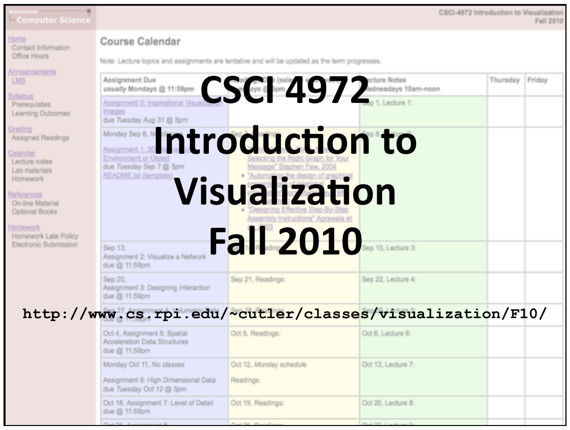
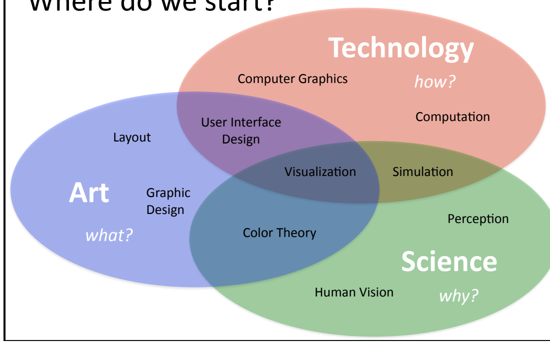
CSCI 4972 Introduction to Visualization Fall 2010

http://www.cs.rpi.edu/~cutler/classes/visualization/F10/

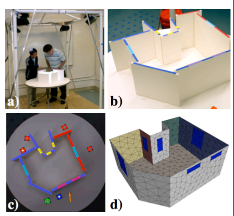


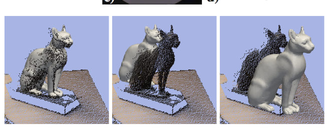
"Introduction" to Visualization: Where do we start?



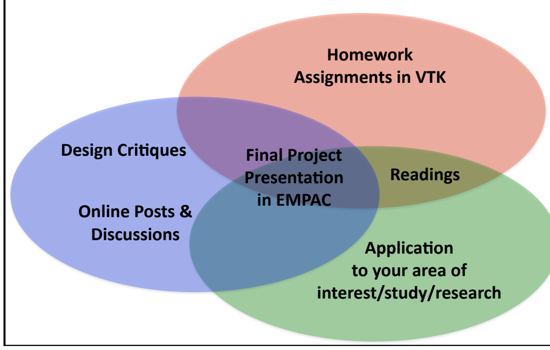
Introductions of People

- Barb Cutler
  - Faculty in Computer Science
  - Computer Graphics & Computational Geometry & Architectural Design Tools
- David Doria
  - PhD student in Electrical, Computer, & Systems Engineering
  - Research in Computer Vision & Point Cloud Data
  - VTK contributer
- And you?
  - Major/Research Area
  - Skills & Strengths





Course Structure



Readings (20%)

- Several readings (typically academic research papers in visualization) assigned each week
- Select one paper and read it carefully
  - If the paper is too advanced, pick one concept from the paper and use Google/Wikipedia/etc. to read more about that concept
  - Make a post about the paper on LMS focusing on what you learned by Tuesday @5pm
  - Join the online discussion of the papers

Participation & Presentations (15%)

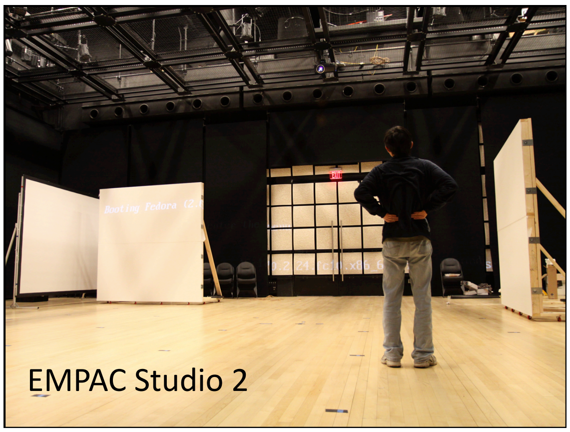
- Contribute to in-class & online discussions ☺
- Each student will do a 3 minute presentation of one of the readings during the semester
  - Focus on the contributions of the paper
  - Summarize the online discussion

