2], [1,2,3], [0,2,1]])
>>> mesh = tvtk.PolyData()
>>> mesh.points = points
>>> mesh.polys = triangles
>>> temperature = array([10, 20 ,20, 30], 'f')
>>> mesh.point_data.scalars = temperature
>>> import operator # Array's are Pythonic.
>>> reduce(operator.add, mesh.point_data.scalars, 0.0)
80.0
>>> pts = tvtk.Points() # Demo of from_array/to_array
>>> pts.from_array(points)
>>> print pts.to_array()
```

Array example: contrast with VTK

VTK and arrays

```
>>> mesh = vtk.vtkPolyData()
>>> # Assume that the points and triangles are set.
... sc = vtk.vtkFloatArray()
>>> sc.SetNumberOfTuples(4)
>>> sc.SetNumberOfComponents(1)
>>> for i, temp in enumerate(temperatures):
...     sc.SetValue(i, temp)
...
>>> mesh.GetPointData().SetScalars(sc)
```

Equivalent to (but more inefficient):

TVTK and arrays

```
>>> mesh.point_data.scalars = temperature
```

TVTK: easier and more efficient!

Some issues with array handling

- Details of array handling documented in `tvtk/docs/README.txt`
- Views and copies: a copy is made of the array in the following cases:
  - Python list is passed
  - Non-contiguous numpy array
  - Method requiring conversion to a `vtkBitArray` (rare)
  - Rarely: VTK data array expected and passed numpy array types are different (rare)
  - `CellArray` always makes a copy on assignment
  - Use `CellArray.set_cells()` method to avoid copies
  - `IdList` always makes a copy
  - Warning: Resizing the TVTK array reallocates memory

Summary of array issues

- `DataArray, Points`: don't make copies usually
- Can safely delete references to a numpy array
- Cannot resize numpy array
- `CellArray` makes a copy unless `set_cells` is used
- Warning: Resizing the TVTK array reallocates memory: leads to a copy
    - Note: not a memory leak

Outline

Overview

- An overview of creating datasets with TVTK and NumPy
- We consider simple examples
- Very handy when working with NumPy
- No need to create VTK data files
- `PolyData`, `StructuredPoints`, `StructuredGrid`, `UnstructuredGrid`

PolyData

```
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/verts use: mesh.lines = lines; mesh.verts = vertices
# Now create some point data.
temperature = array([10, 20 ,20, 30], 'f')
mesh.point_data.scalars = temperature
mesh.point_data.scalars.name = 'temperature'
# Some vectors.
velocity = array([[0,0,0], [1,0,0], [0,1,0], [0,0,1]], 'f')
mesh.point_data.vectors = velocity
mesh.point_data.vectors.name = 'velocity'
# Thats it!
```

Structured Points: 2D

```
# The scalar values.
from numpy import arange, sqrt
from scipy import special
x = (arange(50.0)-25)/2.0
y = (arange(50.0)-25)/2.0
r = sqrt(x[:,None]**2+y**2)
z = 5.0*special.j0(r)  # Bessel function of order 0
# ---------------------------------------------------
# Can't specify explicit points, the points are implicit.
# The volume is specified using an origin, spacing and dimensions
spoints = tvtk.StructuredPoints(origin=(-12.5,-12.5,0),
                                spacing=(0.5,0.5,1),
                                dimensions=(50,50,1))
# Transpose the array data due to VTK's implicit ordering. VTK
# assumes an implicit ordering of the points: X co-ordinate
# increases first, Y next and Z last.  We flatten it so the
# number of components is 1.
spoints.point_data.scalars = z.T.flatten()
spoints.point_data.scalars.name = 'scalar'
```

Structured Points: 3D

