Introduction to VTK and Paraview

Scientific Visualisation
HT 2018
Lecture 2

Fredrik Nysjö
Centre for Image analysis
Uppsala University
The Visualization Toolkit (VTK)

- Open source, freely available C++ toolkit for
  - scientific visualization
  - 3D computer graphics
  - mesh and image processing
- Managed by Kitware Inc.
VTK

- Object-oriented design
- High level of abstraction (compared to graphics APIs like OpenGL or Direct3D)
- Provides bindings to Tcl/Tk, Python, and Java
- GUI bindings: Qt, wxWidgets, Tkinter, etc
Heavily object-oriented (and a bit over-designed...)
Some examples of what you can do with VTK

- Create visualizations of
  - scalar, vector, and tensor fields
  - volume data (e.g., 3D CT or MRI scans)
- Mesh and polygon processing
- Image analysis (2D and 3D images)
- Isosurface extraction
- Implement your own algorithms
Volume rendering
Rendering graphical 3D models (imported from .stl, .ply, .obj, etc)
Rendering performance

- VTK has decent rendering performance and is good for prototyping 3D visualization tools
- Not suitable for demanding real-time rendering applications (games, VR)
The visualization pipeline

# vtk DataFile Version 3.0
vtk output
BINARY
DATASET STRUCTURED_POINTS
DIMENSIONS 256 256 124
SPACING 0.9 0.9 0.9
ORIGIN 0 0 0
CELL_DATA 7998075
POINT_DATA 8126464
COLOR_SCALARS ImageFile 1
^D^E^C^G^D^B^B^B^C^D^E^D^E^C^C
^D^C^C^C^C^E^D^C^A^B^B^B^F^A^C^E
^D^D^E^A^A^C^B^B^E^B^A^A^E^B^E^E
^A^C^C^G^C^D^F^B^D^E^@^G^C^D^D^C
^D^C^F^C^B^E^E^E^B^C^C^B^C^B^C^B
^C^C^F^E^F^C^D^A^A^C^F^D^D^E^E^B

Input data

Visualization
The visualization pipeline

• To visualize your data in VTK, you normally set up a pipeline like this:
Sources

- VTK provides various source classes that can be used to construct simple geometric objects like spheres, cubes, cones, cylinders, etc...

- Examples: \texttt{vtkSphereSource}, \texttt{vtkCubeSource}, \texttt{vtkConeSource}

\texttt{source/reader} → \texttt{filter} → \texttt{mapper} → \texttt{actor} → \texttt{renderer} → \texttt{renderWindow} → \texttt{interactor}
Readers

- Reads data from file
- You can use, e.g., \texttt{vtkStructuredPointsReader} to read a volumetric image from a .vtk file
- or \texttt{vtkSTLReader} to load a 3D polygon model from a .stl file
- If VTK cannot read your data, write your own reader!

source/reader $\rightarrow$ filter $\rightarrow$ mapper $\rightarrow$ actor $\rightarrow$ renderer $\rightarrow$ renderWindow $\rightarrow$ interactor
Filters

- Takes data as input, modifies it in some way, and returns the modified data
- Can be used to (for example)
  - select data of a particular size, strength, intensity, etc
  - process 2D/3D images or polygon meshes
  - generate geometric objects from data

source/reader → filter → mapper → actor → renderer → renderWindow → interactor
Mappers

- Maps data to graphics primitives (points, lines, or triangles) that can be displayed by the renderer
- Example: `vtkPolyDataMapper` maps polygonal data (`vtkPolyData`) to graphics primitives

source/reader $\rightarrow$ filter $\rightarrow$ mapper $\rightarrow$ actor $\rightarrow$ renderer $\rightarrow$ renderWindow $\rightarrow$ interactor

Image source: http://www.realtimerendering.com
Actors

- **vtkActor** represents an object (geometry and properties) in a rendering scene.
- Has position, scale, orientation, various rendering properties, textures, etc. Keeps a reference to the mapper.

source/reader → filter → mapper → actor → renderer → renderWindow → interactor
Renderer

- **vtkRenderer** controls the rendering process for actors and scenes
- Under the hood, VTK uses OpenGL for rendering

source/reader → filter → mapper → actor → **renderer** → renderWindow → interactor

Image source: http://www.realtimerendering.com
Render window

- The `vtkRenderWindow` class creates a window for renderers to draw into

source/reader → filter → mapper → actor → renderer → `renderWindow` → interactor
Interactors

- The `vtkRenderWindowInteractor` class provides platform-independent window interaction via the mouse and keyboard.
- Allows you to rotate/zoom/pan the camera, select and manipulate actors, etc.
- Also handles time events.

source/reader → filter → mapper → actor → renderer → renderWindow → interactor
Example 1: Rendering a cube
Pipeline for the cube example

1. `vtkCubeSource`
2. `vtkPolyDataMapper`
3. `vtkActor`
4. `vtkRenderer`
5. `vtkRenderWindow`
6. `vtkRenderWindowInteractor`
import vtk

# Generate polygon data for a cube
cube = vtk.vtkCubeSource()
# Create a mapper for the cube data
cube_mapper = vtk.vtkPolyDataMapper()
cube_mapper.SetInputConnection(
    cube.GetOutputPort())

