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

Week 7: Miscellaneous Topics

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 Manipulation and Display  (10/1)
6. Marcel: Graphical User Interfaces  (10/15)
7. Miscellaneous Topics  (11/5)
8. Local Extensions  (11/12)

Revision -- VTK Pipeline (I)

Data Representation

(vtkDataSet)

Images are Simpler

vtkImageData

• 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)
Creating an Image

# Create the Structure
vtkImageData img
# Set Dimensions
img SetDimensions 10 10 2
# 3D Coordinates of Bottom Left Voxel
img SetOrigin 0 0 0
# Voxel Size
img SetSpacing 0.78 0.78 1.5
# Image Type
img SetScalarTypeToShort
# Number of Components (or frames for timeseries)
img SetNumberOfScalarComponents 1
# Allocate the Memory
img AllocateScalars

Manipulating An Image with TCL

• Some operations require pointers and require the use of C++. We focus here on those operations that are accessible at the scripting level.
• The dimensions/spacing/origin of the image are accessible via
  set dim [ img GetDimensions ]
  set sp [ img GetSpacing ]
  set ori [ img GetOrigin ]
  dim, sp and ori a tcl lists of length 3. To access individual elements we use the lindex command, e.g.,
  set numslices [ lindex $dim 2 ]
  Set voxelwidth [ lindex $sp 0 ]
  Set voxelheight [ lindex $sp 1 ]
  Set slicethickness [ lindex $sp 2 ]
  (Strictly speaking this value should be the distance between slice centers and not the slice thickness)

Data is stored in an one-dimensional array. To find the index in that array of a voxel (i,j,k)
set index [ img ComputePointId $i $j $k ]

Accessing the Intensity Array

The intensities of the image are stored in the Scalars member of the PointData of the image. This is a DataArray of the appropriate type. To get a pointer to this array use
set data [ [ img GetPointData ] GetScalars ]

Then given the index of a given point (i,j,k) we can set/get the voxel Intensity value using
set v [ $data GetComponent $index 0 ]
Or more directly
set v [ img GetScalarComponentAsFloat $i $j $k 0 ]
Where index is the voxel id (obtained using ComputePointId) and 0 is the frame/component number. We can set the intensity to 123 using
$data SetComponent $index 123
Unfortunately there is no direct method for setting.

Manipulating the Intensity Array

To Fill a particular component with a particular value use
set data [ [ img GetPointData ] GetScalars ]
$data FillComponent 0 0

To get the range of values in a particular component use
set range [ $data GetRange ]
set minimum [ lindex $range 0 ]
set maximum [ lindex $range 1 ]

To get the maximum/minimum values allowed in the array use:
set max [ $data GetDataTypeMax ]
set max [ $data GetDataTypeMin ]

Transformations

• Five major transformation types:
  • vtkTransform represents a 4x4 linear transformation. It also allows for the concatenation of different linear operations such as translations, rotations, scales, multiplication with a 4x4 matrix etc.
  • vtkGridTransform represents a transformation as a vector field
  • vtkThinPlateSplineTransform represents the transformation as a mapping between two sets of corresponding points and a suitably defined interpolation kernel
  • vtkGeneralTransform allows for the concatenation of different types of transformations to form a single composite
  • vtkLandmarkTransform allows for the computation of least squares rigid/similarity or affine transformations given two sets of corresponding points. (These are represented as 4x4 matrices)
Concatenation

- Key item to remember is that VTK uses a pre-multiply scheme by default i.e. A concatenated with B means \([A][B]\) not the more conventional \([B][A]\).
- This behavior can be rectified in \(\text{vtkTransform}\) and \(\text{vtkGeneralTransform}\) using the \texttt{PostMultiply}\) method.
- All transformations support the \texttt{TransformPoint}\) method and the Identity Method.

Transforming a Surface

Given a surface poly we can translate it by \((10,5,0)\) using

\[
\text{vtkTransform } tr
\]
\[
tr \text{ Translate } 10 \ 5 \ 0
\]
\[
\text{vtkTransformPolyDataFilter } tf
\]
\[
tf \text{ SetInput } \text{poly}
\]
\[
tf \text{ SetTransform } tr
\]
\[
tf \text{ Update}
\]
This will operate using transformation \(tr\) on each point.

