Scientific Visualization

Spring 2009
Reference

The Visualization Toolkit : An Overview by William J. Schroeder, Kitware, Inc.
Agenda

- VTK Technology
  - Background
  - The Graphics Model
  - The Visualization Model
The Visualization Toolkit (VTK)

Overview
The Visualization Toolkit
An Object-Oriented Approach To 3D Graphics
Will Schroeder, Ken Martin, Bill Lorensen
ISBN 0-13-954694-4
Prentice Hall

Work on first edition began in 1994
What Is VTK?

A visualization toolkit

- Designed and implemented using object-oriented principles
- C++ class library (250,000 LOC, <100,000 executable lines)
- Automated Java, TCL, Python bindings
- Portable across Unix, Windows
- Supports 3D/2D graphics, visualization, image processing, volume rendering
VTK Is Not a System

- Embeddable
  - Plays with other software

- Separable
  - Can pull out “pieces”

- Adaptable
  - Not dependent on GUI
  - Not dependent on rendering library
VTK Architecture

Hybrid approach

- compiled C++ (faster algorithms)
- interpreted applications (rapid development) (Java, Tcl, Python)
- A toolkit

Interpreter
generated automatically

C++
core
Interpreters

- Tcl
- Java
- Python

*Interpreters provide faster turn-around; suffer from slower execution*
Example Code

http://www.vtk.org/example-code.php

Create a sphere
```cpp
#include "vtkSphereSource.h"
#include "vtkPolyDataMapper.h"
#include "vtkActor.h"
#include "vtkRenderWindow.h"
#include "vtkRenderer.h"
#include "vtkRenderWindowInteractor.h"

void main()
{
    // create sphere geometry
    vtkSphereSource *sphere = vtkSphereSource::New();
    sphere->SetRadius(1.0);
    sphere->SetThetaResolution(18);
    sphere->SetPhiResolution(18);

    // map to graphics library
    vtkPolyDataMapper *map = vtkPolyDataMapper::New();
    map->SetInput(sphere->GetOutput());

    // actor coordinates geometry, properties, transformation
    vtkActor *aSphere = vtkActor::New();
    aSphere->SetMapper(map);
    aSphere->GetProperty()->SetColor(0,0,1); // sphere color blue
}"
```
// a renderer and render window
vtkRenderer *ren1 = vtkRenderer::New();
vtkRenderWindow *renWin = vtkRenderWindow::New();
renWin->AddRenderer(ren1);

// an interactor
vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
iren->SetRenderWindow(renWin);

// add the actor to the scene
ren1->AddActor(aSphere);
ren1->SetBackground(1,1,1); // Background color white

// render an image (lights and cameras are created automatically)
renWin->Render();

// begin mouse interaction
iren->Start();
}
Graphics Model

Instances of render window (vtkRenderWindow)

Renderer instances (vtkRenderer)

Actor instances (vtkActor)
Graphics Model

- **RenderWindow** - contains final image
- **Renderer** - *draws into render window*
- **Actor** - *combines properties / geometry*
- **Lights** - illuminate actors
- **Camera** - renders scene
- **Mappers** - represent geometry
- **Transformations** - position actors
TCL Code

- package require vtk
- package require vtkinteration

- # create sphere geometry
  - vtkSphereSource sphere
  - sphere SetRadius 1.0
  - sphere SetThetaResolution 18
  - sphere SetPhiResolution 18

- # map to graphics library
  - vtkPolyDataMapper map;
  - map SetInput [sphere GetOutput]

- # actor coordinates geometry, properties, transformation
  - vtkActor aSphere
  - aSphere SetMapper map
  - [aSphere GetProperty] SetColor 0 0 1; # blue
TCL Code

# create a window to render into
vtkRenderWindow renWin
vtkRenderer ren1
renWin AddRenderer ren1

# create an interactor
vtkRenderWindowInteractor iren
iren SetRenderWindow renWin

# add the sphere
ren1 AddActor aSphere
ren1 SetBackground 1 1 1;# Background color white

# Render an image; since no lights/cameras specified, created automatically
renWin Render

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

<table>
<thead>
<tr>
<th>C++ Code</th>
<th>Python Code</th>
</tr>
</thead>
</table>
| ```cpp
// create sphere geometry
vtkSphereSource *sphere = vtkSphereSource::New();
sphere->SetRadius(1.0);
sphere->SetThetaResolution(18);
sphere->SetPhiResolution(18);
``` | ```python
# create sphere geometry
vtkSphereSource sphere
sphere.SetRadius(1.0)
sphere.SetThetaResolution(18)
sphere.SetPhiResolution(18)
``` |
| ```cpp
// map to graphics library
vtkPolyDataMapper *map = vtkPolyDataMapper::New();
map->SetInput(sphere->GetOutput());
``` | ```python
# map to graphics library
vtkPolyDataMapper map;
map.SetInput([sphere.GetOutput])
``` |
| ```cpp
// actor coordinates geometry, properties, transformation
vtkActor *aSphere = vtkActor::New();
aSphere->SetMapper(map);
aSphere->GetProperty()->SetColor(0,0,1); //blue
``` | ```python
# actor coordinates geometry, properties…
vtkActor aSphere
aSphere.SetMapper(map)
[aSphere.GetProperty].SetColor(0, 0, 1); # blue
``` |
Agenda

