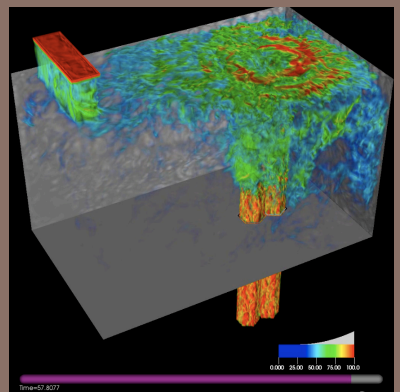
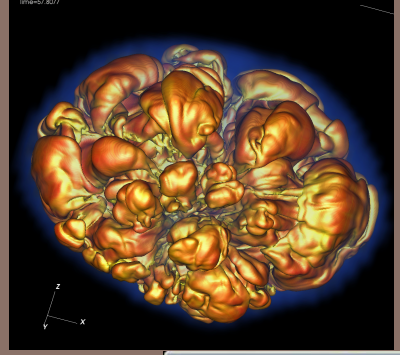
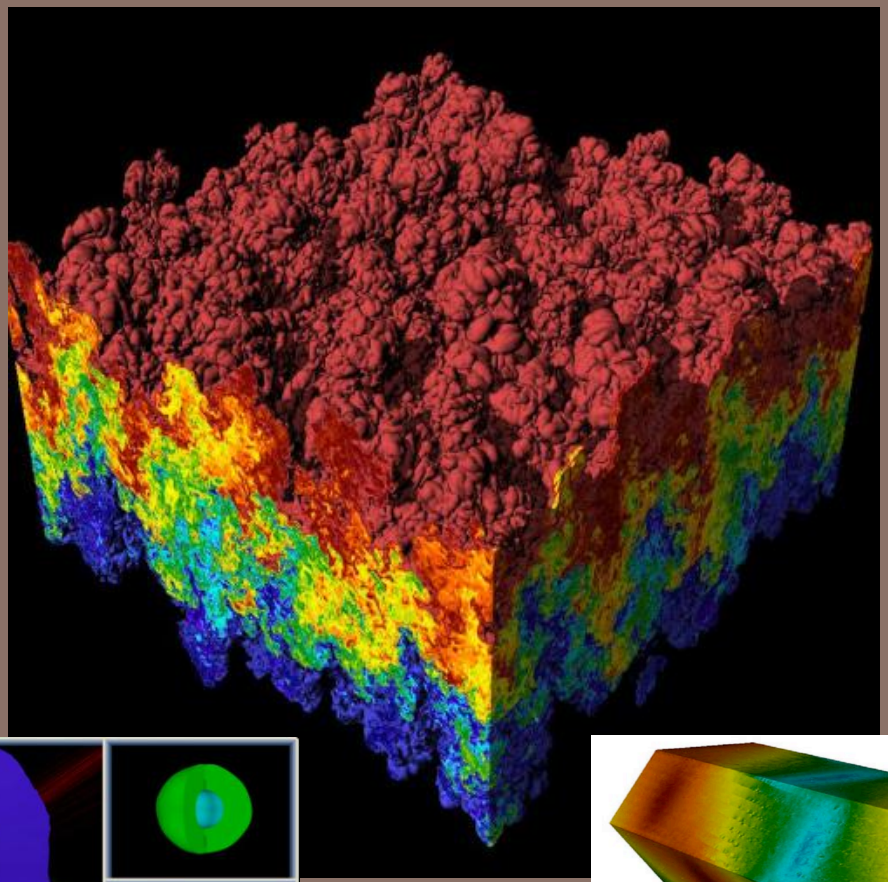
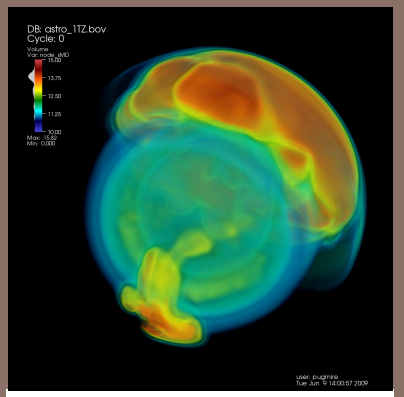
CIS 441/541: Introduction to Computer Graphics

