Technical University of Łódź
Institute of Electronics
Medical Electronics Division

Image Processing and Computer Graphics

The Visualization Toolkit (VTK)

Author: Marek Kociński

April 2010

1 Purpose

To get acquainted with basic capabilities with the Visualization Toolkit (VTK). The VTK is an open-source, freely available software system for $3D$ computer graphics, image processing and visualization. VTK is cross-platform and runs on Linux, Windows, Mac and Unix platforms.

Time

4×45 minutes

2 The Graphics Model

There are seven basic objects that we use to render a scene. Documentation of all objects and classes used in vtk library is available on the webpage: http://www.vtk.org/doc/release/5.4/html/classes.html.

1. *vtkRenderWindow* — manages a window on the display device; one or more renderers draw into an instance of vtkRenderWindow.
2. *vtkRenderer* — coordinates the rendering process involving lights, cameras, and actors.
3. *vtkLight* — a source of light to illuminate the scene.
4. *vtkCamera* — defines the view position, focal point and other viewing properties of the scene.
5. *vtkActor* — represents an object rendered in the scene, including its properties (color, shading type, etc.) and position in the words coordinate system. (Note: vtkActor is a subclass of vktProp. vtkProp is a more general form of actor that includes annotation and 2D drawing classes.)
6. *vtkProperty* — defines the appearance properties of an actor including color, transparency, and lighting properties such as specular and diffuse. Also representational properties like wireframe and solid surface.
7. *vtkMapper* — the geometric representation for an actor. More than one actor may refer to the same mapper.

3 Tasks

1. Open Python interpreter window (Start→ Programy→ EPD32-6.0.2 → IDLE)
2. Open new Editor Window (File → New Window) and write your code into it.

3. Import needed modules, e.g. *vtk*.

4. Count distance between two points. In this example additional package *math* is needed.

```
import vtk
import math

p0 = (0,0,0)
p1 = (1,1,1)

distSquared = vtk.vtkMath.Distance2BetweenPoints(p0,p1)
dist = math.sqrt(distSquared)

print "p0 = ", p0
print "p1 = ", p1
print "distance squared = ", distSquared
print "distance = ", dist
```

5. Draw triangle on the black background (Fig. 1). Pay attention to used pipeline of the basic vtk objects in the graphic model.

```
import vtk

# create a rendering window and renderer
ren = vtk.vtkRenderer()
renWin = vtk.vtkRenderWindow()
renWin.AddRenderer(ren)

# create a renderwindowinteractor
iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)

# create points
points = vtk.vtkPoints()
points.InsertNextPoint(1.0,0.0,0.0)
points.InsertNextPoint(0.0,0.0,0.0)
points.InsertNextPoint(0.0,1.0,0.0)

triangle = vtk.vtkTriangle()
triangle.GetPointIds().SetId(0,0)
triangle.GetPointIds().SetId(1,1)
triangle.GetPointIds().SetId(2,2)

triangles = vtk.vtkCellArray()
```

```
triangles.InsertNextCell(triangle)

# polydata object
trianglePolyData = vtk.vtkPolyData()
trianglePolyData.SetPoints( points )
trianglePolyData.SetPolys( triangles )

# mapper
mapper = vtk.vtkPolyDataMapper()
mapper.SetInput(trianglePolyData)

# actor
actor = vtk.vtkActor()
actor.SetMapper(mapper)

# assign actor to the renderer
ren.AddActor(actor)

# enable user interface interactor
iren.Initialize()
renWin.Render()
iren.Start()
```

6. Draw a sphere, use *vtkSphereSource* class. Change some of the parameters: *PhiResolution*, *ThetaResolution*, *Radius* and *Position* of the sphere in the $3D$ space.

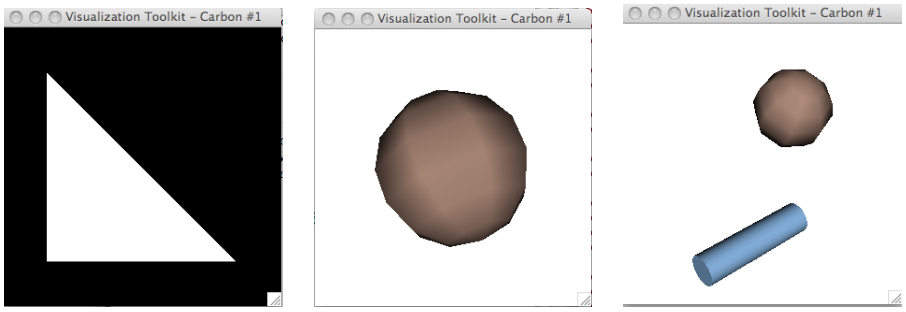
```
# create source
source = vtk.vtkSphereSource()
source.SetCenter(0,0,0)
source.SetRadius(5.0)
```

