# 3D visualization with TVTK and MayaVi2

Prabhu Ramachandran

Department of Aerospace Engineering
IIT Bombay

14th–18th August, 2006

## Outline

## VTK

- TVTK and MayaVi2 use VTK
- VTK:
  - 3D graphics, imaging and visualization
  - C++ code wrapped to Python (Tcl, Java . . . )
  - Pipeline architecture
  - Huge: 900 classes!
  - Cross-platform, BSD license
- VTK-Python not "Pythonic" enough
  - Native array interface
  - Using numpy arrays: non-trivial, inelegant, inefficient
  - Native iterator interface
  - Can't be pickled
  - GUI editors need to be "hand-made" ($>$ 800 classes!)
- TVTK: "Traitified", Pythonic wrapper for VTK-Python

## MayaVi-1

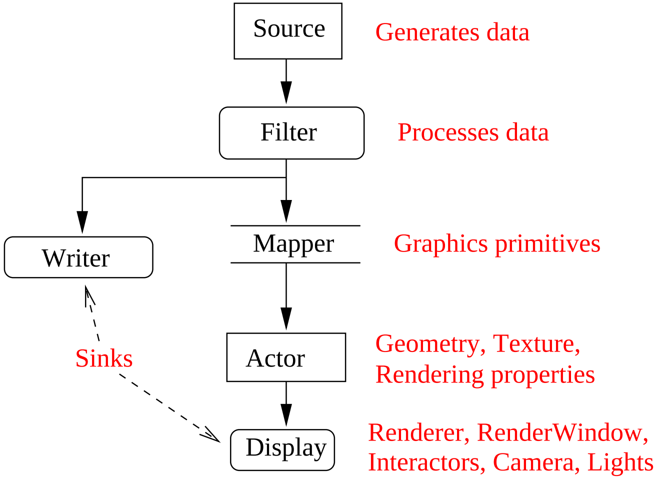
- MayaVi-1:
  - 3D/2D visualization (scalars, vectors, rank 2 tensors)
  - 100% Python, lightweight, pretty fast
  - Interactively (and otherwise) scriptable (but only just)
  - Extensible via user defined code
  - Clunky (function-is-everything) Tkinter UI
  - Cross-platform and BSD license
  - Released in 2001, amazingly it is still used!?
- Problems:
  - No MVC
  - Ugly(?) UI
  - File format: hack!
  - Not embeddable
  - Not easily scriptable
- MayaVi-2: MayaVi-1 reloaded: re-designed, re-implemented

## Outline

## Features

- "Traitified" and Pythonic wrapper atop VTK
- Elementary pickle support
- 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

## VTK / TVTK pipeline



## Example VTK script

```
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()
```

## 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)
```

## 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 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

```
>>> 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):

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

TVTK: easier and more efficient!