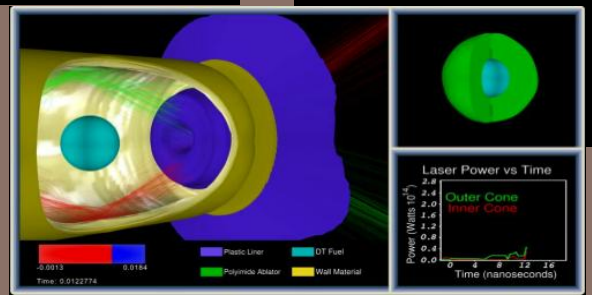
Lecture 16: textures

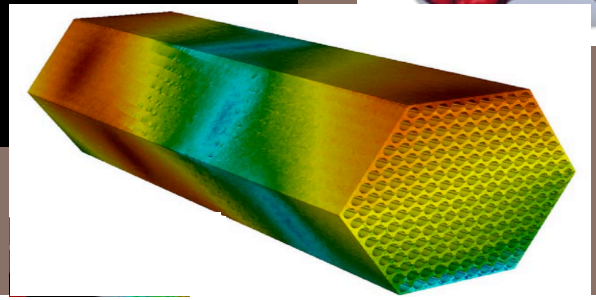












Oct. 28th, 2016

Hank Childs, University of Oregon

Announcements

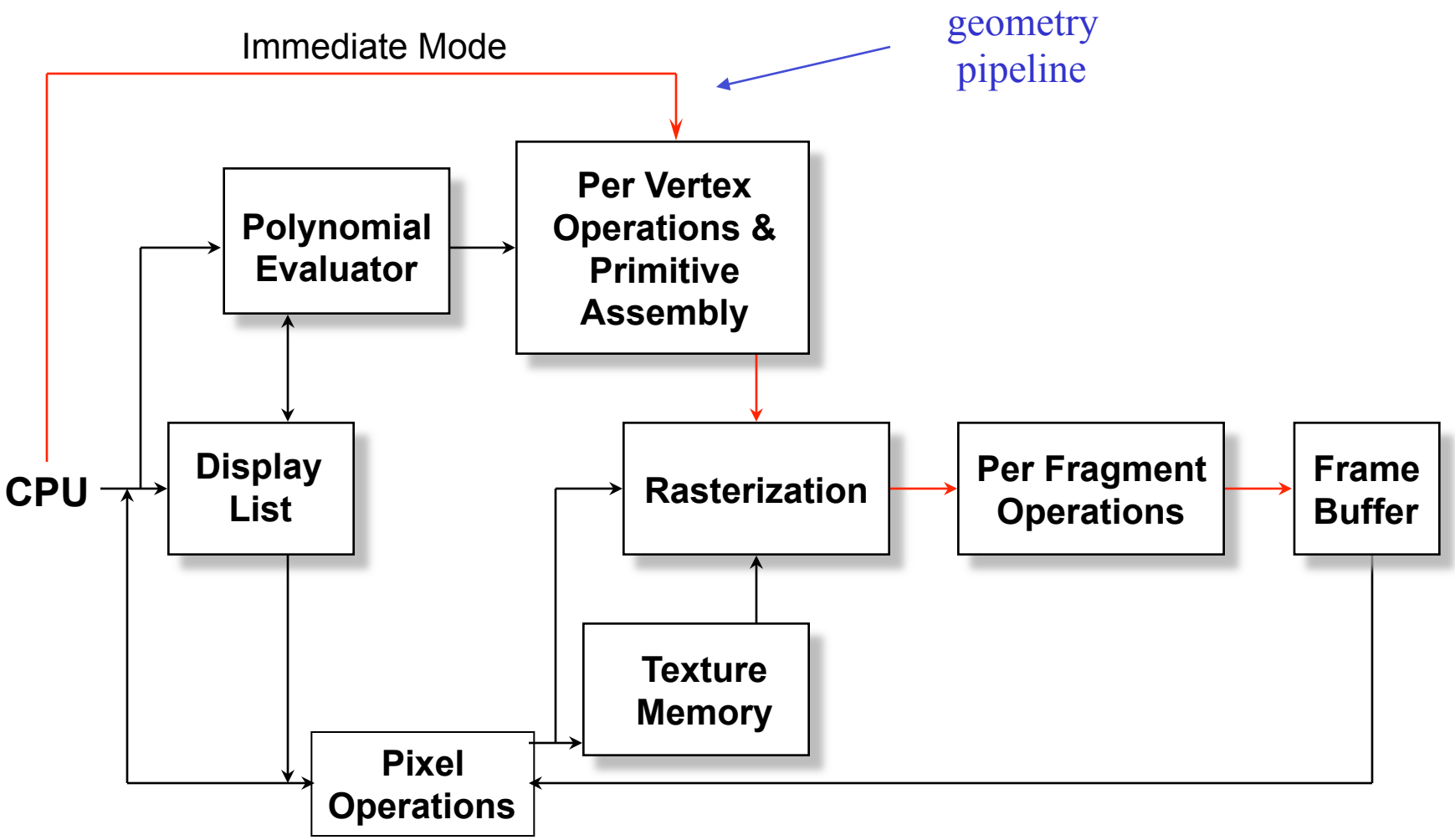
- Midterm on November 18th
- Old:
  - 441 students must do self-defined final project
- New:
  - 441 students do self-defined final project
  - -- or –
  - 441 students do additional projects, as defined by 541 students

Project 1

- Any questions on Project 1?
- (who made a movie?)

Review

OpenGL Architecture



OpenGL Functions

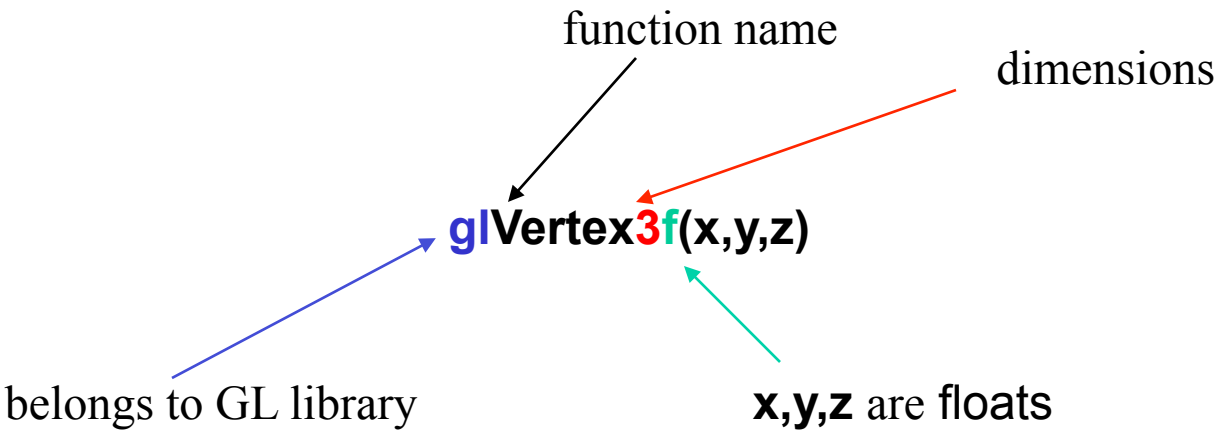
- Primitives
  - Points
  - Line Segments
  - Polygons
- Attributes
- Transformations
  - Viewing
  - Modeling
- Control (GLUT)
- Input (GLUT)
- Query

} VTK

OpenGL State

- OpenGL is a state machine
- OpenGL functions are of two types
  - Primitive generating
    - Can cause output if primitive is visible
    - How vertices are processed and appearance of primitive are controlled by the state
  - State changing
    - Transformation functions
    - Attribute functions