Homework Assignments (35%)

- Weekly programming assignments
  - In C++ using the VTK libraries
  - We'll start working on the homework during the last half of each Wednesday class period (bring your laptop to class)
  - Generally due on Mondays @ 11pm
  - Individual & team-based assignments
- Creativity & thoughtful design encouraged and will be rewarded

Final Project (20%)

- Topic of your choice
- Team projects *highly* encouraged
- Last 5 weeks of the semester
- Presentation/Demonstration in EMPAC Studio 2



EMPAC Studio 2

Today's Class

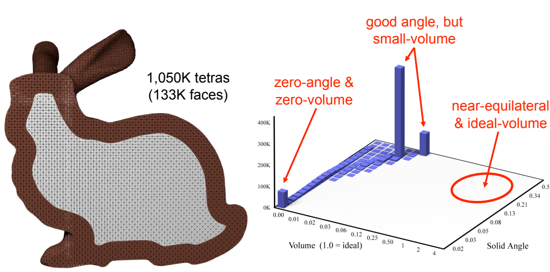
- Webpage, Syllabus, Course Structure, etc.
- Learning Outcomes
- Highlights from Assignment #0
- Readings for Next Week

- VTK Overview
- Using Transformations

Learning Outcomes

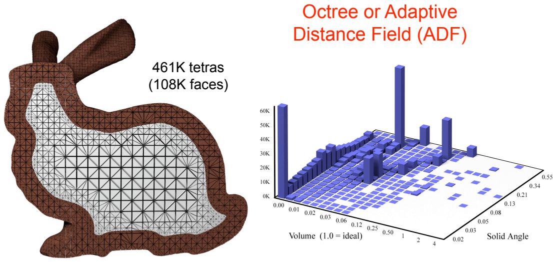
- Analyze, interpret, and evaluate a specific visualization example and discuss how the visualization might be improved for more accurate interpretation or communication of patterns in the data.
- Select or design an effective visualization strategy for a variety of different types of data.
- Create a visualization of a new dataset using available open-source visualization resources.
- Use visualization to communicate results of experiments and research in their field of study.
- Incorporate visualization for debugging and improved program development or experimental data analysis in their field of study.

Visualization of Tetrahedra Quality



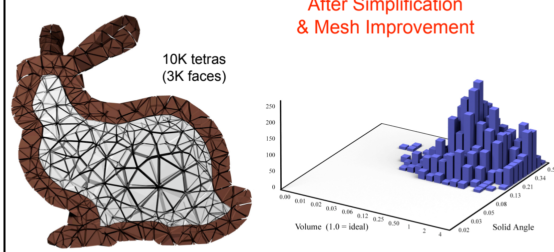
Visualization of Tetrahedra Quality

Octree or Adaptive
Distance Field (ADF)

