Programming for Image Processing/Analysis and Visualization using The Visualization Toolkit

Week 4: Surface and Image Manipulation

http://noodle.med.yale.edu/~papad/seminar/
Man Pages etc: http://noodle.med.yale.edu/tcl
and http://noodle.med.yale.edu/vtk/

Xenios Papademetris
papad@noodle.med.yale.edu
BML 325, 5-7294

Revised Schedule for Part 1

1. Introduce VTK (9/3)
2. Introduce Tcl/Tk (9/10)
3. Simple Visualization Tasks (9/17)
4. Surface and Image Manipulation (9/23)
5. Image Display (Volume Rendering) (10/1)
6. Local Extensions (10/8)
7. Tk and GUIs (10/15)
8. [Incr] Tcl and other extensions (10/22)

Revision -- VTK Pipeline (I)

A Closer Look at vtkPolyData

- vtkPolyData is a complex class which has many members. The key ones are:
  - Points of type vtkPoints -- represents the geometry of the surface (i.e. the points)
  - Polys of type vtkCellArray -- represents part of the topology of the surface (i.e. the polygons)
  - PointData of type vtkPointData -- represents data associated with the points (e.g. normals, colors etc)
  - CellData of type vtkCellData -- represents data associated with the cells (e.g. again normals, colors etc)
  - Lots of other members ......

Representing arrays with vtkDataArray

- vtkDataArray is an abstract superclass for classes representing arrays of vectors called tuples (or numbers treated as vectors of length 1). Each tuple consists of a set of numbers or components
- Derived Classes include vtkUnsignedCharArray, vtkShortArray, vtkFloatArray, vtkDoubleArray etc.
- Can function either as a dynamic array (lower performance) or a fixed length array
- All data in VTK is stored ultimately in one of the many derived classes of vtkDataArray.
  e.g. in the case of vtkImageData the intensities are stored in a vtkDataArray having dimensions equal to the number of voxels and vector length typically equal to 1 (3 for color images, Number of Frames for multiframe data such as MRI cardiac etc.)
**vtkDataArray Hierarchy**

**A concrete example vtkFloatArray**

**Mode 1 – Fixed Length Array:**

To create in TCL type:

```tcl
tkFloatArray arr
arr SetNumberOfComponents 1
arr SetNumberOfTuples 20
arr SetComponent 10 0 10.0
arr SetTuple1 11 9.0
set b [arr GetComponent 10 0]
```

This creates an array of 20 (Number of Tuples) vectors each having size 1 (Number of Components).

We access elements in this array by using the `SetComponent` and `GetComponent` methods. All indices start at 0.

**Mode 2 – Dynamic Array**

To create in TCL type:

```tcl
tkFloatArray arr
arr SetNumberOfComponents 1
arr InsertNextTuple 5
arr InsertNextTuple 10
set b [arr GetComponent 1 0]
```

This creates a dynamic array of vectors each having size 1 (Number of Components). The `InsertNextTuple` command allocates memory dynamically and places the value there.

Note also (InsertNextTuple2, InsertNextTuple3,4,9 for multi-component arrays)

**A final note on vtkDataArray**

The following constants are often defined:

- `VTK_VOID` as 0
- `VTK_BIT` as 1
- `VTK_CHAR` as 2
- `VTK_UNSIGNED_CHAR` as 3
- `VTK_SHORT` as 4
- `VTK_UNSIGNED_SHORT` as 5
- `VTK_INT` as 6
- `VTK_UNSIGNED_INT` as 7
- `VTK_LONG` as 8
- `VTK_UNSIGNED_LONG` as 9
- `VTK_FLOAT` as 10
- `VTK_DOUBLE` as 11

Creating a Surface Manually – Step 1

```tcl
# Create Points for a cube
vtkPoints pt
pt SetNumberOfPoints 8
pt SetPoint 0 0 0 0; pt SetPoint 1 1 0 0;
pt SetPoint 1 0 1 0; pt SetPoint 0 1 0 0;
pt SetPoint 1 1 1 1; pt SetPoint 0 1 1 1;
pt SetPoint 1 0 1 1; pt SetPoint 0 0 1 1;

# Create Polygons
vtkCellArray cl

# Insert a Square
cl InsertNextCell 4
cl InsertCellPoint 0; cl InsertCellPoint 1;
cl InsertCellPoint 2; cl InsertCellPoint 3;

# Insert a Triangle
cl InsertNextCell 3
cl InsertCellPoint 2; cl InsertCellPoint 3;
cl InsertCellPoint 6
```