OpenGL function format



glVertex3fv(p)

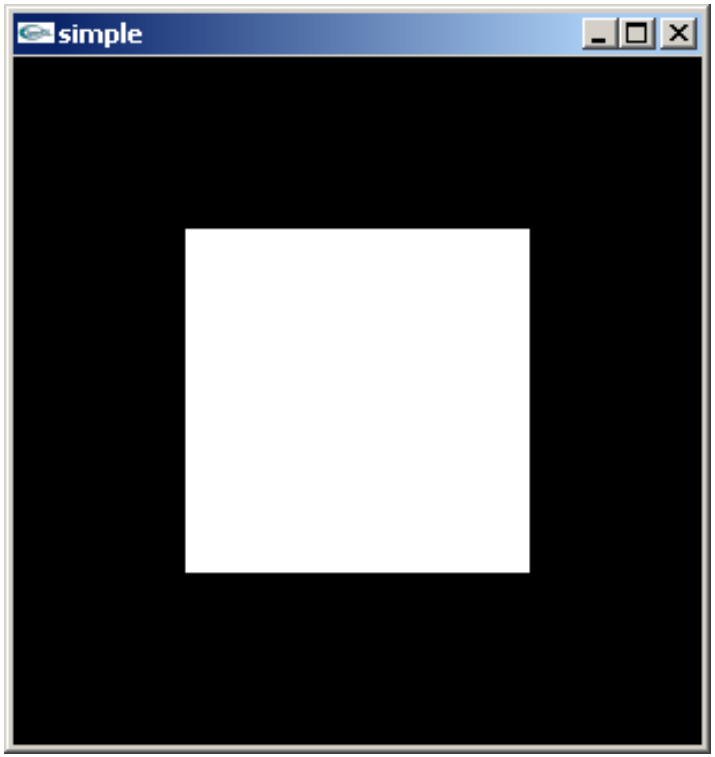
p is a pointer to an array

OpenGL #defines

- Most constants are defined in the include files **`gl.h`**, **`glu.h`** and **`glut.h`**
  - Note **`#include <GL/glut.h>`** should automatically include the others
  - Examples
  - **`glBegin(GL_POLYGON)`**
  - **`glClear(GL_COLOR_BUFFER_BIT)`**
- include files also define OpenGL data types: **`GLfloat`**, **`GLdouble`**,….

A Simple Program

Generate a square on a solid background



simple.c

```
#include <GL/glut.h>
void mydisplay(){
    glClear(GL_COLOR_BUFFER_BIT);
     glBegin(GL_POLYGON);
          glVertex2f(-0.5, -0.5);
          glVertex2f(-0.5, 0.5);
          glVertex2f(0.5, 0.5);
          glVertex2f(0.5, -0.5);
     glEnd();
     glFlush();
}
int main(int argc, char** argv){
     glutCreateWindow("simple");
     glutDisplayFunc(mydisplay);
     glutMainLoop();
}
```

Event Loop

- Note that the program defines a *display callback* function named **`mydisplay`**
  - Every glut program must have a display callback
  - The display callback is executed whenever OpenGL decides the display must be refreshed, for example when the window is opened
  - The **`main`** function ends with the program entering an event loop

```
VTK will be similar … callback
issued to render geometry
```

Defaults

- **`simple.c`** is too simple
- Makes heavy use of state variable default values for
  - Viewing
  - Colors
  - Window parameters
- Next version will make the defaults more explicit

How to make a graphics program?

- Need to create a window
  - This window contains a “context” for OpenGL to render in.
- Need to be able to deal with events/ interactions
- Need to render graphics primitives
  - OpenGL!

NEW STUFF

Windows and Events

- Creating windows and dealing with events varies from platform to platform.

- "Hello World" with X-Windows.

Compile with:

- gcc -L/usr/X11R6/lib -lX11 hello-x.c -o hello-x

```c
#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
   Display *d;
   Window w;
   XEvent e;
   char *msg = "Hello, World!";
   int s;

   d = XOpenDisplay(NULL);
   if (d == NULL) {
      fprintf(stderr, "Cannot open display\n");
      exit(1);
   }

   s = DefaultScreen(d);
   w = XCreateSimpleWindow(d, RootWindow(d, s), 10, 10, 100, 100, 1,
                           BlackPixel(d, s), WhitePixel(d, s));
   XSelectInput(d, w, ExposureMask | KeyPressMask);
   XMapWindow(d, w);

   while (1) {
      XNextEvent(d, &e);
      if (e.type == Expose) {
         XFillRectangle(d, w, DefaultGC(d, s), 20, 20, 10, 10);
         XDrawString(d, w, DefaultGC(d, s), 10, 50, msg, strlen(msg));
      }
      if (e.type == KeyPress)
         break;
   }

   XCloseDisplay(d);
   return 0;
}
```

Windows and Events

- Creating windows and dealing with events varies from platform to platform.
- Some packages provide implementations for key platforms (Windows, Unix, Mac) and abstractions for dealing with windows and events.
- GLUT: library for cross-platform windowing & events.
  - My experiments: doesn't work as well as it used to.
- VTK: library for visualization
  - But also contains cross-platform windowing & events.