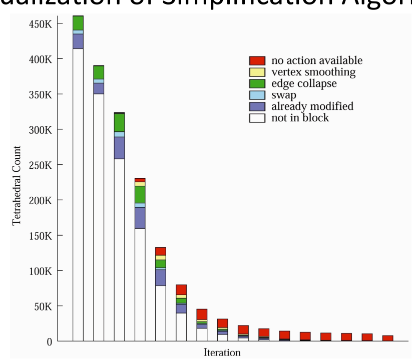


Visualization of Tetrahedra Quality

After Simplification
& Mesh Improvement



Visualization of Simplification Algorithm



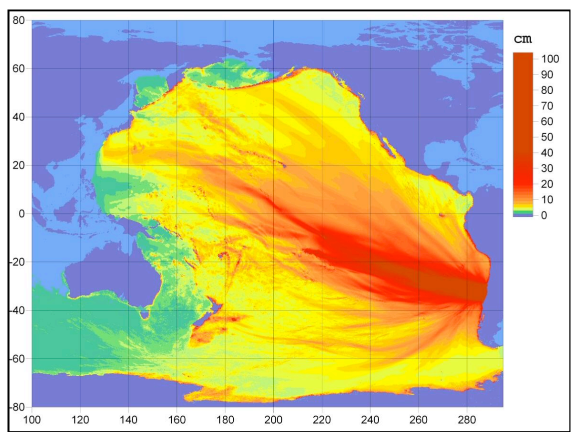
Today's Class

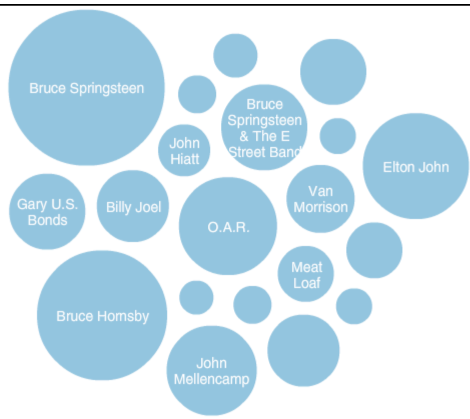
- Webpage, Syllabus, Course Structure, etc.
- Learning Outcomes
- Highlights from Assignment #0
- Readings for Next Week

- VTK Overview
- Using Transformations

Visualization Design Principles

- Scientific Visualization vs. Information Visualization
- Simple clean design vs. "Chart Junk"
- Managing & leveraging huge amounts of data
- Understanding your Audience
  - E.g., Visualization for Science, Communication, Education, Debugging, etc.
- Importance of companion text (title, axis labels, legend, caption)
- Targeting visualization design to human perception & low-level vision processing





Visualization Design Principles

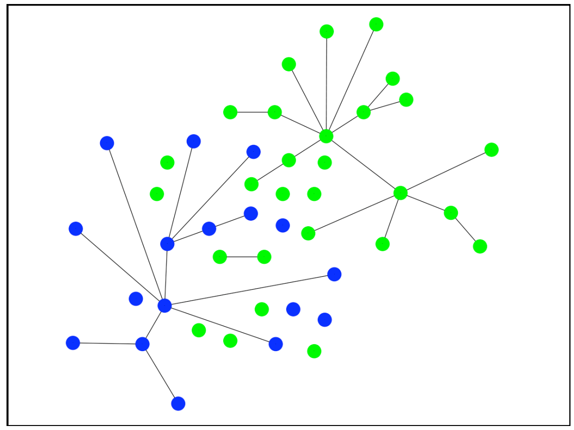
- Scientific Visualization vs. Information Visualization
- Simple clean design vs. "Chart Junk"
- Managing & leveraging huge amounts of data
- Understanding your Audience
  - E.g., Visualization for Science, Communication, Education, Debugging, etc.
- Importance of companion text (title, axis labels, legend, caption)
- Targeting visualization design to human perception & low-level vision processing

Visualization Design Principles

- Scientific Visualization vs. Information Visualization
- Simple clean design vs. "Chart Junk"
- Managing & leveraging huge amounts of data
- Understanding your Audience
  - E.g., Visualization for Science, Communication, Education, Debugging, etc.
- Importance of companion text (title, axis labels, legend, caption)
- Targeting visualization design to human perception & low-level vision processing