7. Change some of the surface properties of the sphere with the use of *GetProperty()* object:

   - *SetColor()* — RGB color in range (0.0–1.0)
   - *SetDiffuse()* — in range (0.0–1.0)
   - *SetSpecular()* — in range (0.0–1.0)
   - *SetSpecularPower()* — in range (0–255)

   and background color using *SetBackground(...)* method on the **renderer** object (Fig. 1).

8. With the use of *vtkCylinderSource* object draw cylinder. Use additional *vtkPolyDataMapper* and *vtkActor* for this purpose (Fig 1).



(a) Triangle *(b) Sphere* *(c) Sphere and cylinder*

Figure 1: Basic objects in the $3D$ space

9. It is possible to divide *RenderWindow* among few *Renderers* (see Fig. 2).
   (a) create 4 renderers (*vtkRenderer* class)
   (b) set different colors for each of them with the use of *SetBackground(...)* function
   (c) put every renderer in appropriate position inside RendererWindow
      - *ren1.SetViewport(0.0,0.0,0.5,0)*
      - *ren2.SetViewport(0.5,0.0,1.0,0.5)*
      - *ren3.SetViewport(0.0,0.5,0.5,1.0)*
      - *ren4.SetViewport(0.5,0.5,1.0,1.0)*
   (d) add each renderer to renderer window (use *AddRenderer(...)* function)
   (e) create 4 different $3D$ objects to render in every renderer:
      - Cone
        - use: *vtkConeSource, SetCenter(...),SetHeight(...), SetRadius(...), SetResolution(...), SetAngle(...)*
      - Cube
        - use: *vtkCubeSource, SetXLength(...), SetYLenght(...), SetZLength(...), SetCenter(...)*
      - Use other objects e.g.: *vtkArrowSource, vtkTextCource, vtkDiskSource, vtkEarthSource, vtkTexturedSphereSource, vtkPlaneSource,...*
   (f) for each $3D$ object create mapper and actor (*vtkPolyDataMapper, vtkActor*)
   (g) add actors to the rendreres (*AddActor(...)*)

10. *VTK* has implemented many components to image processing. To read and display $2D$ image it is enough to run code as follows (Fig. 3)

```
import vtk

reader = vtk.vtkBMPReader()
reader.SetFileName ("lenna.bmp")
```