The University of New Mexico

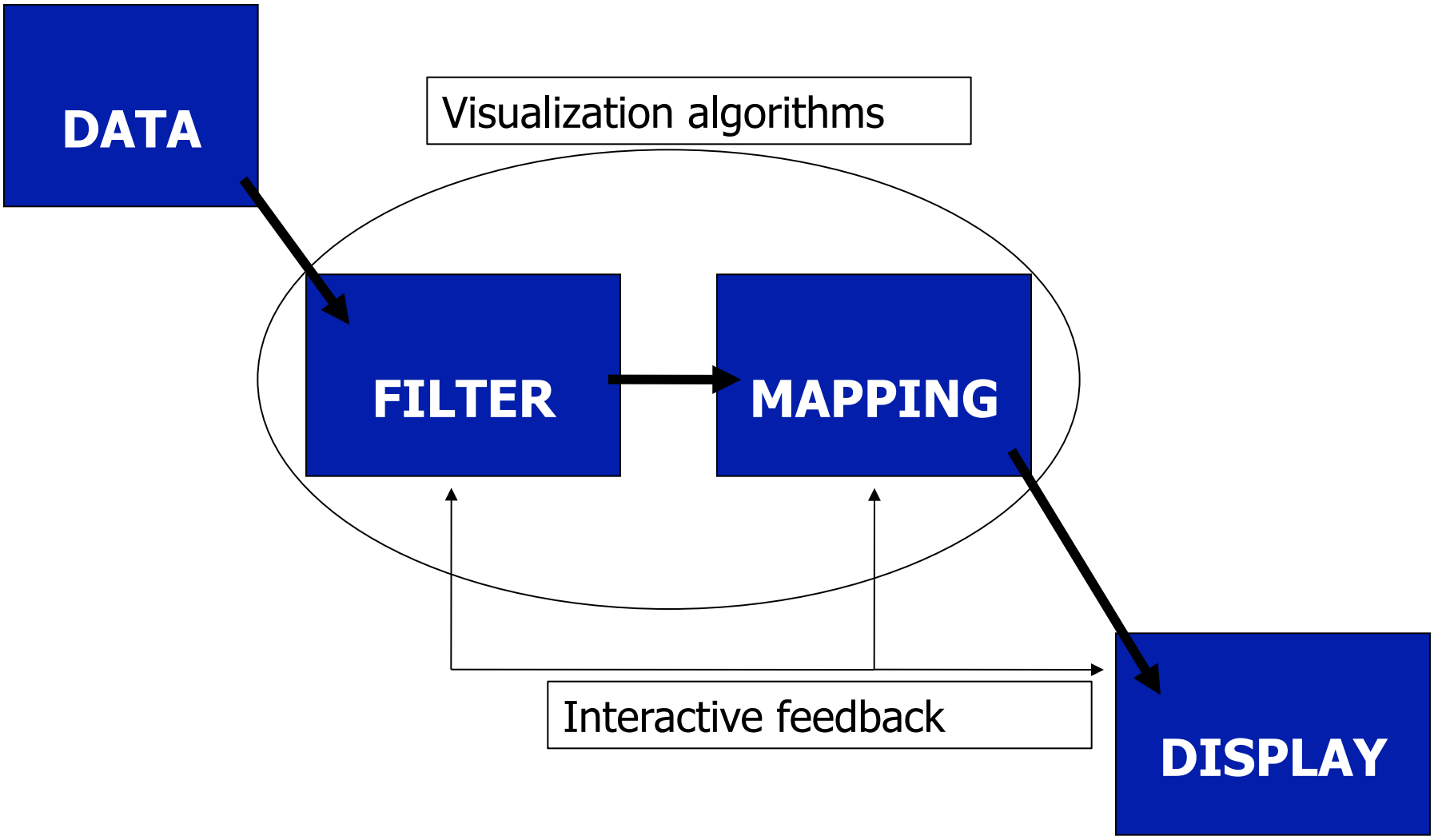
Visualization with VTK

Content from: Erik Vidholm, Univ of Uppsula, Sweden
David Gobbi, Robarts Research Institute, London, Ontario, Canada

VTK – The Visualization ToolKit

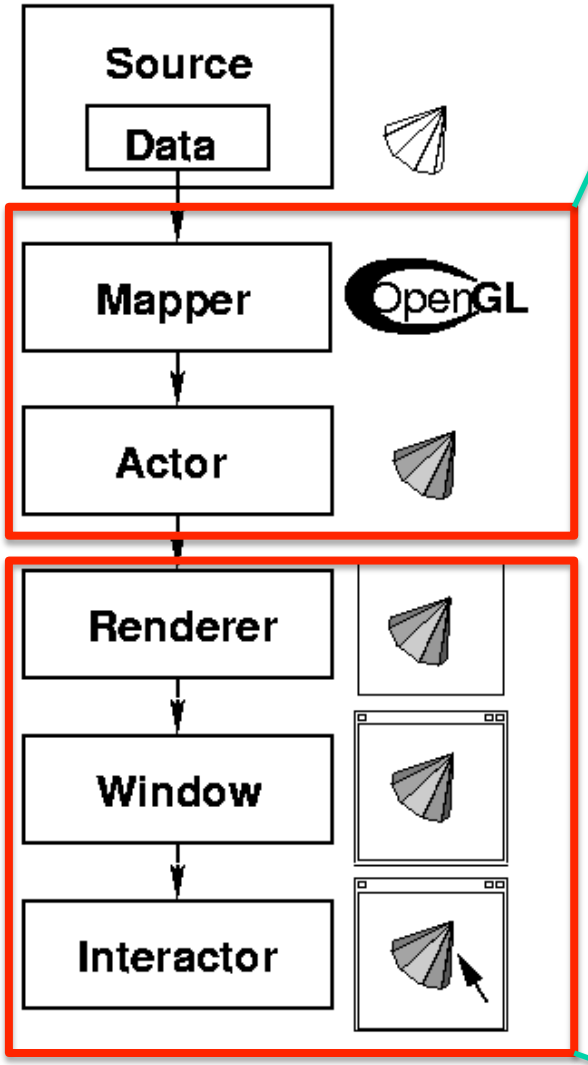
- Open source, freely available software for 3D computer graphics, image processing, and visualization
- Managed by Kitware Inc.
- Use C++, Tcl/Tk, Python, Java

The visualization pipeline



We will replace these and write our own GL calls.

Cone.py Pipeline Diagram (type "python Cone.py" to run)



Either reads the data from a file or creates the data from scratch.

```
from vtkpython import *

cone = vtkConeSource()
cone.SetResolution(10)
```

Moves the data from VTK into OpenGL.

```
coneMapper = vtkPolyDataMapper()
coneMapper.SetInput(cone.GetOutput())
```

For setting colors, surface properties, and the position of the object.

```
coneActor = vtkActor()
coneActor.SetMapper(coneMapper)
```

The rectangle of the computer screen that VTK draws into.

```
ren = vtkRenderer()
ren.AddActor(coneActor)
```

The window, including title bar and decorations.

```
renWin = vtkRenderWindow()
renWin.SetWindowName("Cone")
renWin.SetSize(300,300)
renWin.AddRenderer(ren)
```

Allows the mouse to be used to interact wth the data.

```
iren = vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
iren.Start()
```

We will re-use these.