Transforming a Surface II

We can create a transformation that first translates by \((10,5,0)\), then rotates by \(20°\) about the Z-axis and finally magnifies the object by a factor of 2 as:

\[
\text{vtkTransform } tr
\]
\[
tr \text{ PostMultiply}
\]
\[
tr \text{ Translate } 10 \ 5 \ 0
\]
\[
tr \text{ RotateZ } 20
\]
\[
tr \text{ Scale } 2 \ 2 \ 2
\]

Explicit 4x4 Matrix Definition

We can explicitly specify an arbitrary 4x4 matrix transformation using \(\text{vtkMatrix4x4}\) (in this case we specify a translation of \((10,0,0)\))

\[
\text{vtkMatrix4x4 } \text{mat}
\]
\[
\text{mat Identity}
\]
\[
\text{mat SetElement } 0 \ 3 \ 10.0
\]
\[
\text{vtkTransform } tr
\]
\[
tr \text{ SetMatrix } \text{mat}
\]
\[
\text{mat Delete}
\]

Reslicing an Image

1. In reslicing images we compute a transformation \(T\) which maps positions on the reference image to positions in the transform image, \(e.g. r' = F(r)\).
2. The standard image reslicing algorithms \(\text{vtkImageReslice}\) then takes the intensity at \(r'\) \(T(r')\) and places it at position \(r\) in the new resampled image.
3. Hence while \(F\) maps points from \(R\) to \(T\), it maps intensities from \(T\) to \(R\)!!

\[
\text{vtkImageReslice}
\]

- \(\text{vtkImageReslice}\) is one of the most powerful classes in VTK.
- It can be used to reslice, resample and reorient images using different transformations and interpolation strategies.
- In its simplest complete usage it takes a number of parameters the most useful being:
  - The input image \(T\) to be resliced
  - The information input image \(R\) that defines the output geometry \([\text{Note that the output geometry can also be defined explicitly without using an information input}]\)
  - The reslice transformation that maps \(R\) to \(T\) \([\text{identity if not specified}]\)
  - The interpolation strategy used to interpolate \(T\) \([\text{nearestneighbor, linear, cubic}]\)
An Example Using a GUI examples\example6_1.tcl

- In previous examples we used vtkRenderWindow and vtkImageWindow as the final output window on the display. These are separate from the rest of the GUI and to make more integrated applications vtk supplies
  - vtkTkImageWindowWidget – a Tk widget that contains a vtkImageWindow
  - vtkTkRenderWindowWidget – a Tk widget that contains a vtkRenderWindow
- These widgets can be used to embed vtk windows into a Tk-based graphical user interface
- The underlying vtkImageWindow and vtkRenderWindow are accessed using the GetImageViewer and GetRenderWindow methods respectively.

Image Reslice Example – the GUI

```tcl
# Set the Window size to 400 x 250
wm geometry . 400x250
# Create Two frames, .top and .bottom and pack them
frame .top ; frame .bottom
pack .bottom -side bottom -expand false -fill x -pady 2 -padx 20
pack .top -side top -expand true -fill both
# Add an exit button to the bottom frame and pack it
button .bottom.exit-text "Exit!" -command { destroy . ; exit }
pack .bottom.exit-side left
# Create two frames in the top frame called .left and .right
frame .top.left -bg green -height 200; frame .top.right -bg red -height 200
pack .top.left.top.right -side left -expand true -fill both -pad -pady 2
# Add labels to left and right to label the images to be displayed!
label .top.left.lab-text "Original"
pack .top.left.lab-side top -expand false -fill
label .top.right.lab-text "Resliced"
pack .top.right.lab-side top -expand false -fill
```

Image Reslice Example – the Viewers

```tcl
# Create the left viewer storing reference in v1
set v1 [ vtkTkImageViewerWidget.top.left.v -height 170 -width 170 ]
pack .top.left.v -side bottom -expand true -fill both -pady 2
# Create the right viewer storing reference in v2
set v2 [ vtkTkImageViewerWidget.top.right.v -height 170 -width 170 ]
pack .top.right.v -side bottom -expand true -fill both -pady 2
# This is important to synchronize the toolkit with Open GL
# Force GUI to be drawn
Update
# Get the Image Viewers and store in vleft and vright
# vleft and vright are of type vtkImageWindow that we have seen before
set vleft [ $v1 GetImageViewer ]
set vright [ $v2 GetImageViewer ]
```