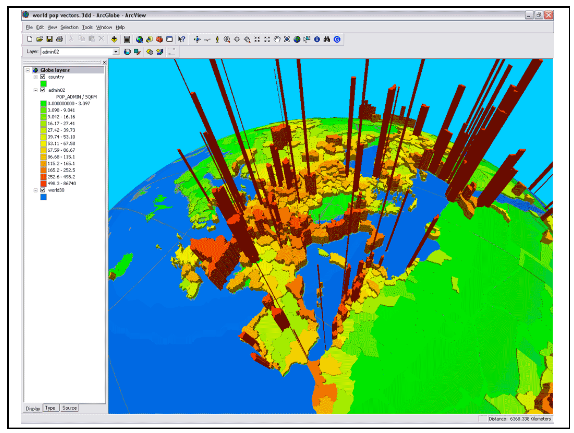
Visualization Design Principles

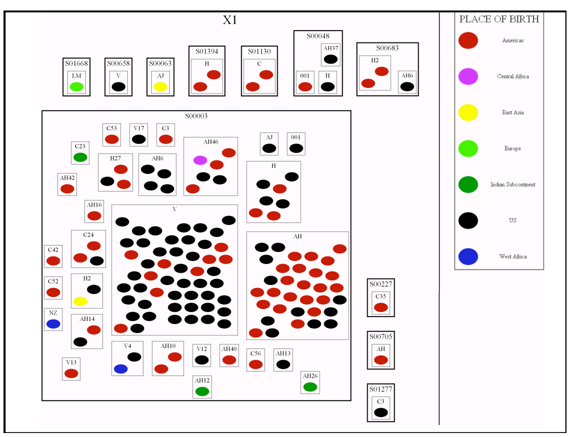
- Scientific Visualization vs. Information Visualization
- Simple clean design vs. "Chart Junk"
- Managing & leveraging huge amounts of data
- Understanding your Audience
  - E.g., Visualization for Science, Communication, Education, Debugging, etc.
- Importance of companion text (title, axis labels, legend, caption)
- Targeting visualization design to human perception & low-level vision processing



Visualization Design Principles

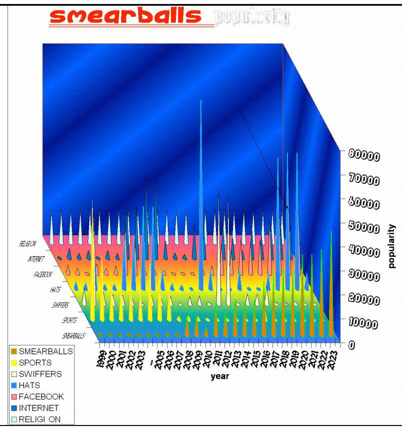
- Scientific Visualization vs. Information Visualization
- Simple clean design vs. "Chart Junk"
- Managing & leveraging huge amounts of data
- Understanding your Audience
  - E.g., Visualization for Science, Communication, Education, Debugging, etc.
- Importance of Companion text (title, axis labels, legend, caption)
- Targeting visualization design to human perception & low-level vision processing





Visualization Design Principles

- Scientific Visualization vs. Information Visualization
- Simple clean design vs. "Chart Junk"
- Managing & leveraging huge amounts of data
- Understanding your Audience
  - E.g., Visualization for Science, Communication, Education, Debugging, etc.
- Importance of companion text (title, axis labels, legend, caption)
- Targeting visualization design to human perception & low-level vision processing



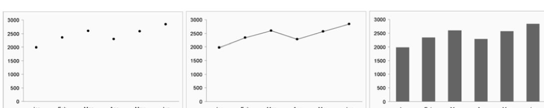
Today's Class

- Webpage, Syllabus, Course Structure, etc.
- Learning Outcomes
- Highlights from Assignment #0
- Readings for Next Week

- VTK Overview
- Using Transformations

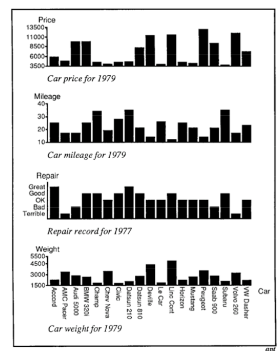
Readings for Next Week:

- "Eenie, Meenie, Minie, Moe: Selecting the Right Graph for Your Message" Stephen Few, 2004



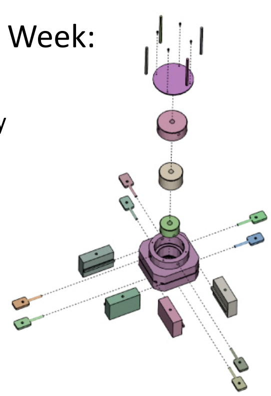
Readings for Next Week:

- "Automating the design of graphical presentations of relational information" Jock Mackinlay, 1986



Readings for Next Week:

- "Designing Effective Step-By-Step Assembly Instructions"
  Agrawala et al., 2003



Today's Class

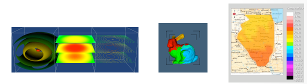
- Webpage, Syllabus, Course Structure, etc.
- Learning Outcomes
- Highlights from Assignment #0
- Readings for Next Week

- VTK Overview
- Using Transformations

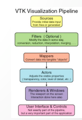
Introduction To VTK: Overview

Outline

- What is VTK?
- What can it do?