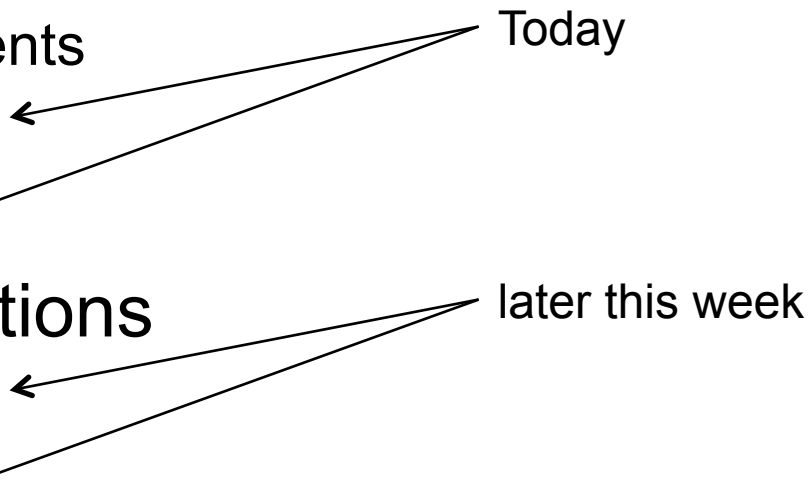
How to make a graphics program?

- Need to create a window
  - This window contains a “context” for OpenGL to render in.
- Need to be able to deal with events/ interactions
- Need to render graphics primitives
  - OpenGL!

Borrow Build

OpenGL Functions

- Primitives
  - Points
  - Line Segments
  - Polygons
- Attributes
- Transformations
  - Viewing
  - Modeling
- Control (VTK)
- Input (VTK)
- Query

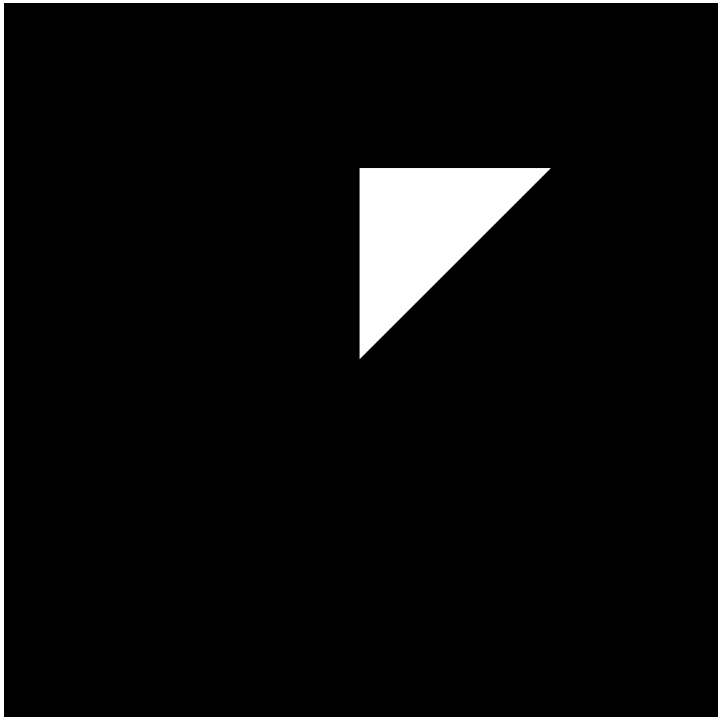


First OpenGL programs

- Remember: none of these programs have windowing or events
- They contain just the code to put primitives on the screen, with lighting and colors.

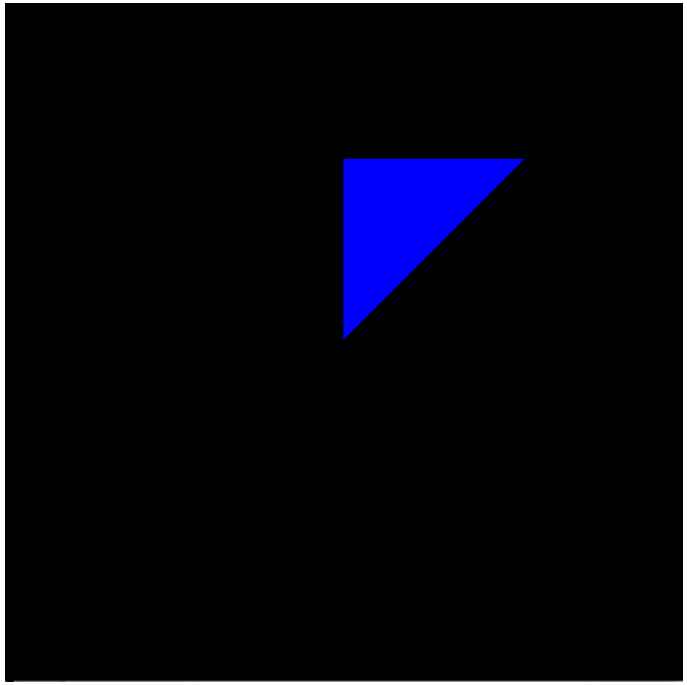
First OpenGL programs

```
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
 public:
   static vtk441PolyDataMapper *New();
   virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
   {
     float ambient[3] = { 1, 1, 1 };
     glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
     glBegin(GL_TRIANGLES);
     glVertex3f(0,0,0);
     glVertex3f(0,1,0);
     glVertex3f(1,1,0);
     glEnd();
   }
};
```



First OpenGL programs

```
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
 public:
   static vtk441PolyDataMapper *New();
   virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
   {
     glEnable(GL_COLOR_MATERIAL);
     float ambient[3] = { 1, 1, 1 };
     glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
     glBegin(GL_TRIANGLES);
     glColor3ub(0, 0, 255);
     glVertex3f(0,0,0);
     glVertex3f(0,1,0);
     glVertex3f(1,1,0);
     glEnd();
   }
};
```



glEnable/glDisable: important functions

www.opengl.org/sdk/docs/man/xhtml/glEnable.xml

Name

glEnable — enable or disable server-side GL capab

C Specification

```
void glEnable(GLenum cap);
```

Parameters

cap

Specifies a symbolic constant indicating a GL capability.