VTK Technology
- Background
- The Graphics Model
- The Visualization Model
What Is The Visualization Pipeline?

A sequence of process objects that operate on data objects to generate geometry that can be rendered by the graphics engine.
Visualization Model

- **Data Objects**
  - represent data
  - provide access to data
  - compute information particular to data
    (e.g., bounding box, derivatives)

- **Process Objects**
  - Ingest, transform, and output data objects
  - represent visualization algorithms
Data Objects / Data Sets

- **vtkDataObject** is a “blob” of data
  - Contains an instance of **vtkFieldData**

- **vtkDataSet** is data with geometric & topological structure; and with **attribute** data

<table>
<thead>
<tr>
<th>Geometry &amp; Topology</th>
<th>Points &amp; Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Set Attributes</td>
<td>Point Data</td>
</tr>
<tr>
<td></td>
<td>Cell Data</td>
</tr>
</tbody>
</table>
A dataset is a data object with structure.

Structure consists of:
- cells (e.g., polygons, lines, voxels)
- points (x-y-z coordinates)
- cells defined by connectivity list referring to points
- implicit representations
- explicit representations
Dataset Types

vtkPolyData

vtkStructuredPoints

vtkStructuredGrid

vtkRectilinearGrid

vtkUnstructuredGrid
Data Set Attributes

- **Scalars** - 1-4 values (vtkScalars)
  - single value ranging to RGBA color
- **Vectors** - 3-vector (vtkVectors)
- **Tensors** - 3x3 symmetric matrix (vtkTensors)
- **Normals** - unit vector (vtkNormals)
- **Texture Coordinates** - 1-3D (vtkTCoords)
- **Field Data** - (an array of arrays) (vtkFieldData)
Process Objects

Source

1 or more outputs

1 or more inputs

Filter

1 or more outputs

1 or more inputs

Mapper
Pipeline Execution Model

direction of data flow (via Execute())

Source \[\xrightarrow{\text{Data}}\] Filter \[\xrightarrow{\text{Data}}\] Mapper \[\xrightarrow{\text{Render()}}\]

direction of update (via Update())
Creating Pipeline Topology

aFilter->SetInput( bFilter->GetOutput());

The Role of Type-Checking

– SetInput() accepts dataset type or subclass
– C++ compile-time checking
– Interpreter run-time checking
Example Pipeline

- Decimation, smoothing, normals
- Implemented in C++

Note: data objects are not shown -> they are implied from the output type of the filter
vtkCyberReader *cyber = vtkCyberReader::New();
cyber->SetFileName("../../vtkdata/fran_cut");

tvtkDecimatePro *deci = vtkDecimatePro::New();
deci->SetInput( cyber->GetOutput() );
deci->SetTargetReduction( 0.9 );
deci->PreserveTopologyOn();
deci->SetMaximumError( 0.0002 );
Smoother & Graphics Objects

```cpp
vtkSmoothPolyDataFilter *smooth = vtkSmoothPolyDataFilter::New();
smooth->SetInput(deci->GetOutput());
smooth->SetNumberOfIterations( 20 );
smooth->SetRelaxationFactor( 0.05 );

vtkPolyDataNormals *normals = vtkPolyDataNormals::New();
normals->SetInput( smooth->GetOutput() );

vtkPolyDataMapper *cyberMapper = vtkPolyDataMapper::New();
cyberMapper->SetInput( normals->GetOutput() );

vtkActor *cyberActor = vtkActor::New();
cyberActor->SetMapper (cyberMapper);
cyberActor->GetProperty()->SetColor ( 1.0, 0.49, 0.25 );
cyberActor->GetProperty()->SetRepresentationToWireframe();
```
More Graphics Objects

```c++
vtkRenderer *ren1 = vtkRenderer::New();

vtkRenderWindow *renWin = vtkRenderWindow::New();
  renWin->AddRenderer( ren1 );

vtkRenderWindowInteractor *iren =
  vtkRenderWindowInteractor ::New();
  iren->SetRenderWindow( renWin );

ren1->AddActor( cyberActor );
ren1->SetBackground( 1, 1, 1 );
renWin->SetSize( 500, 500 );
iren->Start();
```
Results