Image Reslice Example – the Reslicing Code

```tcl
# Create the TIFF reader
vtkTIFFReader tr
tr SetFileName examples\brain.tif
tr Update
# This creates a rotation of 10 about the center of the image
# by i) shifting the image center to the origin ii) rotating and
# iii) shifting the image center back to its original position
vtkTransform xform
xform PostMultiply
xform Translate -62.0 -80.0 0
xform RotateZ 10.0
xform Translate 62.0 80.0 0
# This is the reslice, note that we use the image itself to define the reference
# geometry (not always a good idea!)
vtkImageReslice resl
resl SetInput [ tr GetOutput ]
resl SetInformationInput [ tr GetOutput ]
resl SetResliceTransform xform
resl SetInterpolationModeToLinear
resl Update
```

Image Reslice Example – the Display Code

```tcl
# Set The Original Image as the input to the left viewer
vleft SetInput [ tr GetOutput ]
vleft SetZSlice 0
vleft SetColorLevel 128
vleft SetColorWindow 255
delv left Render
# Set The Resliced Image as the input to the right viewer
vright SetInput [ resl GetOutput ]
vright SetZSlice 0
vright SetColorLevel 128
vright SetColorWindow 255
delv right Render
# Start the event loop
vwait forever
```

A Complete Application examples\example_viewer.tcl

(Also get MNI Resampled Brain Image: examples\mni\bin.vrt)
The Example Viewer 2

- 400 Lines of Tcl Code of which 120 are comments and 70 blank lines
- Tcl Constructs Used
  1. Graphical User Interface
     Menu, Checkbutton, OptionMenu, Scale, Label, Entry
  2. Global/Local Scoping
     Use of the **global** command
  3. Associative Arrays
  4. Standard Dialogs (FileOpen, Message)
     `tk_getOpenFile`, `tk_messageBox`
  5. Modularization and Procedures

The Example Viewer 2

- 400 Lines of Tcl Code of which 120 are comments and 70 blank lines
- Tcl Constructs Used
  1. Graphical User Interface
     Menu, Checkbutton, OptionMenu, Scale, Label, Entry
  2. Global/Local Scoping
     Use of the **global** command
  3. Associative Arrays
  4. Standard Dialogs (FileOpen, Message)
     `tk_getOpenFile`, `tk_messageBox`
  5. Modularization and Procedures

The Example Viewer 3

- VTK Objects Used
  1. `vtkImageData` — to store the images
  2. `vtkStructuredPointsReader` — to load the images
  3. `vtkImagePermute` — to reorder the voxels so as to display axial/coronal/sagittal slices
  4. `vtkImageFlip` — to flip the images left/right, top/bottom
  5. `vtkImageResample` — to rescale the images for magnification
  6. `vtkImageViewer` (via `vtkTkImageViewerWidget`) — to display the images

Program Structure

1. The program has 12 Procedures and 2 Global Variables, one of which is an associative array that is used to store the state of the application
2. Four procedures are used to build the Graphical User Interface, `GenerateUI` — main procedure which calls
   1. `GenerateMenuUI`
   2. `BuildControlFrameUI`
   3. `CreateViewer`
3. Three procedures are used to manage the global associate array `params`:
   - `InitializeParameters` — called at start of program
   - `ResetParametersAndGUI` — called when new image is loaded
   - `DebugParameters` — prints all parameters as a debugging mechanism
4. Three procedures that deal with images directly
   1. `LoadImage`
   2. `PermuteImage`
   3. `FlipScaleAndDisplayImage`
5. An `UpdateDisplay` procedure to update the display
6. A generic utility procedure `UniqueId` to ensure unique VTK object names.