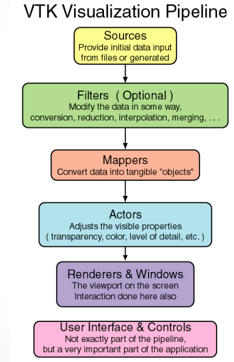


- Typical workflow – (the "pipeline")



- Visualization ToolKit
  - Scientific visualization
  - Information visualization
  - Written in c++
  - C++, Java, Python, Tcl API's
- Much more than that:
  - Excellent framework for scientific programming

Basic Terminology

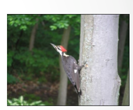


Taken from: http://www.cs.uic.edu/~jbell/CS526/Tutorial/Tutorial.html

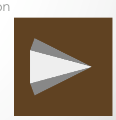
Sources

- The source of data flowing through the visualization pipeline

- Two types (not distinguished)
  - Readers: read data out of files in a wide variety of formats (images, 3D data, just about anything you can think of)

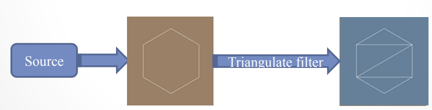


  - Generative sources: generate data based on input parameters. (E.g. a cone source, which generates information describing a cone, given its radius and height.)



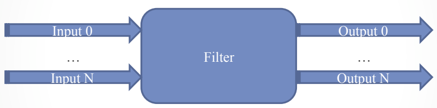
Filters

- Components that receive data from other components
- Modify the data in some way
- Deliver the modified data as output to be used by other components



Filter Connections

- Can have multiple input and output connections



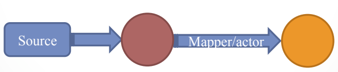
- Set/get outputs with:
  - filter->SetInputConnection(someFilter->GetOutputPort());
- If the input is an object, not a filter, use:
  - filter->SetInputConnection(someObject->GetProducerPort());

Mappers

- VTK conceptually divides the pipeline into two segments
  - Data processing – consists of sources and filters
  - Rendering – consists of actors, renderers, and windows (OpenGL and OS style things)
- Mappers serve as the transition between the two segments

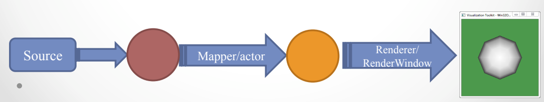
Actors

- Components that allow for the adjustment and control of the appearance properties (color, thickness, etc) of objects rendered onto the screen



Renderer and RenderWindow

- The end of the pipeline – display your work on the screen
- Renderer
  - World to Screen coordinate conversions
  - Lighting
- RenderWindow
  - "Contains" a renderer
  - Set the window size/title/etc



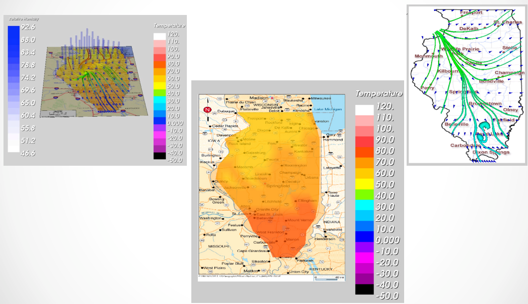
RenderWindowInteractor

- Determine how user actions affect the scene
- Right now, you just need to call Start() to display the RenderWindow
- More on this in a later class…

Updating the Pipeline

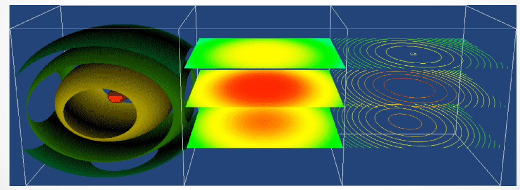
- VTK components do not normally generate their output until requested to do so
- To update the pipeline, call Update() on the object that you want to update!
- This will cause the pipeline to issue an Update() request to all if its inputs, which will in turn issue Update() requests to all of their inputs, and so on

