

# 2D1257 — Visualization

## 2. VTK Introduction

March 20, 2007

### 1 Introduction

We are getting familiar with various VTK methods and algorithms in this lab. After having completed this lab you should be able to:

- Construct simple VTK applications
- Manipulate VTK objects and annotate visualizations
- Use the visualization pipeline with filters
- Read in data from files

VTK references:

- <http://vtk.org/doc/release/5.0/html/>
- <http://www.vtk.org/>
- The Visualisation Toolkit, 4th Edition
- The VTK User's Guide (optional)

### 2 Getting started

Download lab2.zip from the course page and use `unzip lab2.zip` to extract the files on Solaris.

VTK is installed in `/pkg/vtk/5.0.2/os/` and we have provided you with a Makefile that includes the header and library files. The framework for the later part of this lab can be fetched from the course webpage. These files are:

**Makefile** simplifies the compilation. Make sure you understand how it works.

**Cone.cpp** is a minimal example. Read the file and understand what is done to produce the rendered image. What are the different components in the pipeline? What is the superclass of `vtkConeSource` and what other subclasses does the superclass have?

Note that the static method `::New()` and the member method `->Delete()` are used instead of operators `new` and `delete`. `Delete()` keeps track of the number of references to an object and deletes it when no other references exist. Make sure that you always add a `->Delete()` after you have instantiated an object with `::New()`.

**interaction1.cpp** demonstrates mouse/keyboard interaction and callbacks

## 2.1 Compile and run

Unless you have the Sun Studio module active:

```
prompt% module add sunstudio
```

To compile:

```
prompt% make Cone
```

An executable file is produced. To run this program:

```
prompt% ./Cone
```

## 3 Interaction

A static image is produced in the example above. To better make use of the visualization you will add interactivity to the scene.

`vtkRenderWindowInteractor` provides means for interaction in VTK. Read about it in the manual pages. What functionality besides window interaction does it provide?

Create an instance of the class and associate it with `vtkRenderWindow` by using `SetRenderWindow(...)`. When you have defined the whole pipeline, you start the interaction by calling `Start()`. Observe that the visualisation pipeline is executed backwards; it is the `vtkRenderWindowInteractor` that sends an update call through the pipeline. When you have implemented an interactor you can remove the lines of code that are waiting for a keystroke.

### Task 1 — Add interaction:

Add interaction to `Cone.cpp`. Read the manual pages about `vtkRenderWindowInteractor` and `vtkInteractorStyle`. Which is the default interactor style? Change the interactor style such that it behaves like a trackball on startup. What other interactor styles exist in VTK? Try out the different mouse and keystrokes that `vtkRenderWindowInteractor` supports. Get yourself familiar with the structure of the manual pages.

### Task 2:

Read the manual pages about `vtkConeSource` and modify the code such that you can interactively change the width and height of the cone.

## 4 Work with an explicit camera

A default camera and light source are created when the `vtkRenderer` object is instantiated. You can also create your own cameras and light sources explicitly and associate them with the renderer:

```

...
vtkCamera *camera1 = vtkCamera::New();
camera1->SetPosition(0.0,0.0,30.0);
camera1->SetFocalPoint(1,1,0);
camera1->ComputeViewPlaneNormal();
camera1->SetViewUp(0,1,0);
ren->SetActiveCamera(camera1);
...

```

Try to use your own camera in your code. An alternative is to use a pointer to the renderer's active camera to update its properties.

```
ren->GetActiveCamera()->Zoom(1.8);
```

## 5 Light sources

Lightsources in VTK are objects of the type `vtkLight`, look it up in the manual pages. What do the methods `SetPosition(...)`, `SetFocalPoint(...)`, `SetColor(...)` do? What is the difference between a `HeadLight`, `CameraLight` and `SceneLight`?

### Task 3:

Place a blue light source at (1, 0, 0) and a red light source that follows the camera motion. How is a light created and what are its default values? Control how the light moves using the methods in `vtkRenderer` and `vtkRenderWindowInteractor`.