```
from numpy import array, ogrid, sin, ravel
dims = array((128, 128, 128))
vol = array((-5., 5, -5, 5, -5, 5))
origin = vol[::2]
spacing = (vol[1::2] - origin)/(dims -1)
xmin, xmax, ymin, ymax, zmin, zmax = vol
x, y, z = ogrid[xmin:xmax:dims[0]*1j,ymin:ymax:dims[1]*1j,
                zmin:zmax:dims[2]*1j]
x, y, z = [t.astype('f') for t in (x, y, z)]
scalars = sin(x*y*z)/(x*y*z)
# ----------------------------------------------
spoints = tvtk.StructuredPoints(origin=origin, spacing=spacing,
                                dimensions=dims)
# The copy makes the data contiguous and the transpose
# makes it suitable for display via tvtk.
s = scalars.transpose().copy()
spoints.point_data.scalars = ravel(s)
spoints.point_data.scalars.name = 'scalars'
```

Structured Grid

```
r = numpy.linspace(1, 10, 25)
theta = numpy.linspace(0, 2*numpy.pi, 51)
z = numpy.linspace(0, 5, 25)
# Crreate an annulus.
x_plane = (cos(theta)*r[:,None]).ravel()
y_plane = (sin(theta)*r[:,None]).ravel()
pts = empty([len(x_plane)*len(height),3])
for i, z_val in enumerate(z):
    start = i*len(x_plane)
    plane_points = pts[start:start+len(x_plane)]
    plane_points[:,0] = x_plane
    plane_points[:,1] = y_plane
    plane_points[:,2] = z_val
sgrid = tvtk.StructuredGrid(dimensions=(51, 25, 25))
sgrid.points = pts
s = numpy.sqrt(pts[:,0]**2 + pts[:,1]**2 + pts[:,2]**2)
sgrid.point_data.scalars = numpy.ravel(s.copy())
sgrid.point_data.scalars.name = 'scalars'
```

Unstructured Grid