Before
(52,260 triangles)

After Decimation
and Smoothing
(7,477 triangles)
Filter Overview: Sources

Readers
- vtkOBJReader
- vtkBYUReader
- vtkCyberReader
- vtkDataSetReader
- vtkMCubesReader
- vtkPLOT3DReader
- vtkPolyDataReader
- vtkRectilinearGridReader
- vtkSLCReader
- vtkSTLReader
- vtkStructuredGridReader
- vtkStructuredPointsReader
- vtkUnstructuredGridReader
- vtkVolume16Reader
- vtkFieldDataReader
- vtkBMPReader
- vtkPNMReader
- vtkTIFFReader
Sources

**Procedural Sources**

- vtkEarthSource
- vtkConeSource
- vtkCylinderSource
- vtkDiskSource
- vtkLineSource
- vtkOutlineSource
- vtkPlaneSource
- vtkPointSource
- vtkTextSource
- vtkVectorText
- vtkSphereSource
- vtkTexturedSphereSource
- vtkAxes
- vtkCursor3D
- vtkProgrammableSource
- vtkPointLoad
Filters

- vtkAppendFilter
- vtkAppendPolyData
- vtkBooleanTexture
- vtkBrownianPoints
- vtkCastToConcrete
- vtkCellCenters
- vtkCellDataToPointData
- vtkCullVisiblePoints
- vtkCleanPolyData
- vtkClipPolyData
- vtkClipVolume
- vtkConnectivityFilter
- vtkContourFilter
- vtkCutter
- vtkDashedStreamLine
- vtkDecimate
- vtkDecimatePro
- vtkDelaunay2D
- vtkDelaunay3D
- vtkDicers
Filters (2)

- vtkEdgePoints
- vtkElevationFilter
- vtkExtractEdges
- vtkExtractGeometry
- vtkExtractGrid
- vtkExtractTensorComponents
- vtkExtractUnstructuredGrid
- vtkExtractVOI
- vtkExtractVectorComponents
- vtkFeatureEdges
- vtkGaussianSplatter
- vtkGeometryFilter
- vtkGlyph3D
- vtkHedgeHog
- vtkHyperStreamline
- vtkIdFilter
- vtkLinearExtrusionFilter
- vtkMaskPolyData
- vtkOutlineFilter
- vtkPointDataToCellData
Filters (3)

- vtkProgrammableFilter
- vtkProjectedTexture
- vtkRecursiveDividingCubes
- vtkReverseSense
- vtkRibbonFilter
- vtkRotationalExtrusionFilter
- vtkShepardMethod
- vtkShrinkFilter
- vtkShrinkPolyData
- vtkSmoothPolyDataFilter

- vtkMaskPoints
- vtkMaskPolyData
- vtkMergeFilter
- vtkMergePoints
- vtkPolyDataNormals
- vtkProbeFilter
- vtkProgrammableAttributeDataFilter
- vtkSelectVisiblePoints
- vtkSpatialRepresentationFilter
- vtkStreamLine
Filters (4)

- vtkStreamPoints
- vtkStripper
- vtkStructuredGridGeometryFilter
- vtkStructuredGridOutlineFilter
- vtkStructuredPointsGeometryFilter
- vtkTensorGlyph
- vtkTextureMapToBox
- vtkTextureMapToCylinder
- vtkTextureMapToPlane
- vtkTextureMapToSphere
- vtkTexturedSphereSource
- vtkThreshold
- vtkThresholdPoints
- vtkThresholdTextureCoords
- vtkTransformFilter
- vtkTransformPolyDataFilter
- vtkTransformTextureCoords
- vtkTriangleFilter
- vtkTriangularTCoords
- vtkTriangularTexture
Filters (5)

- vtkTubeFilter
- vtkVectorDot
- vtkVectorNorm
- vtkVectorTopology
- vtkVoxelModeller
- vtkWarpScalar
- vtkWarpTo
- vtkWarpVector
Mappers

Writers
- vtkIVWriter
- vtkBYUWriter
- vtkSTLWriter
- vtkMCubesWriter
- vtkPolyDataWriter
- vtkRectilinearGridWriter
- vtkStructuredGridWriter
- vtkStructuredPointsWriter
- vtkUnstructuredGridWriter
- vtkFieldDataWriter
- vtkBMPWriter
- vtkPNMWriter
- vtkTIFFWriter

Graphics Mappers
- vtkPolyDataMapper
- vtkDataSetMapper
- (volume mappers - later)
- (image mappers - later)