## 6 Object properties

`vtkActor` inherits methods for rotation, translation and scaling, as well as for modifying color, material features, etc, from `vtkProp3D`. Change the material features of the cone. What other subclasses do `vtkProp3D` have and what do they do?

## 7 Interact with the object and update the pipeline

Use the `interaction1.cpp` from the lab directory for this part of the lab. Make changes to the Makefile such that you can compile like this:

```
prompt% make interaction1
```

There is code in `interaction1.cpp` for changing the properties of the cone and updating the pipeline. Use the `i` key to switch interaction mode so that you can change the height and width of the cone by moving the mouse. Use the `i` key to switch back to camera control.

To interact with the pipeline this way it can be convenient to use `vtkInteractorStyleUser` and use callback functions. The pipeline is executed again when a feature is changed. To modify the objects we have added global pointers to them. Read the callback function and understand how it is used in the program. How is the interaction changed when you press `i`?

**Task 4:**

Change the code so that the resolution of the cone is changed as you move the mouse.

## 8 Read in a file in the standard format

VTK has support for a number of different file formats. Look for the classes that end with “Reader” and “Importer” in the Class Hierarchy list in the manual pages. You will load a 3D model from a stereo-lithography file (`42400-IDGH.stl`), which can be specified with the method `SetFileName(...)`.

**Task 5:**

Copy `Cone.cpp` to a new file `Read.cpp` and remove the light sources and the camera. Replace the cone with the data in the file `42400-IDGH.stl`. Make changes in the `Makefile` to compile `Read.cpp`:

```
prompt% make Read
```

## 9 Filtering data

One of the advantages of VTK is the possibility to filter data. As an introduction we use `vtkShrinkPolyData` that has the method `SetShrinkFactor(...)`.

**Task 6:**

Shrink all polygons in `42400-IDGH.stl` with a factor of 0.8.

**Task 7:**

Change the program such that the callback function can modify the factor interactively.

## 10 Level of Detail (LOD)

VTK supports a simple form of LOD. You simply use this feature by interchanging `vtkActor` with `vtkLODActor`. Use `SetDesiredUpdateRate(...)` in the renderer to specify the desired performance. The interactor object will modify this frequency as you interact with the object. The interactor can be assigned two different frequency values. One is for idle situations and the other for interaction with the object.

**Task 8:**

Use LOD on the model that you read in the previous example. Use 4-5000 points and experiment with the desired frame rate. [NOTE: if the computer is fast you will perhaps not notice that much of a difference]

`vtkLODProp3D` provides even better control of LOD. An advantage is that different rendering techniques can be used. You can mix the techniques of rendering dots, polygons and volume to get true interactive frame rates.

**Task 9:**

Study the manual pages for `vtkLODProp3D` and explain how it works, how you choose LOD level and how different LOD objects are created and assigned.

## 11 Compositions

VTK provides the possibility to group actors in assemblies with the help of `vtkAssembly`.

**Task 10:**

Study the manual pages for `vtkAssembly` and `vtkPropAssembly`. What are the main differences between the two?

## 12 Annotation

A `vtkFollower` is an actor that always faces the camera. This is useful for the implementation of text labels or “billboards” (e.g., trees in video games were commonly implemented as simple one-sided textures that rotate such that they always face the camera, which greatly reduces the necessary geometry). The active camera can be obtained from `vtkRenderer`.

**Task 11:**

Create a plain C++ class (there is no need to inherit from any VTK classes) `AnnotatedObject` with a constructor that takes a `vtkPolyDataAlgorithm` object, a text string and a `vtkRenderer`. In the constructor, create the pipeline for the `vtkPolyDataAlgorithm` object, make the object transparent and add the resulting `vtkActor` to the `vtkRenderer`. Use `vtkTextSource` to create a `vtkFollower` from the text string and add it to the `vtkRenderer` as well. Add a `SetPosition(...)` method that changes the position for the `vtkActor` and the `vtkFollower`. Make sure that you clean up all allocated memory after use. Use this class to draw a cube with length=100 at (300, 0, 0), a cone with dimensions=100x30 at (-300, 0, 0) and a sphere with radius=100 at (-300, -150, 0).