# Lecture 4: CS6250 Graphics & Visualization

- Visualization Pipeline

# Transformation Matrices

- Origin
- Position
- Orientation
- Scale

An object is first translated to its origin, scaling and rotations occur about this point. Rotations occur in the order y, x, and then z axis. Finally, the object is translated back to the correct position (combination of Origin and Position translations).

Use the `RotateX`, `RotateY` and `RotateZ` methods, rather than the `SetOrientation` Method when possible.

# Stupid Cow Tricks

15 degrees

30 degrees

60 degrees

# More Cows

90 degrees

120 degrees

# The Visualization Pipeline

Visualization involves the transformation of data. This process is modeled in VTK as a pipeline. Data flows into the pipeline is transformed, and then mapped to a display.

Each transformation of the data must be represented somehow in the computer. Eventually we want to represent the data as a collection of images on the computer.

Visualization transforms a computational or numerical form of data into a graphical form.

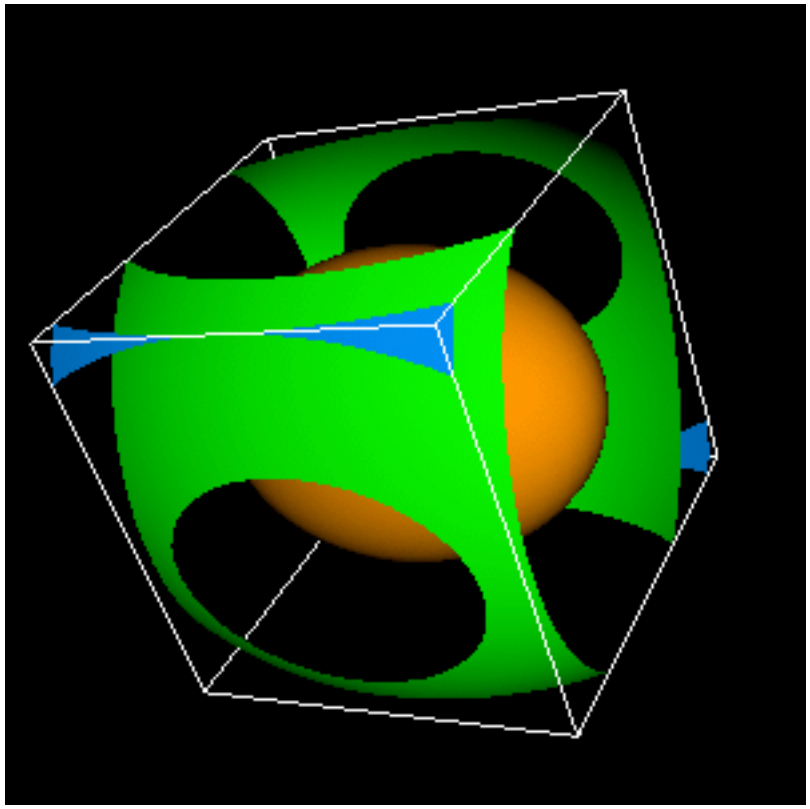
 

# An Example – a quadric equation

$$F(x, y, z) = a_0x^2 + a_1y^2 + a_2z^2 + a_3xy + a_4yz + a_5xz + a_6x + a_7y + a_8z + a_9$$

This is a function over three dimensions. What does it mean? What does it look like?

# Quadric Surface Plot



# How else can we visualize this function?

# The Functional Model

# Sources, Sinks, Filters

Sources are

Sinks are

Filters are

# Types of Objects in the Model

Data Objects

These Objects store or represent information. Usually they provide methods to create, access, and delete the information.

Information hiding is one concept they support.

One can obtain characteristics about the data present as well, such as aggregate information.

# Process Objects

These objects operate on input data to generate output data.

There are three kinds of process objects:

- Source objects

  Procedural objects – from local parameters (e.g. Cone)

  From external data sources – readers, importers

- Filter objects

  Shrink filter

- Mapper objects

  Terminate the visualization pipeline.

  May send data to a file (writer).

# Maintaining Compatible Types

One way to design a filtering system is to have all objects be of one type (dataset).

Are there any problems with this design?

What other approaches are there?

What difficulties occur in these systems?

# Multiplicity of input and output

What is the difference between multiple fan-out and multiple output?

Multiple fan-out is:

Multiple output is:

# Loops

Why would we want loops in our visualization networks?

# Executing the network

When should the network be activated and executed?

How can we keep from performing extra computation?

# Synchronization?

Explicit execution vs. implicit execution

What are the benefits/problems with each approach?

Explicit:

# Implicit Execution

Advantages/disadvantages?

# Conditional Execution

Change what we do based on partial results.

Yields greater flexibility.

# Data Interfaces

Given that you have some data, and you want to put it through a visualization pipeline, what choices do you have about how to do this?

- Programming interface

  Write a c++ program.

- File interface

  Suck in data in a form VTK supports, and write it back out in another format VTK supports.

- System interface

  Make use of other systems to help. E.G. raytracer, VRML models, etc.

# Where does VTK fit into all this?

# Example

```
int main( int argc, char *argv[] )
{
  // create a rendering window and renderer
  vtkRenderer *ren = vtkRenderer::New();
  vtkRenderWindow *renWindow = vtkRenderWindow::New();
    renWindow->AddRenderer(ren);
  vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
    iren->SetRenderWindow(renWindow);
  renWindow->SetSize( 300, 300 );

  // create an actor and give it cone geometry
  vtkConeSource *cone = vtkConeSource::New();
    cone->SetResolution(8);

  vtkShrinkPolyData *shrink = vtkShrinkPolyData::New();
     shrink->SetInput(cone->GetOutput());
     shrink->SetShrinkFactor(0.7);

  vtkPolyDataMapper *coneMapper = vtkPolyDataMapper::New();
    coneMapper->SetInput(shrink->GetOutput());
  vtkActor *coneActor = vtkActor::New();
    coneActor->SetMapper(coneMapper);
```

```
  // assign our actor to the renderer
  ren->AddActor(coneActor);
  // draw the resulting scene
  renWindow->Render();

  //  Begin mouse interaction
  iren->Start();

  // Clean up
  ren->Delete();
  renWindow->Delete();
  iren->Delete();
  cone->Delete();
  shrink->Delete();
  coneMapper->Delete();
  coneActor->Delete();
}
```

# Output