Examples: Weather

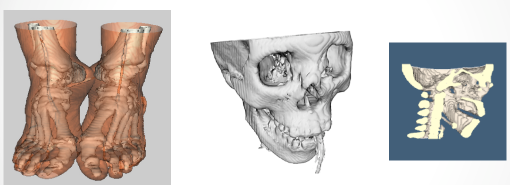


Examples: Mathematics

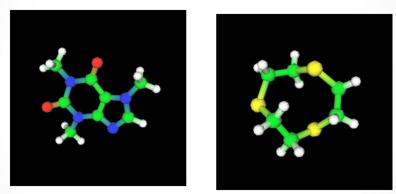
- $F(x,y,z) = a_0x^2 + a_1y^2 + a_2z^2 + a_3xy + a_4yz + a_5xz + a_6x + a_7y + a_8z + a_9$



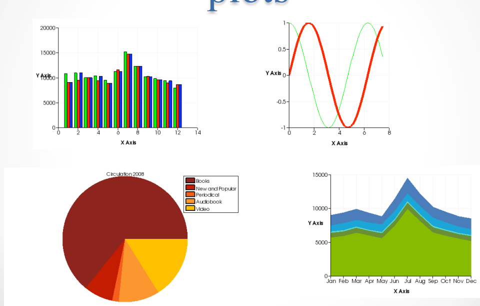
Examples: Medical Data



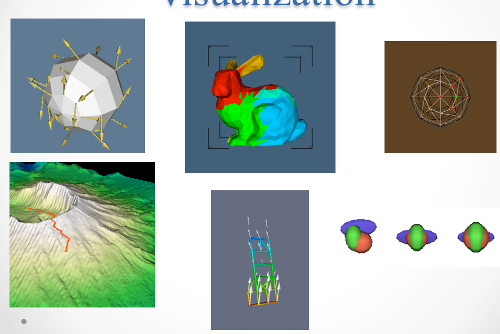
Examples: Chemistry



Examples: "Excel style" plots

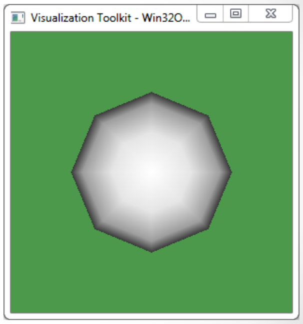


Examples: Generic data visualization



What does the code look like?

```
#include <vtkSphereSource.h>
#include <vtkPolyData.h>
#include <vtkSmartPointer.h>
#include <vtkPolyDataMapper.h>
#include <vtkActor.h>
#include <vtkRenderWindow.h>
#include <vtkRenderer.h>
#include <vtkRenderWindowInteractor.h>

int main(int , char *[])
{

  //Create a sphere
  vtkSmartPointer<vtkSphereSource> sphereSource =
    vtkSmartPointer<vtkSphereSource>::New();
  sphereSource->SetCenter(0.0, 0.0, 0.0);
  sphereSource->SetRadius(5.0);

  //Create a mapper and actor
  vtkSmartPointer<vtkPolyDataMapper> mapper =
    vtkSmartPointer<vtkPolyDataMapper>::New();
  mapper->SetInputConnection(sphereSource->GetOutputPort());

  vtkSmartPointer<vtkActor> actor =
    vtkSmartPointer<vtkActor>::New();
  actor->SetMapper(mapper);

  //Create a renderer, render window, and interactor
  vtkSmartPointer<vtkRenderer> renderer =
    vtkSmartPointer<vtkRenderer>::New();
  vtkSmartPointer<vtkRenderWindow> renderWindow =
    vtkSmartPointer<vtkRenderWindow>::New();
  renderWindow->AddRenderer(renderer);
  vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor =
    vtkSmartPointer<vtkRenderWindowInteractor>::New();
  renderWindowInteractor->SetRenderWindow(renderWindow);

  //Add the actor to the scene
  renderer->AddActor(actor);
  renderer->SetBackground(.3, .6, .3); // Background color green

  //Render and interact
  renderWindow->Render();
  renderWindowInteractor->Start();

  return EXIT_SUCCESS;
}
```



Code Breakdown: Includes

Get used to seeing long lists of includes. This is about the minimal set.

```
#include <vtkSphereSource.h>
#include <vtkPolyData.h>
#include <vtkSmartPointer.h>
#include <vtkPolyDataMapper.h>
#include <vtkActor.h>
#include <vtkRenderWindow.h>
#include <vtkRenderer.h>
#include <vtkRenderWindowInteractor.h>
```

