TVTK and MayaVi2: creating datasets

Prabhu Ramachandran

Department of Aerospace Engineering
IIT Bombay

16, August 2007
SciPy07 Conference

Outline

Outline

Introduction

- Cross-platform 2D/3D visualization for scientists and engineers
- Most scientists not interested in details of visualization
- Almost all 2D plotting: matplotlib and Chaco
- More complex 2D/3D visualization
  - Unfortunately, not as easy as 2D (yet)

Introduction

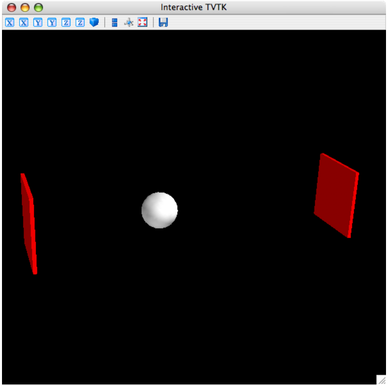
- VTK: Powerful 3D visualization
- MayaVi/TVTK: tools for easy visualization
- TVTK: VTK + Traits + Numpy support == Pythonic VTK

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
```



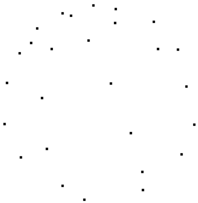
Outline

1. Introduction
2. Creating TVTK Datasets from NumPy
   - Creating the datasets from Python

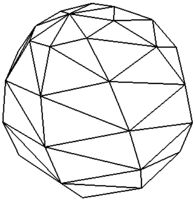
Datasets: Why the fuss?

- Visualizing 3D data requires a little more information than 2D
- Need to specify a topology (i.e. how are the points connected)
- In 2D things are a lot easier to figure out

An example of the difficulty



Points

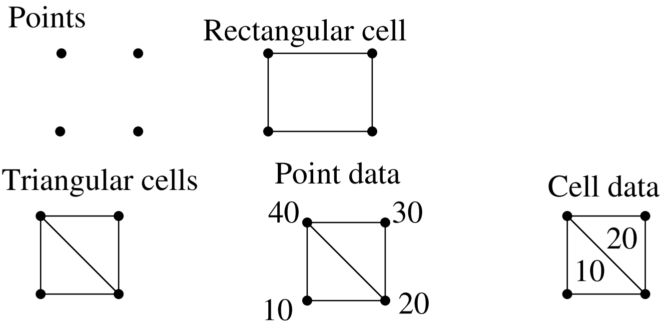


Wireframe

Surface

The general idea

- Specify the points of the space
- Specify the connectivity between the points (topology)
- The connectivity lets you build "cells" that break the space into pieces
- Specify "attribute" data at the points or cells



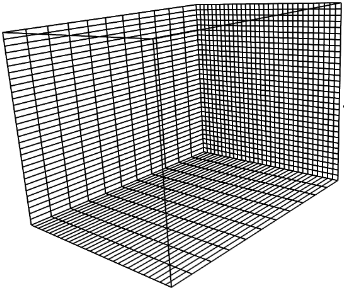
Types of datasets

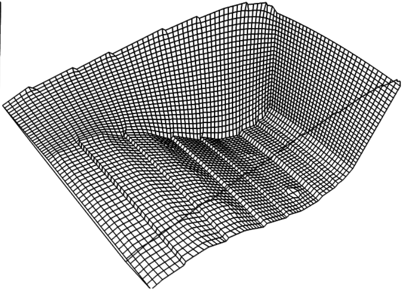
- Implicit topology (structured):
  - Image data (structured points)
  - Rectilinear grids
  - Structured grids
- Explicit topology (unstructured):
  - Polygonal data (surfaces)
  - Unstructured grids

Implicit versus explicit topology

Structured grids

- 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

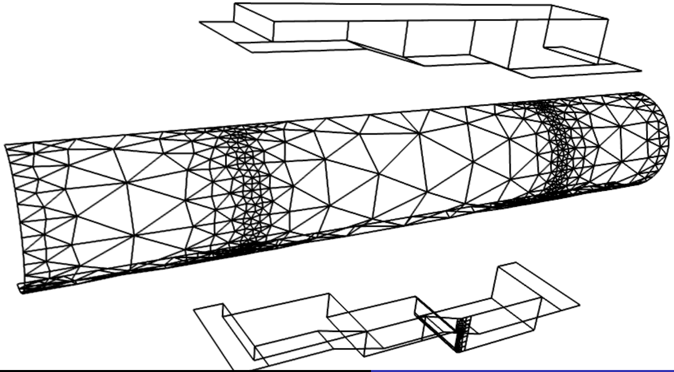




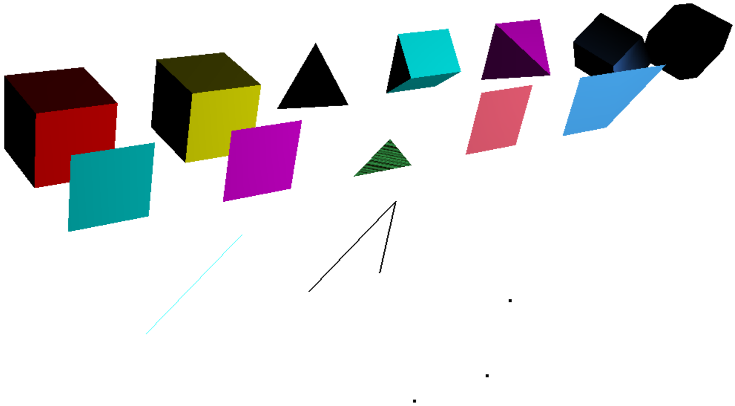
Implicit versus explicit topology

Unstructured grids

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



Different types of cells



Dataset attributes

- Associated with each point/cell one may specify an attribute
  - Scalars
  - Vectors
  - Tensors
- Cell and point data attributes
- Multiple attributes per dataset

Structured Points: 2D

```
# The scalar values.
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.
# 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: 2D

```
# The scalar values.
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.
# 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

```
x, y, z = ogrid[-5:5:128j,-5:5:128j,
                -5:5:128j]
x, y, z = [t.astype('f') for t in (x, y, z)]
scalars = sin(x*y*z)/(x*y*z)


# -----------------------------------------------------
spoints = tvtk.StructuredPoints(origin=(-5,-5,5),
                               spacing=(10./127,10./127,10./127),
                               dimensions=(128,128,128))
# 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)
# Create 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'
```

Structured Grid

```
r = numpy.linspace(1, 10, 25)
theta = numpy.linspace(0, 2*numpy.pi, 51)
z = numpy.linspace(0, 5, 25)
# Create 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'
```

PolyData

```python
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'
```

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'
```

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(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'
```

More information

- Examples are all in the mayavi2 examples
- More elaborate information available at the MayaVi2 wiki: `https://svn.enthought.com/enthought/wiki/MayaVi`

The future: MayaVi a reusable library

- MayaVi was (till yesterday) not usable outside Envisage app
- No longer true
- Makes mayavi a truly reusable library for easy visualization