## Outline

## 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`

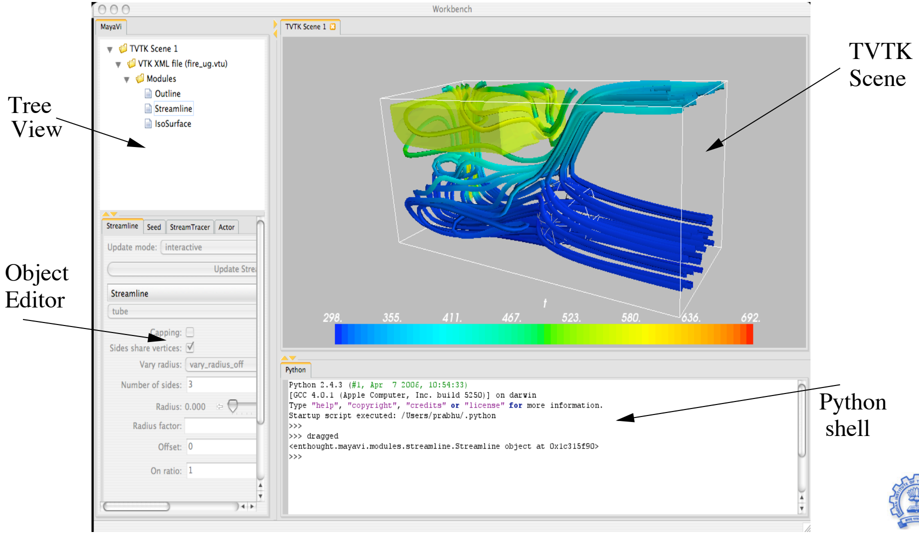
## Outline

## 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

## Outline

## Example view of MayaVi-2



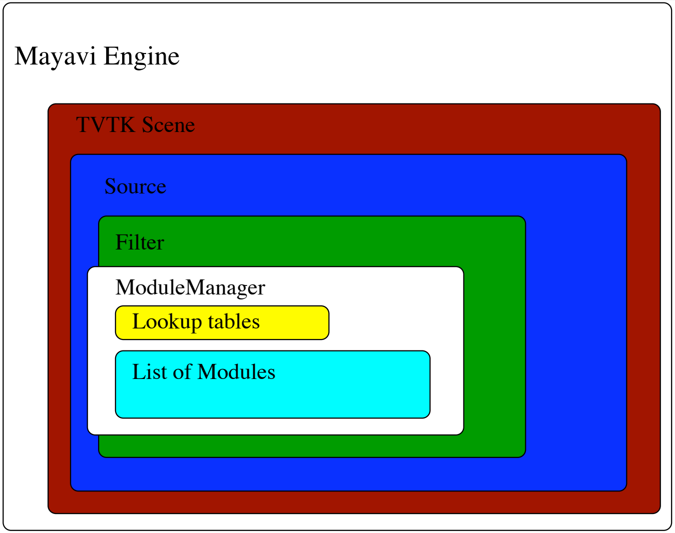
TVTK Scene 1
VTK XML file (fire_ug.vtu)
Modules
Outline
Streamline
IsoSurface

Streamline | Seed | StreamTracer | Actor

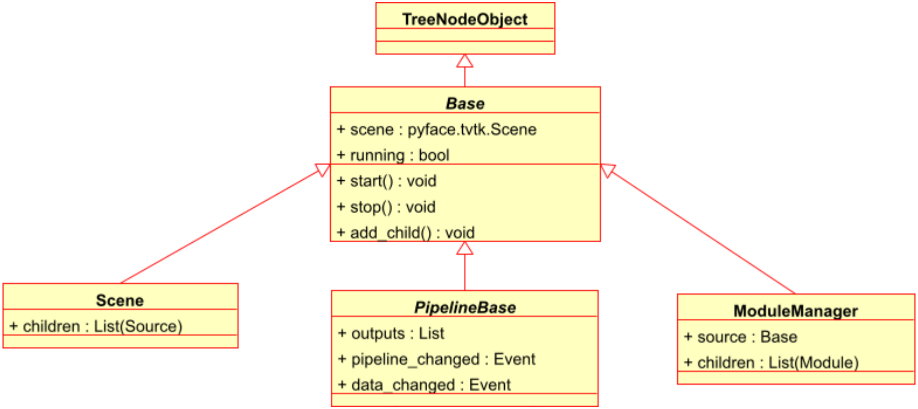
Update mode: interactive
Streamline
tube
Capping: ☐
Sides share vertices: ✓
Vary radius: vary_radius_off
Number of sides: 3
Radius: 0.000
Radius factor:
Offset: 0
On ratio: 1

```
Python 2.4.3 (#1, Apr  7 2006, 10:54:33)
[GCC 4.0.1 (Apple Computer, Inc. build 5250)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Startup script executed: /Users/prabhu/.python
>>>
>>> dragged
<enthought.mayavi.modules.streamline.Streamline object at 0x1c315f90>
>>>
```

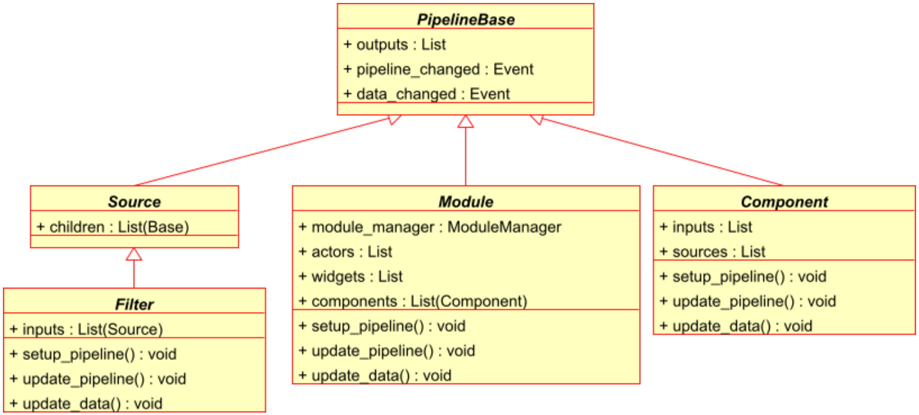
## The big picture



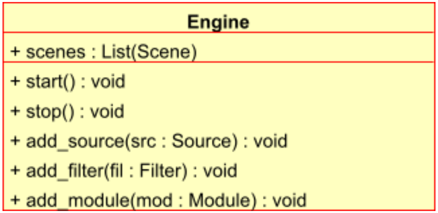
## Class hierarchy



## 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`

## 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)`

## Stand alone scripts

- Subclass `enthought.mayavi.app.Mayavi`
- Override the `run()` method
- `self.script` is a `Script` instance

## ipython -wthread

```
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
```

## Status

- TVTK: stable, tested, documented
- MayaVi2: core is stable, but feature incomplete, and not fully documented, definitely usable