Creating a Surface Manually – Step 2

```tcl
# Create the Surface
vtkPolyData sur

# Set the points and cleanup
sur SetPoints pt; pt Delete

# Set the polygons and cleanup
sur SetPolys cl; cl Delete

# Create a Mapper and set its input
vtkPolyDataMapper map; map SetInput sur

# Create the actor and set it to display wireframes
vtkActor act; act SetMapper map

# Create the actor and set it to display wireframes
vtkActor act; act SetMapper map

# Create the renderer
vtkRenderer ren; ren AddActor act

# Set camera mode to Orthographic as opposed to Perspective
[ren GetActiveCamera] ParallelProjectionOn
```
Creating a Surface Manually – Step 3  
example4_1.tcl

```
# Rest is standard window/interactor etc.
# Render Window
vtkRenderWindow renWin
renWin.AddRenderer ren
renWin.SetSize 300 300

# Interactor
vtkRenderWindowInteractor iren
iren.SetRenderWindow renWin
iren.Initialize
iren.AddObserver SetExitMethod { exit }

# prevent the tk window from showing up then start the event loop
wm.withdraw .
vwaitforever
```

Finishing the Cube
example4_2.tcl

```
# Insert rest of cube - rest of the definitions
cl.InsertNextCell 4
cl.InsertCellPoint 0; cl.InsertCellPoint 1;
class InsertCellPoint 5; cl.InsertCellPoint 4;
cl.InsertNextCell 4
cl.InsertCellPoint 0; cl.InsertCellPoint 3;
class InsertCellPoint 7; cl.InsertCellPoint 4;
cl.InsertNextCell 4
cl.InsertCellPoint 1; cl.InsertCellPoint 2;
class InsertCellPoint 6; cl.InsertCellPoint 5;
cl.InsertNextCell 4
cl.InsertCellPoint 4; cl.InsertCellPoint 5;
class InsertCellPoint 6; cl.InsertCellPoint 7;
class InsertCellPoint 3; cl.InsertCellPoint 7; cl.InsertCellPoint 6
```

Adding Colors  
exampie4_3.tcl

```
cl.InsertNextCell 3
cl.InsertCellPoint 3; cl.InsertCellPoint 7; cl.InsertCellPoint 6

# Create color array, colors stored as RGB values
# 0-255
vtkUnsignedCharArray ar
ar.SetNumberOfComponents 3
ar.SetNumberOfTuples 8
ar.SetTuple3 0 255 0 ; ar.SetTuple3 1 255 0
ar.SetTuple3 2 255 0 ; ar.SetTuple3 3 255 0
ar.SetTuple3 4 255 0 ; ar.SetTuple3 5 255 0
ar.SetTuple3 6 255 255 255; ar.SetTuple3 7 255 255 255

vtkPolyData sur
sur.SetPoints pt; pt.Delete
sur.SetPolys cl; cl.Delete

# Set The Colors and Cleanup
[ sur.GetPointData ] SetScalars ar
ar.Delete
```

Viewing Surface with Solid Faces  
exampie4_4.tcl  example4_5.tcl

```
vtkUnsignedCharArray ar
ar.SetNumberOfComponents 3
ar.SetNumberOfTuples 8
ar.SetTuple3 0 255 255 0 ; ar.SetTuple3 1 255 0 0
ar.SetTuple3 2 255 128 0 ; ar.SetTuple3 3 255 0 0
ar.SetTuple3 4 255 0 0 ; ar.SetTuple3 5 255 255 255
ar.SetTuple3 6 0 255 255

vtkPolyData sur
sur.SetPoints pt; pt.Delete
sur.SetPolys cl; cl.Delete

# Add Colors to polygons not points (i.e. use Cell Data)
[ sur.GetCellData ] SetScalars ar
ar.Delete
```