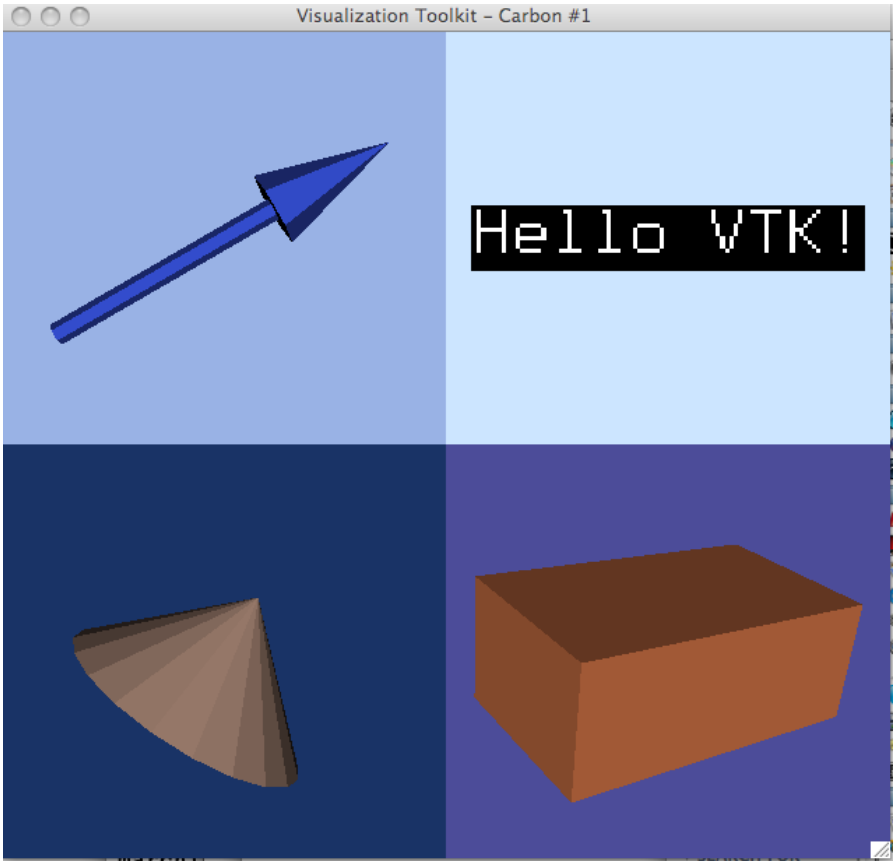


Figure 2: *Four renderers in one window*

```
iren = vtk.vtkRenderWindowInteractor()

viewer = vtk.vtkImageViewer2()
viewer.SetupInteractor(iren)
viewer.SetInputConnection(reader.GetOutputPort())
viewer.SetColorLevel(125)
viewer.SetColorWindow(255)
viewer.Render()

iren.Start()
```

11. It is easy to warp image in the direction perpendicular to the image plane using the visualization filter *vtkWartScalr*. Set few values for *SetScaleFactor* (Fig. 4).

```
import vtk
```

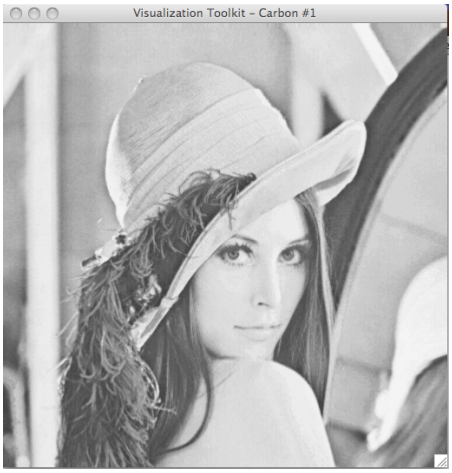


Figure 3: Imaged displayed with the use of *vtkImageViewer2*. Mouse buttons manipulation changes contrast and brightness of the image

```
reader = vtk.vtkBMPReader()
reader.SetFileName ("lenna.bmp")

imgGeometry = vtk.vtkImageDataGeometryFilter()
imgGeometry .SetInput (reader .GetOutput ())

warp = vtk.vtkWarpScalar ()
warp .SetInput (imgGeometry .GetOutput ())
warp .SetScaleFactor (0.7)

wl = vtk.vtkWindowLevelLookupTable ()

mapper = vtk.vtkPolyDataMapper ()
mapper .SetInputConnection (warp .GetOutputPort ())
mapper .SetScalarRange (0,2000)
mapper .ImmediateModeRenderingOff ()
mapper .SetLookupTable (wl)

imageActor = vtk.vtkImageActor ()
imageActor .SetInput (reader .GetOutput ())

warpActor = vtk.vtkActor ()
warpActor .SetMapper (mapper)

ren1 = vtk.vtkRenderer()
ren1 .SetBackground (0.2,0.2,0.4)
```