C Specification

```
void glDisable(GLenum cap);
```

Parameters

cap

Specifies a symbolic constant indicating a GL capability.

Both `glEnable` and glDisable take a single argument, *`cap`*, which can assume one of the following values:

`GL_BLEND`

If enabled, blend the computed fragment color values with the values in the color buffers. See glBlendFunc.

`GL_CULL_FACE`

If enabled, cull polygons based on their winding in window coordinates. See glCullFace.

`GL_DEPTH_TEST`

If enabled, do depth comparisons and update the depth buffer. Note that even if the depth buffer exists and the depth mask is non-zero, the depth buffer is not updated if the depth test is disabled. See glDepthFunc and glDepthRangef.

`GL_DITHER`

If enabled, dither color components or indices before they are written to the color buffer.

`GL_POLYGON_OFFSET_FILL`

If enabled, an offset is added to depth values of a polygon's fragments produced by rasterization. See glPolygonOffset.

`GL_SAMPLE_ALPHA_TO_COVERAGE`

If enabled, compute a temporary coverage value where each bit is determined by the alpha value at the corresponding sample location. The temporary coverage value is then ANDed with the fragment coverage value.

`GL_SAMPLE_COVERAGE`

If enabled, the fragment's coverage is ANDed with the temporary coverage value. If `GL_SAMPLE_COVERAGE_INVERT` is set to `GL_TRUE`, invert the coverage value. See glSampleCoverage.

`GL_SCISSOR_TEST`

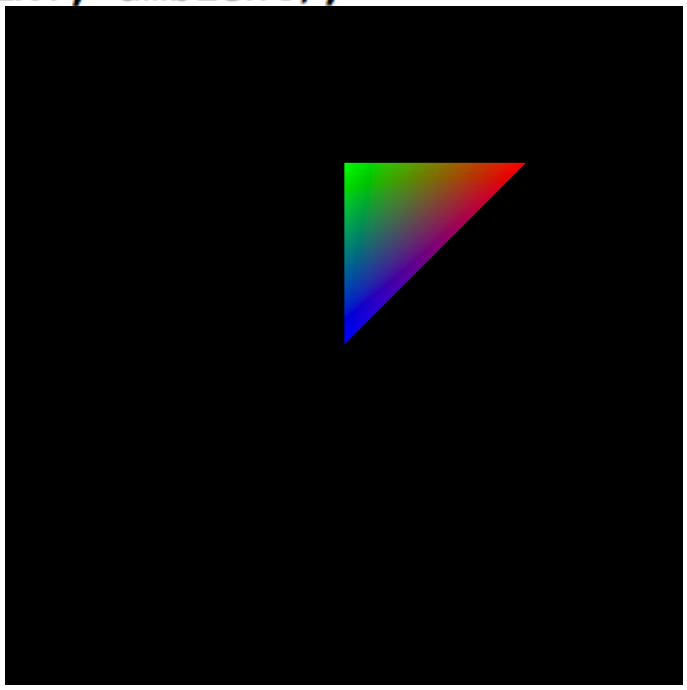
If enabled, discard fragments that are outside the scissor rectangle. See glScissor.

`GL_STENCIL_TEST`

If enabled, do stencil testing and update the stencil buffer. See glStencilFunc and glStencilOp.

