VTK: The Visualization Toolkit

Part I: Overview and Graphics Models

Han-Wei Shen
The Ohio State University
What is VTK?

- An open source, freely available software system for 3D graphics, image processing, and visualization.
- Support for hundreds of algorithms in visualization and image processing
- Object-oriented design with different interpreted language wrappers.
At a Glance

• The core of VTK is written entirely in C++ and contains hundreds of classes
• VTK compiles and run on Windows, MacOS, Linux
• Different interfaces for fast prototyping: Tcl/Tk, Java, and Python
• Have users all over the world – The beauty of Open Source!
System Architecture

Interpreted Wrapper (Tcl, Java, Python)

- Tcl/Tk shell
- Java interpreter
- Python interpreter

• Tcl/Tk source
• Java JDK
• Python source

C++ core

- Libraries and includes (dll and .h files)
- Or (.a and .h files)

- All class source code (could take hours to compile)

Binary Installation: if you will use The classes to build your application
Source code Installation: If you want to extend vtk
VTK classes

(http://www.vtk.org/doc/nightly/html/classes.html)
VTK Object Models

- Graphics and Visualization Models
  - Graphics objects: rendering
  - Visualization objects: generating graphical objects to represent the data

Data Flow System: Pipeline execution
Pipeline Execution

Direction of data flow

Visualization model
Source → Filter → Mapper

Graphics model
Mapper → Actor → Renderer

Direction of ‘update’
VTK Mappers

• Mappers convert data into graphical primitives or write to a file (writer)
  – Mappers require one or more input data objects
  – Mappers terminate the visualization pipeline
• Example: vtkPolyDataMapper, which takes geometry such as cylinder or cone as input and convert it to renderable geometry

![Visualization model](image1)

![Graphics model](image2)
VTK Actors

- Actors represent graphical data or objects
- A VTK actor contains
  - object properties (color, shading type, etc)
  - geometry
  - transformations
- VTK actors need to work together with lights (vtkLight) and camera (vtkCamera) to make a scene
- The scene is then rendered to an image by a renderer (vtkRenderer)

Visualization model  Graphics model
VTK Renderer

- The Renderer in VTK, vtkRenderer, is to coordinate the rendering process that involves lights, cameras, and actors.
- vtkRenderer creates a default camera and lights if not present, but needs to have at least one actor.
- vtkRenderer needs to be connected with a vtkRenderWindow.

Visualization model

| Source | Filter | Mapper | Actor | Renderer |

Graphics model
VTK Render Window

• The class, vtkRenderWindow ties the entire rendering process together
• It manages all the platform dependent window management issues and hide the details from the user
• It also stores graphics specific information such as window size, position, title, frame buffer depth, etc.

Visualization model
Source -> Filter -> Mapper -> Actor -> Renderer

Graphics model

Render window
The Graphics Model

- Camera
- Actor
- Screen
- Light
- Viewer
The Graphics Model

camera
vtkCamera

Actor

Light

screen

Viewer
The Graphics Model

camera
vtkCamera

Actor
vtkActor
• vtkProperty
• vtkMapper
• vtkTransform

screen

Light

Viewer
The Graphics Model

camera
tvtkCamera

Actor
vtkActor
  • vtkProperty
  • vtkMapper
  • vtkTransform

Light
tvtkLight

screen

Viewer
The Graphics Model

camera
ttkCamera

Actor
• vtkActor
• vtkProperty
• vtkMapper
• vtkTransform

Light
ttkLight

vtkRenderer

screen

Viewer
The Graphics Model

The purpose is to render the geometry (volume) on the screen
Main() {
    create geometry;
    create a mapper;
    give the geometry to the mapper;
    create an actor;
    give the geometry to the mapper;
    create a renderer;
    give the actor to the renderer;
    create a window;
    give the renderer to the window;
    window->render();
}
VTK Simple Pseudo Code

Main() {

create geometry;
create a mapper;
give the geometry to the mapper;
create an actor;
give the mapper to the actor;
create a renderer;
give the actor to the renderer;
create a window;
give the renderer to the window;

window->render();
}

Geometry

create geometry
create a mapper
give the geometry to the mapper
create an actor
give the mapper to the actor
create a renderer
give the actor to the renderer
create a window
give the renderer to the window
Main() {
    create geometry;
    create a mapper;
    give the geometry to the mapper;
    create an actor;
    give the mapper to the actor;
    create a renderer;
    give the actor to the renderer;
    create a window;
    give the renderer to the window;
    window->render();
}
VTK Simple Pseudo Code

```plaintext
Main() {
    create geometry;
    create a mapper;
    give the geometry to the mapper;
    create an actor;
    give the mapper to the actor;
    create a renderer;
    give the actor to the renderer;
    create a window;
    give the renderer to the window;

    window->render();
}
```
VTK Simple Pseudo Code

```cpp
Main() {
    create geometry;
    create a mapper;
    give the geometry to the mapper;
    create an actor;
    give the mapper to the actor;
    create a renderer;
    give the actor to the renderer;
    create a window;
    give the renderer to the window;
    window->render();
}
```
VTK Simple Pseudo Code