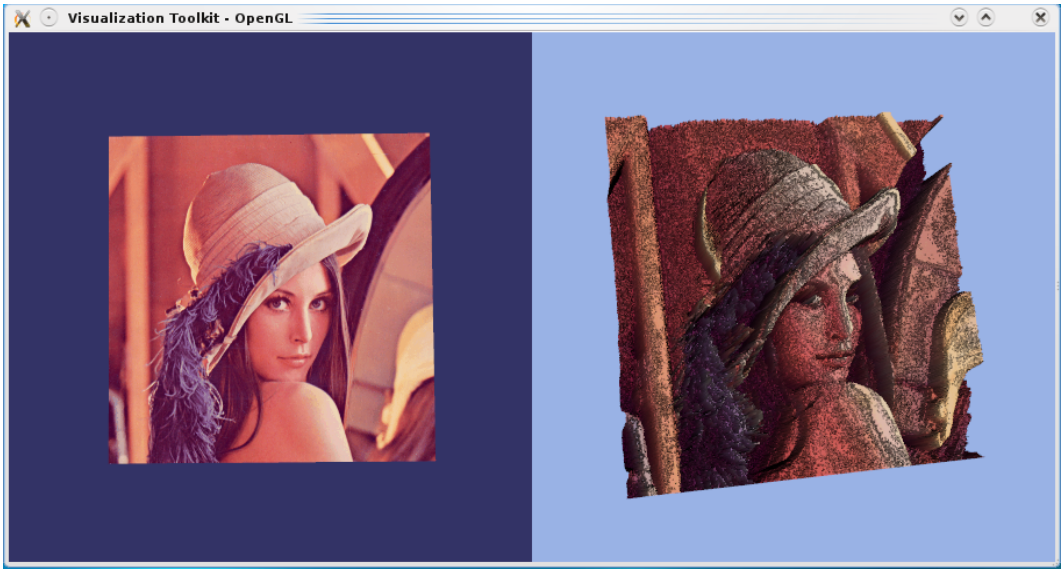
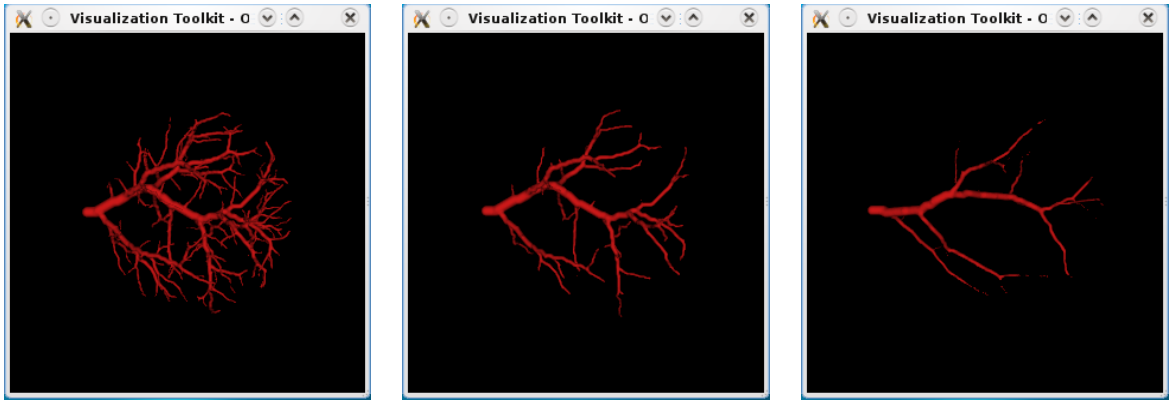


Figure 4: Imaged displayed with the use of *vtkImageViewer2* and warped the data in the direction perpendicular to the image plane

```
ren1 .AddActor (imageActor )
ren1 .SetViewport (0.0, 0.0, 0.5, 1.0)

ren2 = vtk .vtkRenderer()
ren2 .SetBackground( 0.6, 0.7, 0.9)
ren2 .SetViewport (0.5, 0.0, 1.0, 1.0)
ren2 .AddActor (warpActor )

renderWindowInteractor = vtk.vtkRenderWindowInteractor ()
renWin =vtk.vtkRenderWindow ()
renWin .AddRenderer (ren1)
renWin .AddRenderer (ren2)
renWin .SetInteractor (renderWindowInteractor )
renWin .SetSize (900,450)
renWin .Render ()

renderWindowInteractor .Start ()
```

12. Iso-surface extraction is possible with the use of *vtkMarchingCubes* algorithm. Run follwing code. Play with *SetValue(...)* function in range (10–255) (Fig. 5).

```
import vtk

imageReader = vtk.vtkImageReader()
```