Viewing Surface with Solid Faces  
example4_6.tcl

```
vtkFloatArray ar
ar.SetNumberOfComponents 1
ar.SetNumberOfTuples 7
ar.SetTuple1 0 0 ; ar.SetTuple1 1 1
ar.SetTuple1 2 2 ; ar.SetTuple1 3 3
ar.SetTuple1 4 4 ; ar.SetTuple1 5 5
ar.SetTuple1 6

# Colors are Red/Green/Blue/Opacity in range 0 to 1
vtkLookupTable lkup
lkup.SetNumberOfTableValues 7
lkup.SetTableRange 0 6
lkup.SetTableValue 0 1.0 0.0 1.0
lkup.SetTableValue 1 0.75 0.0 1.0
lkup.SetTableValue 2 0.50 0.0 1.0
lkup.SetTableValue 3 0.25 0.0 1.0
lkup.SetTableValue 4 0.0 0.0 1.0
lkup.SetTableValue 5 1.0 1.0 1.0
lkup.SetTableValue 6 0.0 1.0 1.0

vtkPolyData sur
sur.SetPoints pt; pt.Delete
sur.SetPolys cl; cl.Delete

# Add Colors to polygons not points (i.e. use Cell Data)
[ sur.GetCellData ] SetScalars ar
ar.Delete

vtkPolyDataMapper map
map.SetInput sur
map.SetScalarRange 0 6
map.SetScalarModeToUseCellData
Images are Simpler

```plaintext
tvtkImageData
- vtkImageData is the basic VTK class for storing images.
- It is defined by 4 key elements
  - Dimensions -- these define the size of the image
  - Origin -- the position in 3D space of point 0 0 0
  - Spacing -- the voxel dimensions
  - Scalar Type -- the type of the image (e.g. float, short etc)
- An 4x4x4 image has 4x4x4=64 points and 3x3x3=27 cubic cells (both are implicitly defined)
```

Manually Creating an Image

```plaintext
vtkImageData img
img SetDimensions 10 10 2
img SetOrigin 0 0 0
img SetSpacing 0.78 0.78 1.5
img SetScalarType VTK_SHORT
img SetNumberOfScalarComponents 1
img AllocateScalars
Intensity values can be accessed using the scalar array i.e. Point 0 is (0,0,0), point 1 is (1,0,0), point 10 is (0,1,0), point 100 is (0,0,1)
```

The Image as a Function

```plaintext
- VTK treats an image as a function which takes values equal to those specified at the points and interpolates in-between
- Standard Interpolation is (Tri-)Linear
- In some cases (vtkImageReslice) other interpolation schemes are available.
```

Image File Read/Write

```plaintext
- VTK supports by default a number of standard image file formats for read/write
  - Binary -- vtkImageReader
  - JPEG -- vtkJPEGReader, vtkJPEGWriter
  - PNG -- vtkPNGReader, vtkPNGWriter (ppm,png)
  - TIFF -- vtkTIFFReader, vtkTIFFWriter
  - BMP -- vtkBMPReader, vtkBMPWriter
- There are local extensions for reading Analyze, Signa LX/SPR, Prism (SPECT) etc
```

ImageToImage Filters

```plaintext
- There are a number of ImageToImage Filters which are analogous to the surface to surface filters we have met before.
- There are exist standard filters derived from vtkImageToImageFilter for among others
  - Smoothing -- vtkImageGaussianSmooth, vtkImageMedian3D
  - Computing Gradients/Laplacians -- vtkImageGradient, vtkImageLaplacian
  - Fourier Operations-- vtkImageFFT, vtkImageRFT
  - Resampling/Reslicing -- vtkImageResample, vtkImageReslice (vtkImageReslice on its own is reason enough to learn VTK, it implements enough operations that would take more than a year to code from scratch)
  - Flipping, Permutting -- vtkImageFlip, vtkImagePermute
```
Visualizing Images

• Images are displayed as textures mapped on polygons

• In OpenGL all textures must have dimensions that are powers of two

• Images are interpolated before display, hence some (small) loss of sharpness takes place (only visible in small images)
  – E.g. an 100x50 image will be resampled to 128x64 before display

• Similar issues for color display i.e. scalars vs lookup tables as in surfaces

• We will examine image display next week!