# This will not work in VTK 6!
cube_mapper.SetInput(cube.GetOutputPort())

source-reader \rightarrow filter \rightarrow \textcolor{red}{mapper} \rightarrow \text{actor} \rightarrow \text{renderer} \rightarrow \text{renderWindow} \rightarrow \text{interactor}
# Connect the mapper to an actor

cube_actor = vtk.vtkActor()
cube_actor.SetMapper(cube_mapper)
cube_actor.GetProperty().SetColor(1.0, 0.0, 0.0)
# Create a renderer and add the cube actor to it
renderer = vtk.vtkRenderer()
renderer.SetBackground(0.0, 0.0, 0.0)
renderer.AddActor(cube_actor)

debug

source/reader → filter → mapper → actor → renderer → renderWindow → interactor
Render window

```python
# Create a render window
render_window = vtk.vtkRenderWindow()
render_window.SetWindowName("Simple VTK scene")
render_window.SetSize(400, 400)
render_window.AddRenderer(renderer)
```

source/reader → filter → mapper → actor → renderer → `renderWindow` → interactor
# Create an interactor
interactor = vtk.vtkRenderWindowInteractor()
interactor.SetRenderWindow(render_window)

# Initialize the interactor and start the rendering loop
interactor.Initialize()
render_window.Render()
interactor.Start()
Example 2:
Earthquake data

1: Change colormap
2: Toggle map
4: Toggle grid
5: Toggle heightmap
6: Toggle outline
7: Start animation
Visualizing the quakes with sphere glyphs

1: Change colormap
2: Toggle map
4: Toggle grid
5: Toggle heightmap
6: Toggle outline
7: Start animation
Sphere glyphs

**Strength**
- `vtkFloatArray`
- `vtkPolyData`
- `vtkGlyph3D`
- `vtkPolyDataMapper`
- `vtkActor`

**Position**
- `vtkPoints`
- `vtkSphereSource`
- `vtkColorTransferFunction`
See this paper for a discussion on why the "rainbow" colormap is a poor choice for most applications.
Colormaps

1: Change colormap
2: Toggle map
4: Toggle grid
5: Toggle heightmap
6: Toggle outline
7: Start animation

Strength

Top view

Side view

Front view

Longitude
Latitude
Depth
Example 3:
Air currents
Arrow glyphs, first try
Arrow glyphs, first try

Direction and speed

vtkStructuredPoints → vtkGlyph3D

vtkArrowSource → vtkColorTransferFunction

vtkPolyDataMapper

vtkActor
Cut planes

Direction and speed

vtkStructuredPoints

vtkImageDataGeometryFilter

vtkPolyDataMapper

vtkColorTransferFunction

vtkActor
Arrow glyphs, second try
Arrow glyphs, second try

Direction and speed

vtkStructuredPoints

vtkImageDataGeometryFilter  vtkArrowSource

vtkGlyph3D  vtkColorTransferFunction

vtkPolyDataMapper

vtkActor
Streamtubes
Streamtubes

Direction and speed
- vtkStructuredPoints

Seeds (starting points)
- vtkPointSource
- vtkStreamLine

vtkTubeFilter
- vtkColorTransferFunction

vtkPolyDataMapper

vtkActor
Example 4: Medical 3D data
Outline

1: Toggle MPR
2: Toggle segmentation
Multi-planar reformatting (MPR)
Multi-planar reformatting (MPR)
Surface rendering

1: Toggle MPR
2: Toggle segmentation
Surface rendering

Segmented volume image

vtkStructuredPoints

vtkImageGaussianSmooth

vtkContourFilter

vtkPolyDataMapper

vtkActor
Combined visualization
VTK - Summary

- VTK contains thousands of classes and might seem a bit overwhelming...
  - however, one can create useful visualizations with just a few core classes
- The pipeline is typically

```
Source/Reader ➔ Filter ➔ Mapper ➔ Actor

Renderer ➔ Render Window ➔ Interactor
```

- In this course, we will use Paraview (instead of VTK directly) to create our visualisations.
Paraview

• Free data visualization tool built on VTK
• Useful for trying out different visualization techniques (without writing a single line of code)
• Links:
  - http://www.paraview.org/
The Paraview GUI
Simple example: Rendering a cube (again)
Rendering a cube

• In VTK

```python
import vtk

cube = vtk.vtkCubeSource()

cube_mapper = vtk.vtkPolyDataMapper()
cube_mapper.SetInputConnection(cube.GetOutputPort())
cube_actor = vtk.vtkActor()
cube_actor.SetMapper(cube_mapper)
cube_actor.GetProperty().SetColor(1.0, 0.0, 0.0)  # make the cube red

renderer = vtk.vtkRenderer()
renderer.SetBackground(0.8, 0.8, 0.8)  # make the background black
renderer.AddActor(cube_actor)

render_window = vtk.vtkRenderWindow()
render_window.SetWindowName("Simple VTK scene")
render_window.SetSize(400, 400)
render_window.AddRenderer(renderer)

interactor = vtk.vtkRenderWindowInteractor()
interactor.SetRenderWindow(render_window)

# Initialize the interactor and start the rendering loop
interactor.Initialize()
render_window.Render()
interactor.Start()```

• In Paraview

1. Go to the **Sources** menu
2. Select **Box**
3. Go to the **Properties** panel and change the color under **Coloring->Edit**
4. Done :)}
Demo 1: Cube
Demo 2: Glyphs
Demo 3: Transparency
Demo 5: Point cloud
Demo 6: Height map
Demo 7: Volume rendering (CT)
Demo 8: Animation
About the assignments

- Two assignments (in Paraview)
- No project this year!
- The lab sessions will be in PC-lab 1312 and 1313
- For the assignments, you may work individually or in pairs
- Paraview 5.4.1 is installed on the lab PCs
- We highly recommend that you also install the software on your own computer!
See you on the first lab!