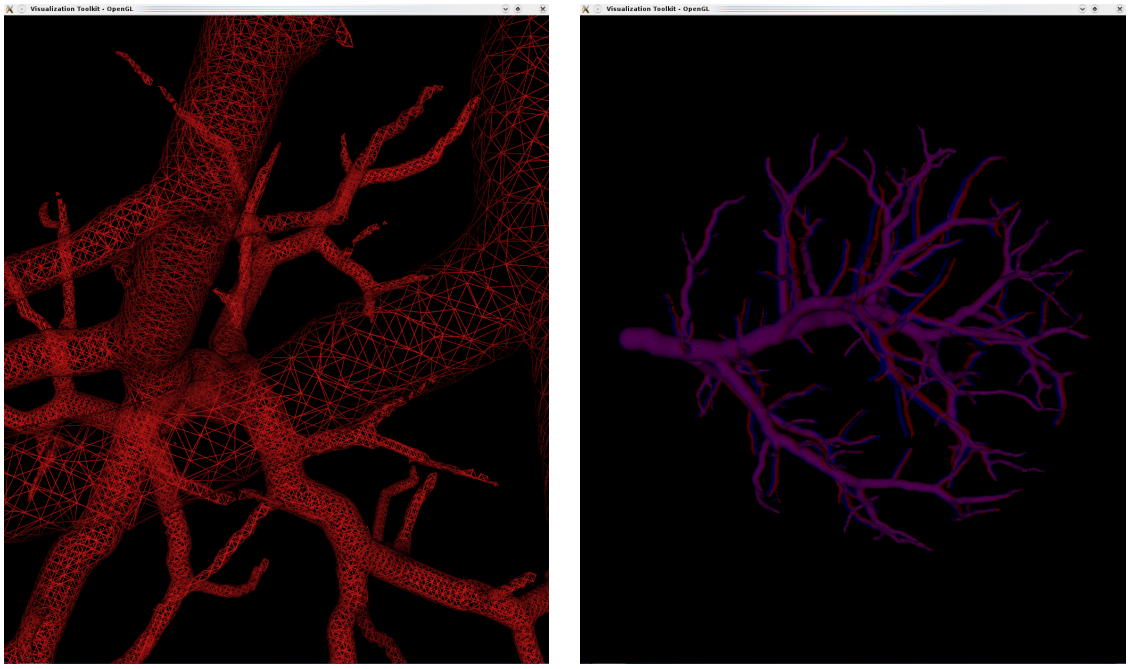


(a) iso-value=50 *(b) iso-value=100* *(c) iso-value=200*

Figure 5: Extraction of surface of vascular tree with different iso-value

```
imageReader.SetFileName("qinp01_3000_036_3_256.raw")
imageReader.SetDataScalarTypeToUnsignedChar()
imageReader.SetDataByteOrder(0)
imageReader.SetFileDimensionality(3)
imageReader.SetDataOrigin(0,0,0)
imageReader.SetDataSpacing(1,1,1)
imageReader.SetDataExtent(0,255,0,255,0,255)
imageReader.SetNumberOfScalarComponents(1)
imageReader.Update()


shrinker = vtk.vtkImageShrink3D()
shrinker.SetInput(imageReader.GetOutput())
shrinker.SetShrinkFactors(1,1,1)
shrinker.AveragingOn()

gaussian = vtk.vtkImageGaussianSmooth()
gaussian.SetDimensionality(3)
gaussian.SetStandardDeviations(1.0, 1.0, 1.0)
gaussian.SetRadiusFactor(1.0)
gaussian.SetInput(shrinker.GetOutput())

marching = vtk.vtkMarchingCubes()
marching.SetInput(gaussian.GetOutput())
marching.SetValue(1,100)
marching.ComputeScalarsOff()
marching.ComputeGradientsOff()
marching.ComputeNormalsOff()
```

```
decimator = vtk.vtkDecimatePro ()
decimator .SetInput (marching .GetOutput ())
decimator .SetTargetReduction (0.1)
decimator .SetFeatureAngle (60)

smoother = vtk .vtkSmoothPolyDataFilter()
smoother .SetInput (decimator.GetOutput())
smoother .BoundarySmoothingOn()
smoother .FeatureEdgeSmoothingOn ()

normals = vtk.vtkPolyDataNormals ()
normals .SetInput (smoother .GetOutput())
normals .SetFeatureAngle (60)

stripper = vtk.vtkStripper ()
stripper .SetInput (normals.GetOutput ())

mapper = vtk.vtkPolyDataMapper ()
mapper .SetInput (stripper .GetOutput())
mapper .ScalarVisibilityOff ()

surf = vtk.vtkProperty ()
surf .SetColor(0.8,0.1,0.1)


actor = vtk.vtkActor()
actor .SetMapper (mapper)
actor .SetProperty (surf)

ren1 = vtk.vtkRenderer()
ren1 .AddActor (actor)

renWin = vtk.vtkRenderWindow ()
renWin .AddRenderer (ren1)

iren = vtk.vtkRenderWindowInteractor ()
iren .SetRenderWindow (renWin)

renWin.Render()
iren .Start ()
```



(a) *mesh* of extracted vascular tree *stereo* mode

Figure 6: Different modes implemented in *vtk*

13. By pressing "w"/"s" key, one can switch between *wire* and *surfce* mode. $3D$ *stereo normal* mode is accessible with key: "3"