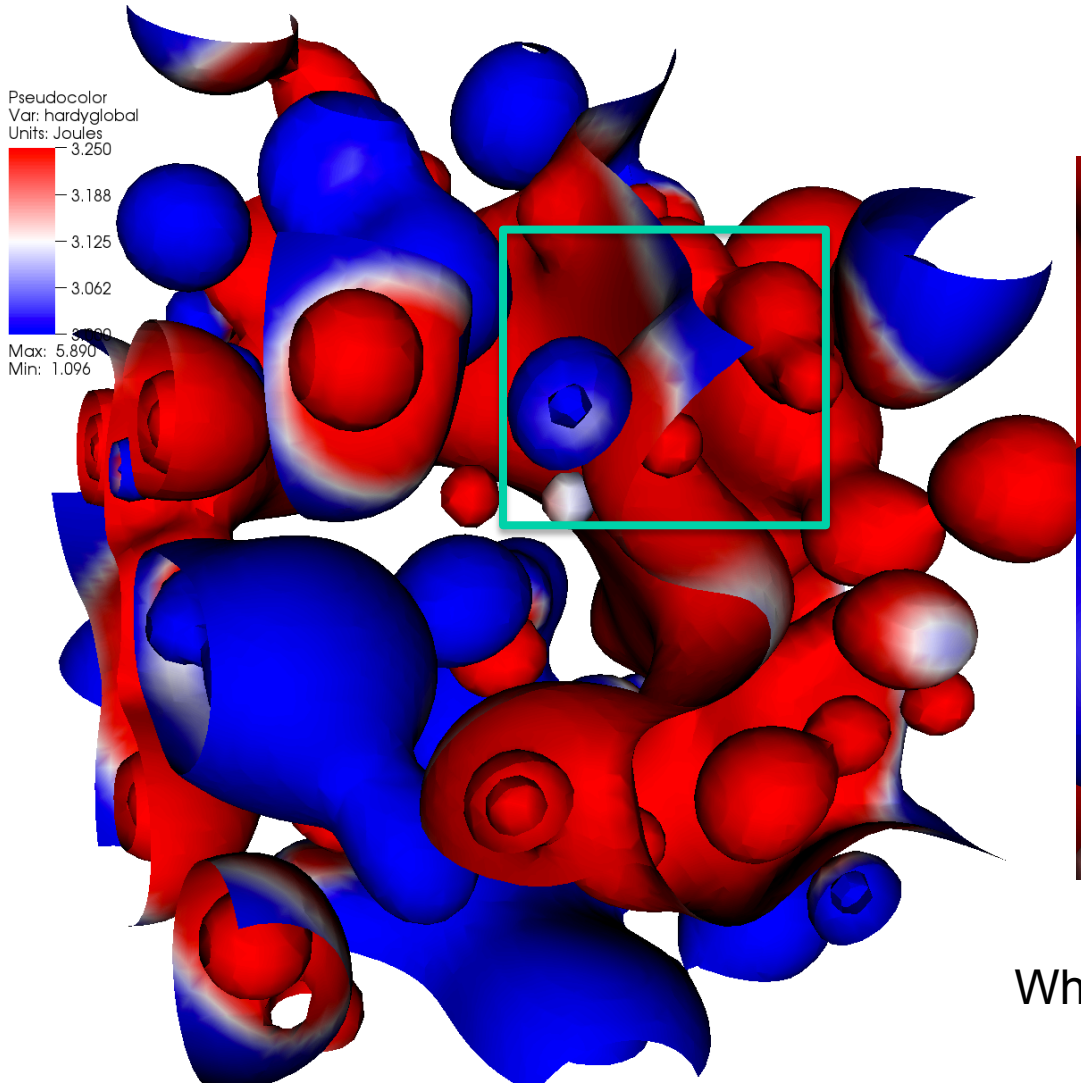
First OpenGL programs

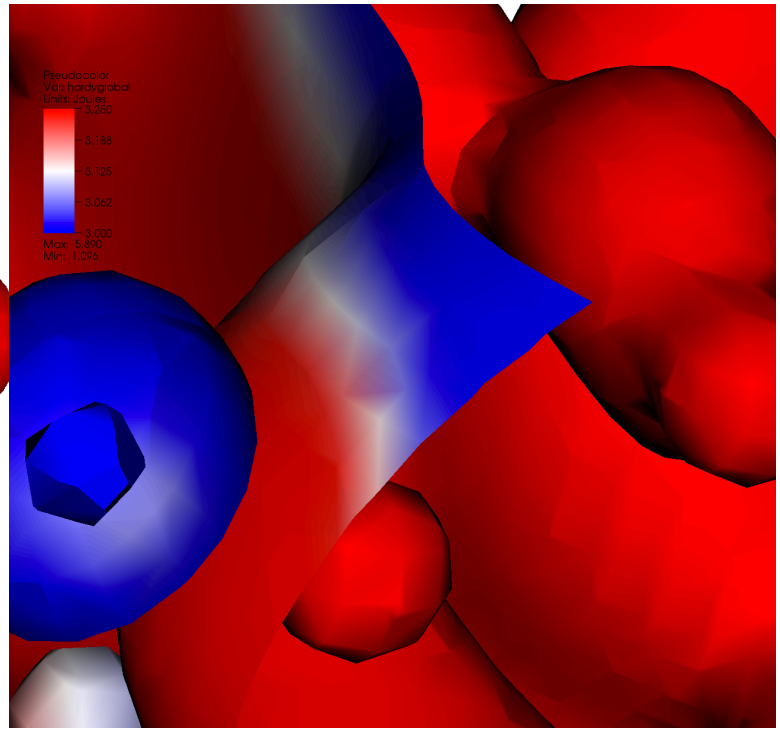
```
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
 public:
   static vtk441PolyDataMapper *New();
   virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
   {
     glEnable(GL_COLOR_MATERIAL);
     float ambient[3] = { 1, 1, 1 };
     glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
     glBegin(GL_TRIANGLES);
     glColor3ub(0, 0, 255);
     glVertex3f(0,0,0);
     glColor3ub(0, 255, 0);
     glVertex3f(0,1,0);
     glColor3ub(255, 0, 0);
     glVertex3f(1,1,0);
     glEnd();
   }
};
```