Main() {
    create geometry;
    create a mapper;
    give the geometry to the mapper;
    create an actor;
    give the mapper to the actor;
    create a renderer;
    give the actor to the renderer;
    create a window;
    give the renderer to the window;
    window->render();
}

Geometry ➔ Mapper ➔ Actor ➔ Renderer ➔ Window
VTK Simple Pseudo Code

```cpp
Main() {
  create geometry;
  create a mapper;
  give the geometry to the mapper;
  create an actor;
  give the mapper to the actor;
  create a renderer;
  give the actor to the renderer;
  create a window;
  give the renderer to the window;
  window->render();
}
```
#include "vtk.h"

Main()
{
  vtkConeSource *cone = vtkConeSource::New();
  cone->SetResolution(8);

  vtkPolyDataMapper *mapper = vtkPolyDataMapper::New();
  mapper->SetInput(cone->GetOutput());

  vtkActor *coneActor = vtkActor::New();
  coneActor->SetMapper(Mapper);

  vtkRenderer *ren = vtkRenderer::New();
  ren->AddActor(coneActor);

  vtkRenderWindow *renWindow = vtkRenderWindow::New();
  renWindow->AddRenderer(ren);

  renWindow->render();
}
#include “vtk.h”

Main()
{
    vtkConeSource *cone = vtkConeSource::New();
    cone->SetResolution(8);

    vtkPolyDataMapper *mapper = vtkPolyDataMapper::New();
    mapper ->SetInput(cone->GetOutput());

    vtkActor *coneActor = vtkActor::New();
    coneActor->SetMapper(Mapper);

    vtkRenderer * ren = vtkRenderer::New();
    ren->AddActor(conActor);

    vtkRenderWindow *renWindow = vtkRenderWindow::New();
    renWindow->AddRenderer(ren);

    renWindow->render();
}
VTK Render a Cone

#include “vtk.h”

Main()
{
  vtkConeSource *cone = vtkConeSource::New();
  cone->SetResolution(8);

  vtkPolyDataMapper *mapper = vtkPolyDataMapper::New();
  mapper->SetInput(cone->GetOutput());

  vtkActor *coneActor = vtkActor::New();
  coneActor->SetMapper(Mapper);

  vtkRenderer *ren = vtkRenderer::New();
  ren->AddActor(conActor);

  vtkRenderWindow *renWindow = vtkRenderWindow::New();
  renWindow->AddRenderer(ren);

  renWindow->render();
}
VTK Render a Cone

```
#include "vtk.h"

Main()
{
  vtkConeSource *cone = vtkConeSource::New();
  cone->SetResolution(8);

  vtkPolyDataMapper *mapper = vtkPolyDataMapper::New();
  mapper->SetInput(cone->GetOutput());

  vtkActor *coneActor = vtkActor::New();
  coneActor->SetMapper(Mapper);

  vtkRenderer *ren = vtkRenderer::New();
  ren->AddActor(conActor);

  vtkRenderWindow *renWindow = vtkRenderWindow::New();
  renWindow->AddRenderer(ren);

  renWindow->render();
}
```
VTK Render a Cone

#include “vtk.h”

Main()
{
  vtkConeSource *cone = vtkConeSource::New();
  cone->SetResolution(8);

  vtkPolyDataMapper *mapper = vtkPolyDataMapper::New();
  mapper->SetInput(cone->GetOutput());

  vtkActor *coneActor = vtkActor::New();
  coneActor->SetMapper(Mapper);

  vtkRenderer *ren = vtkRenderer::New();
  ren->AddActor(conActor);

  vtkRenderWindow *renWindow = vtkRenderWindow::New();
  renWindow->AddRenderer(ren);

  renWindow->render();
}
vtkRenderWindow Output

1 vtkRenderWindow

2 vtkRenderer

vtkCamera

vtkLight

vtkActor

( property, geometry(mapper), transformation, etc)