```
from numpy import array
points = array([[0,0,0], [1,0,0], [0,1,0], [0,0,1]], 'f')
tets = array([[0, 1, 2, 3]])
tet_type = tvtk.Tetra().cell_type # VTK_TETRA == 10
#-----------------------------------------------------------
ug = tvtk.UnstructuredGrid()
ug.points = points
# This sets up the cells.
ug.set_cells(tet_type, tets)
# Attribute data.
temperature = array([10, 20 ,20, 30], 'f')
ug.point_data.scalars = temperature
ug.point_data.scalars.name = 'temperature'
# Some vectors.
velocity = array([[0,0,0], [1,0,0], [0,1,0], [0,0,1]], 'f')
ug.point_data.vectors = velocity
ug.point_data.vectors.name = 'velocity'
```

Scene widget, pipeline browser and `ivtk`

- `enthought.pyface.tvtk`: scene widget
  - Provides a Pyface `tvtk` render window interactor
  - Supports VTK widgets
  - Picking, lighting
- `enthought.tvtk.pipeline.browser`
  - Tree-view of the `tvtk` pipeline
- `enthought.tvtk.tools.ivtk`
  - Like MayaVi-1's `ivtk` module
  - Convenient, easy to use, viewer for `tvtk`

`mlab` interface

- `enthought.tvtk.tools.mlab`
- Provides Matlab like 3d visualization conveniences
- API mirrors that of Octaviz: `http://octaviz.sf.net`
- Place different Glyphs at points
- 3D lines, meshes and surfaces
- Titles, outline

Envisage plugins

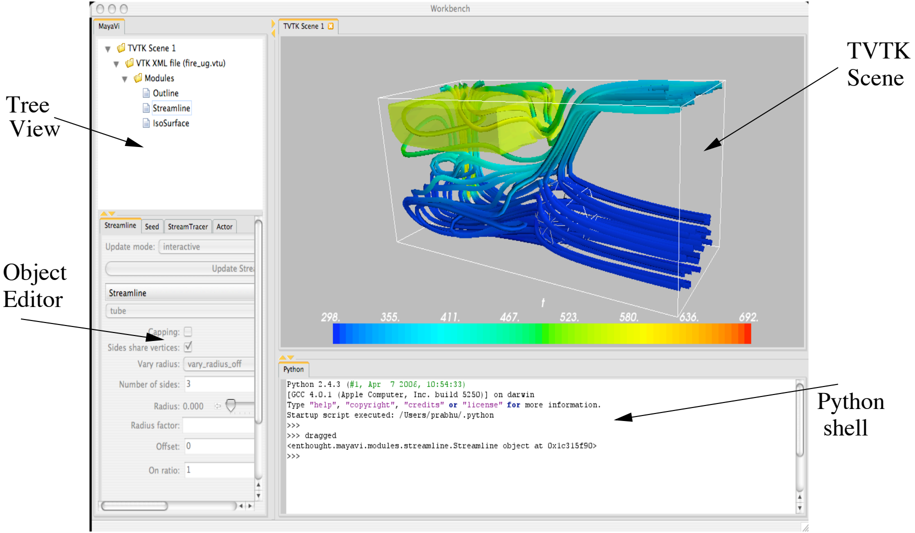
- Envisage: an extensible plugin based application framework
- `enthought.tvtk.plugins.scene`
    - Embed a TVTK render window
    - Features all goodies in `enthought.pyface.tvtk`
- `enthought.tvtk.plugins.browser`

Outline

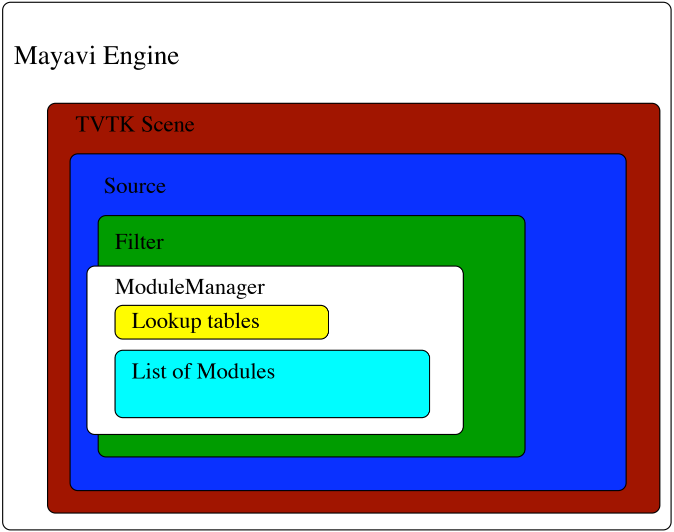
Features

- MayaVi-2: built atop Traits, TVTK and Envisage
- Focus on building the model right
- Uses traits heavily
- MayaVi-2 is an Envisage plugin
- Workbench plugin for GUI
- `tvtk` scene plugin for TVTK based rendering
- View/Controller: “free” with traits and Envisage
- MVC
- Uses a simple, persistence engine

Example view of MayaVi-2



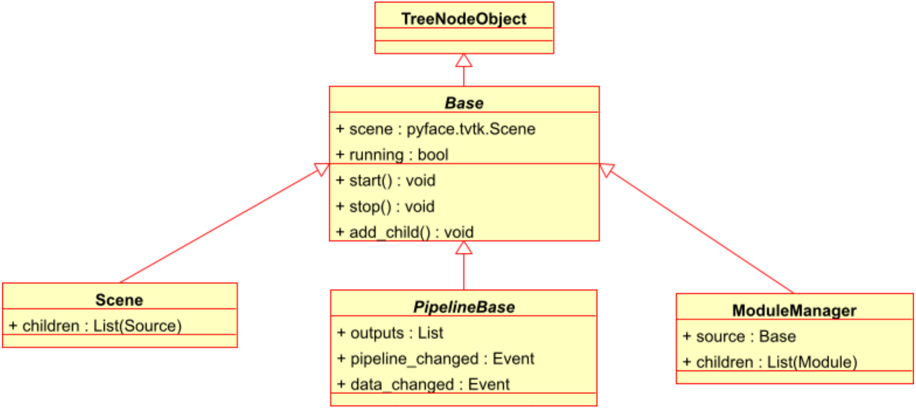
The big picture



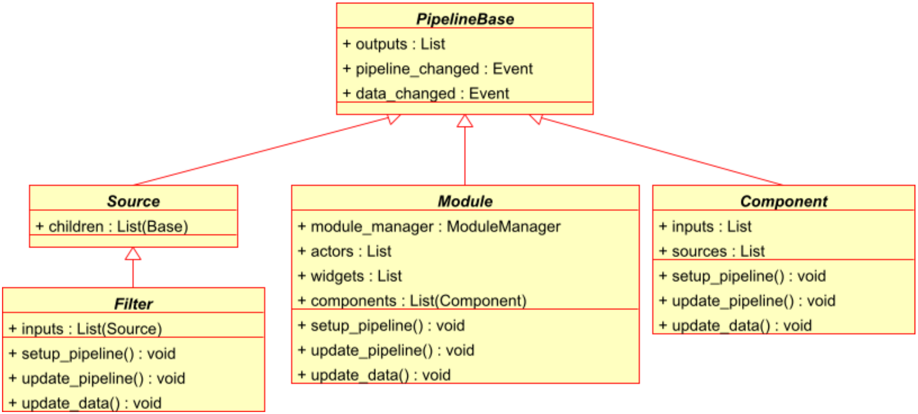
Outline

1. Introduction to visualization
   - Introduction
   - Quick graphics: `visual` and `mlab`
   - Graphics primer
   - Data and its representation
2. Visualization libraries
   - VTK
   - VTK data
   - TVTK
   - Creating Datasets from NumPy
3. MayaVi2
   - Introduction
   - Internals
   - Examples

Class hierarchy



Class hierarchy



Containership relationship

Engine
+ scenes : List(Scene)
+ start() : void + stop() : void + add_source(src : Source) : void + add_filter(fil : Filter) : void + add_module(mod : Module) : void

- `Engine` contains: list of `Scene`
- `Scene` contains: list of `Source`
- `Source` contains: list of `Filter` and/or `ModuleManager`
- `ModuleManager` contains: list of `Module`
- `Module` contains: list of `Component`

Outline

Interactively scripting MayaVi-2

- Drag and drop
- The `mayavi` instance