Code Breakdown: Objects

```
//Create a sphere
vtkSphereSource* sphereSource =
  vtkSphereSource::New();
  sphereSource->SetCenter(0.0, 0.0, 0.0);
  sphereSource->SetRadius(5.0);
  sphereSource->Update();
```

```
//Equivalent using smart pointers
vtkSmartPointer<vtkSphereSource> sphereSource =
  vtkSmartPointer<vtkSphereSource>::New();
sphereSource->SetCenter(0.0, 0.0, 0.0);
sphereSource->SetRadius(5.0);
sphereSource->Update();
```

Code Breakdown: Rendering "Stuff"

(Luckily, this is usually a "copy/paste" with a few name replacements!)

```
//Create a mapper and actor
vtkSmartPointer<vtkPolyDataMapper> mapper =
  vtkSmartPointer<vtkPolyDataMapper>::New();
mapper->SetInputConnection(sphereSource->GetOutputPort());

vtkSmartPointer<vtkActor> actor =
  vtkSmartPointer<vtkActor>::New();
actor->SetMapper(mapper);

//Create a renderer, render window, and interactor
vtkSmartPointer<vtkRenderer> renderer =
  vtkSmartPointer<vtkRenderer>::New();
vtkSmartPointer<vtkRenderWindow> renderWindow =
  vtkSmartPointer<vtkRenderWindow>::New();
renderWindow->AddRenderer(renderer);
vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor =
  vtkSmartPointer<vtkRenderWindowInteractor>::New();
renderWindowInteractor->SetRenderWindow(renderWindow);

//Add the actor to the scene
renderer->AddActor(actor);
renderer->SetBackground(.3, .6, .3); // Background color green

//Render and interact
renderWindow->Render();
renderWindowInteractor->Start();
```

Today's Class

- Webpage, Syllabus, Course Structure, etc.
- Learning Outcomes
- Highlights from Assignment #0
- Readings for Next Week

- VTK Overview
- Using Transformations

Introduction to VTK: Transforms

Types of Transforms

- Basic transforms include:
  - "Rigid" transforms
    - Translation
    - Rotation
  - "Similarity" transformations
    - Scale

Translation

Rotation

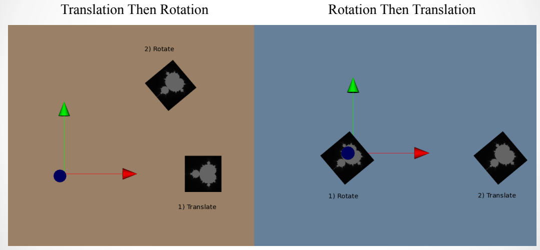
Scale

Order is Important!

- A translation followed by a rotation typically does not produce the same result as a rotation followed by a translation
- This is because the rotation is performed on the whole space around the world origin
- transform->PostMultiply() makes operations behave in the order that you specify them

Order is Important Demo

Translation Then Rotation | Rotation Then Translation



http://www.vtk.org/Wiki/VTK/Examples/Cxx/PolyData/TransformOrder

Transformations in VTK

- Can apply transformation either to the data itself (before Mapper) or to an Actor
- Why transform data?
  - You have access to the numerical result
- Why transform actor?
  - Faster
  - Can have multiple instances of the same data
- In either case, there is no reason to ever look at a transformation matrix

Transforming Data

- Construct a vtkTransform and then apply it with vtkTransformPolyDataFilter

```
vtkSmartPointer<vtkTransform> transformation=
  vtkSmartPointer<vtkTransform>::New();
transformation >Translate(double x, double y, double z);
transformation ->RotateX(double angle);
transformation ->Scale(double x, double y, double z);

vtkSmartPointer<vtkTransformPolyDataFilter> transformFilter =
  vtkSmartPointer<vtkTransformPolyDataFilter>::New();
transformFilter->SetInputConnection(sphereSource->GetOutputPort());
transformFilter->SetTransform(translation);
transformFilter->Update();
```

Transforming Actor

- Construct a vtkTransform and then apply it directly to an actor

```
vtkSmartPointer<vtkTransform> transform =
  vtkSmartPointer<vtkTransform>::New();
transform->RotateZ(90.0);

actor->SetUserTransform(transform);
```