The Main Program

```tcl
# Hide Default Tk Widget
wm withdraw .

# Create a new toplevel widget and set its dimensions and name
toplevel .base
wm geometry .base 600x450

# Initialize Variables
InitializeParameters

# Create The User Interface
GenerateUI .base

# Load an Image
LoadImage

# Start the Event Loop
vwait forever
```

Generating Unique Names

```tcl
# Include Counter Code
# Define proc UniqueId which returns a unique name each time it is called. Note the use of the global command to bring a global variable into local scope, global variables are not by default accessible outside the global scope
# i.e., within a procedure
# ...
set unique_objcounter 0 proc UniqueId { } {
  global unique_objcounter
  incr unique_objcounter
  return "obj_${unique_objcounter}"}

UniqueId obj_1, call again UniqueId obj_2 etc.
```
InitializeParameters

proc InitializeParameters {} {
    global params
    set params(slice) XY; set params(scale) 2.0
    set params(flipx) 0; set params(flipy) 1
    set params(window) 255; set params(level) 128
    set params(sliceno) 0
    set params(viewer) 0; set params(scalewidget) 0
    set params(windowscalewidget) 0
    set params(levelscalewidget) 0
    set params(slicenoscalewidget) 0
    set params(mainwidget) 0
    set params(status) "No Image In Memory"
    set params(currentimage) [ vtkImageData [ UniqueId ] ]
    set params(currentresults) [ vtkImageData [ UniqueId ] ]
}

DebugParameters

# Standard Code to List an Associative Array
proc DebugParameters {} {
    global params
    puts stdout "\n Debugging Parameters"
    puts stdout "--------------------------"
    set keylist [ array names params ]
    for { set k 0 } { $k < [llength $keylist] } { incr k } {
        set key [lindex $keylist $k]
        set value $params($key)
        puts "params($key) = $value"
    }
    puts stdout "--------------------------"
}

GenerateUI

proc GenerateUI { basewidget } {
    global params
    set params(mainwidget) $basewidget
    set basewidget.title $params(mainwidget) "Example Viewer"
    set basewidget.filetop [ frame $basewidget.top ]
    set basewidget.middle [ frame $basewidget.middle ]
    set basewidget.top0 [ frame $basewidget.middle -bg black ]
    set basewidget.top status [ entry $basewidget.status -textvariable params(status) ]
    pack $basewidget.status -side bottom -expand false -fill x -pady 1
    pack $basewidget.menu -side top -expand true -fill both -padx 2 -pady 2
    pack $basewidget.left -side left -bg black -fg white
    set basewidget.left [ frame $basewidget.left -bg black -fg white ]
    pack $basewidget.left -side left -bg black -fg white
    GenerateMenuUI $basewidget.menu
    BuildControlFrameGUI $basewidget.left
    CreateViewer $basewidget
}

GenerateMenuUI

proc GenerateMenuUI { menu } {
    menu $menu.file -text "File" -menu $menu.file.m
    menu $menu.help -text "Help" -menu $menu.help.m
    pack $menu.file -side left; pack $menu.help -side right
    set filemenu [ menu $menu.file.m ]
    filemenu add command -label "Open" -command { LoadImage }
    filemenu add separator
    filemenu add command -label "Exit" -command { destroy ; exit }
    set helpmenu [ menu $menu.help.m ]
    helpmenu add command -label "About!" -command { DebugParameters }
    helpmenu add separator
    helpmenu add command -label "About!" -command { tk_messageBox -title "About This Application" -type ok -message "Example Image Viewer\nVersion 1.0\nX.Papademetris Nov 2002" ]
}

BuildControlFrameGUI

The real strength of TK GUIs is the ability to link variables to graphical user elements i.e. checkbuttons, scales, optionmenus etc.

- params(slice) -- controls what kind of slice to display
- params(flipx) -- 0/1 as to whether to flip the x-axis
- params(flipy) -- 0/1 as to whether to flip the y-axis
- params(scale) -- expand the image by this factor
- params(sliceno) -- actual slice number to display
- params(window) -- sets the window width (colormap)
- params(level) -- sets the window level (colormap)

The code is far too boring and repetitive to be covered orally see the code! Note only that:
1. Changing the UI element changes the value of the linked variable
2. Changing UI elements also results in the call of the appropriate event handling procedure (in yellow above).
CreateViewer

```tcl
proc CreateViewer { viewerframe } {
    global params
    vtkTkImageViewerWidget $viewerframe.right -height 256 -width 256
    pack $viewerframe.right -side right -expand true -fill both -pady 2
    # Important: ensure that the widget has been created
    update
    # Store viewer
    set params(viewer) [ $viewerframe.right GetImageViewer ]
}
```