```
>>> mayavi.new_scene() # Create a new scene
>>> mayavi.save_visualization('foo.mv2')
```

- `mayavi.engine`:

```
>>> e = mayavi.engine # Get the MayaVi engine.
>>> e.scenes[0] # first scene in mayavi.
>>> e.scenes[0].children[0]
>>> # first scene's first source (vtkfile)
```

Scripting . . .

- `mayavi`: instance of `enthought.mayavi.script.Script`
- Traits: `application`, `engine`
- Methods (act on current object/scene):
    - `new_scene()`
    - `add_source(source)`
    - `add_filter(filter)`
    - `add_module(m2_module)`
    - `save/load_visualization(fname)`

Scripting example

```
from enthought.mayavi.sources.vtk_file_reader import VTKFileReade
from enthought.mayavi.modules.outline import Outline
from enthought.mayavi.modules.grid_plane import GridPlane

from enthought.tvtk.api import tvtk
mayavi.new_scene()
src = VTKFileReader()
src.initialize('heart.vtk')
mayavi.add_source(src)
mayavi.add_module(Outline())
g = GridPlane()
g.grid_plane.axis = 'x'
mayavi.add_module(g)
g = GridPlane()
g.grid_plane.axis = 'y'
mayavi.add_module(g)
g = GridPlane()
g.grid_plane.axis = 'z'
mayavi.add_module(g)
```

Stand alone scripts

- Two approaches to doing this:
  - Approach 1:
    - Recommended way: simple interactive script save to a `script.py` file
    - Run it like `mayavi2 -x script.py`
    - Advantages: easy to write, can edit from mayavi and rerun
    - Disadvantages: not a stand-alone Python script
  - Approach 2:
    - Subclass `enthought.mayavi.app.Mayavi`
    - Override the `run()` method
    - `self.script` is a `Script` instance

ipython -wthread

```python
from enthought.mayavi.app import Mayavi
m = Mayavi()
m.main()
m.script.new_scene()
# `m.script` is the mayavi.script.Script instance
engine = m.script.engine
# Script as usual ...
```