LoadImage

```tcl
proc LoadImage {} {
    global params
    # Get Filename
    set f [tk_getOpenFile -title "Load Image" -filetypes { {"VTK Image Format" {.vt}} }] 
    if { [ string length $f ] < 1 } { return 0 } 
    # Load Image
    set reader [ vtkStructuredPointsReader [UniqueId ]]
    $reader SetFileName $f; $reader Update
    $params(currentimage) ShallowCopy [ $reader GetOutput ]
    $reader Delete
    # Set Application Title
    wm title $params(mainwidget) "Example Viewer : [ file tail $f ]"
    # Process Image
    ResetParametersAndGUI
    PermuteImage
    FlipScaleAndDisplayImage
}
```

ResetParametersAndGUI

```tcl
proc ResetParametersAndGUI {} {
    global params
    # Default View
    set params(slice) XY; set params(flipx) 0; set params(flipy) 1
    set params(scale) 2.0
    # Get Intensity Range and Set ColorMap and GUI Ranges
    set meanrange [ expr 0.5*([ lindex $range 0 ] + [ lindex $range 1 ])]
    set fullrange [ expr [ lindex $range 1 ] - [ lindex $range 0 ]]
    $params(windowscalewidget) configure -from 0 -to $fullrange
    $params(levelscalewidget) configure -from [ lindex $range 0 ] -to [ lindex $range 1 ]
    set params(window) $meanrange; set params(level) $meanrange
    # Set The Status Bar To reflect new image
    set img $params(currentimage); set dim [ $img GetDimensions ]
    set vxlsize [ $img GetSpacing ]
    set params(status) "Image Dim = $dim Vox= $vxlsize Range = $range"
}
```

PermuteImage

```tcl
proc PermuteImage {} {
    global params
    set perm [ vtkImagePermute [UniqueId ]]
    $perm SetInput $params(currentimage)
    # Set The Axis to Map to X-Y-Z e.g. 0 2 1 X->X Z->Y Y->Z
    switch $params(slice) {
        XY   { $perm SetFilteredAxes 0 1 2 }
        YX   { $perm SetFilteredAxes 1 0 2 }
        XZ   { $perm SetFilteredAxes 0 2 1 }
        ZX   { $perm SetFilteredAxes 2 0 1 }
        YZ   { $perm SetFilteredAxes 1 2 0 }
        ZY   { $perm SetFilteredAxes 2 1 0 }
    }
    $perm Update
    $params(currentresults) ShallowCopy [ $perm GetOutput ]; $perm Delete
    # Get Number of Slices for this orientation and update GUI
    set numslices [lindex [ $params(currentresults) GetDimensions ] 2 ]
    $params(slicenoscalewidget) configure -from 0 -to [ expr $numslices -1 ]
    set params(sliceno) [ expr $numslices/2 ]
}
```

FlipScaleAndDisplayImage

```tcl
proc FlipScaleAndDisplayImage {} {
    global params
    # Three Filter Pipeline, flipx,flipy,scale
    set tmp [ vtkImageData [UniqueId ]]
    $tmp ShallowCopy $params(currentresults)
    if { $params(flipx) == 1 } {
        set fx [ vtkImageFlip [UniqueId ]]
        $fx SetInput $tmp; $fx SetFilteredAxis 0; $fx Update
        $tmp ShallowCopy [ $fx GetOutput ]; $fx Delete
    } .. Same for y axis
    if { $params(flipy) == 1 } { .. By SetFilteredAxis f ...}
    $imageResample [ vtkImageResample [UniqueId ]]
    $imageResample SetInput $tmp; $imageResample InterpolateOn
    $imageResample SetAxisMagnificationFactor 0 $params(scale)
    $imageResample SetAxisMagnificationFactor 1 $params(scale)
    $imageResample SetAxisMagnificationFactor 2 1.0
    $params(viewer) SetInput [ $imageResample GetOutput ]; $imageResample Delete; $tmp Delete
    UpdateDisplay
}
```

UpdateDisplay

```tcl
proc UpdateDisplay {} {
    global params
    # round() ensures integer output
    $params(viewer) SetZSlice [ expr round($params(sliceno))]
    $params(viewer) SetColorLevel $params(level)
    $params(viewer) SetColorWindow $params(window)
    $params(viewer) Render
}
```