The University of New Mexico

Visualization use case

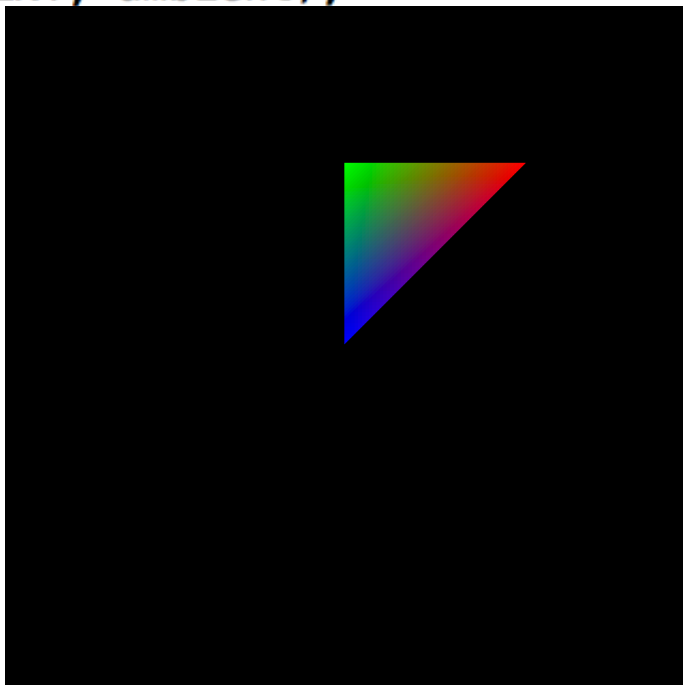




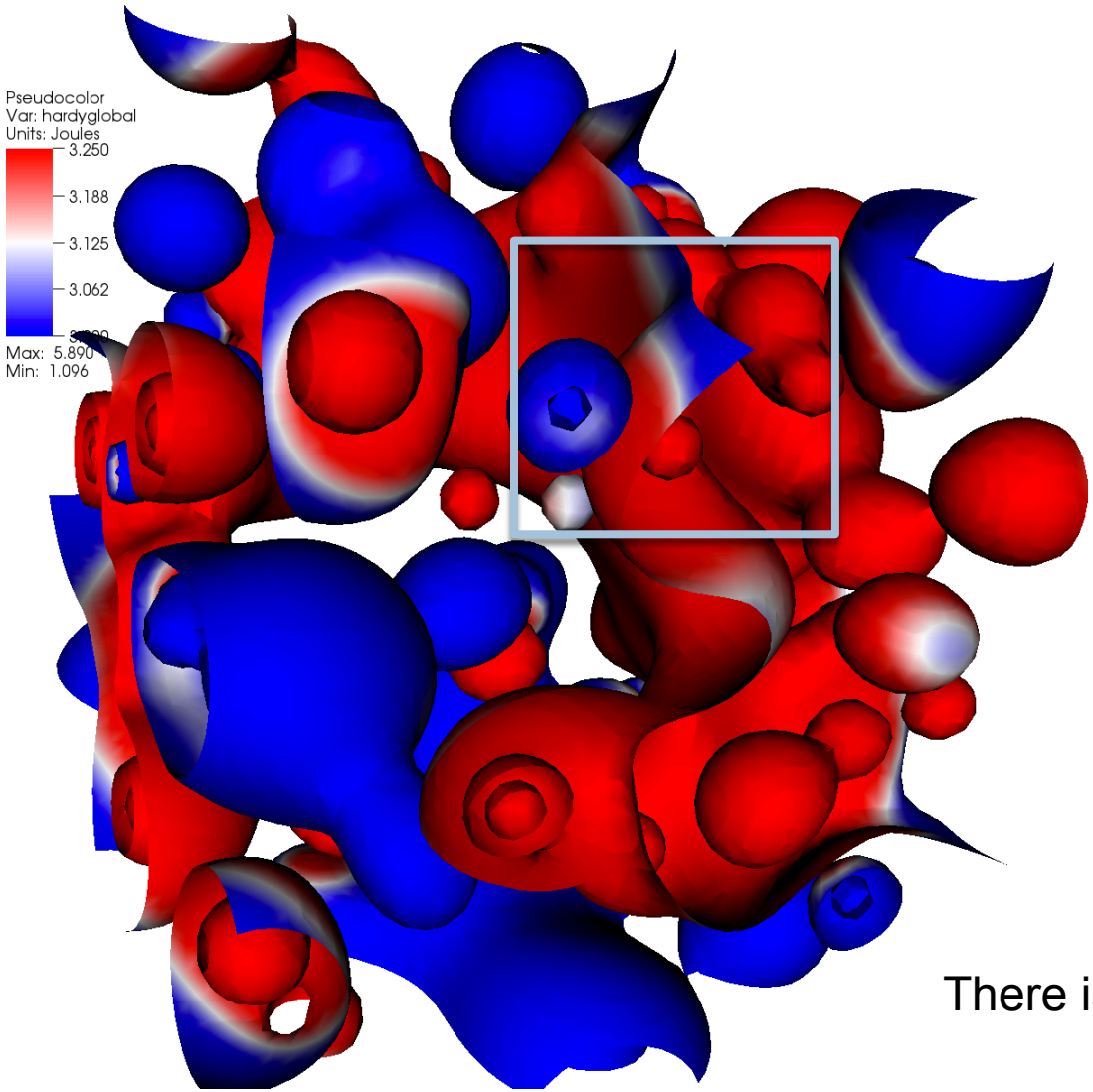
Why is there purple in this picture?

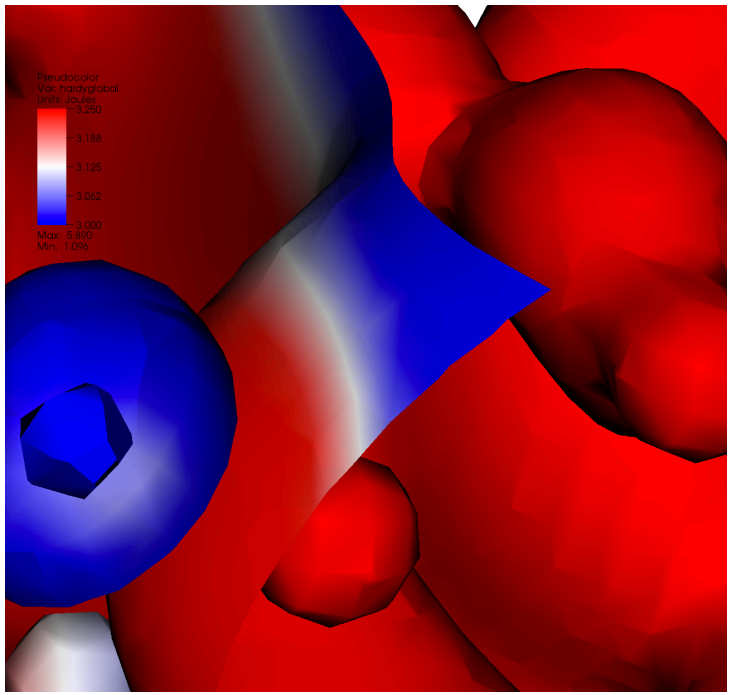
First OpenGL programs

```
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
 public:
   static vtk441PolyDataMapper *New();
   virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
   {
     glEnable(GL_COLOR_MATERIAL);
     float ambient[3] = { 1, 1, 1 };
     glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
     glBegin(GL_TRIANGLES);
     glColor3ub(0, 0, 255);
     glVertex3f(0,0,0);
     glColor3ub(0, 255, 0);
     glVertex3f(0,1,0);
     glColor3ub(255, 0, 0);
     glVertex3f(1,1,0);
     glEnd();
   }
};
```



Textures: a better way to specify a color map





There is no purple when we use textures

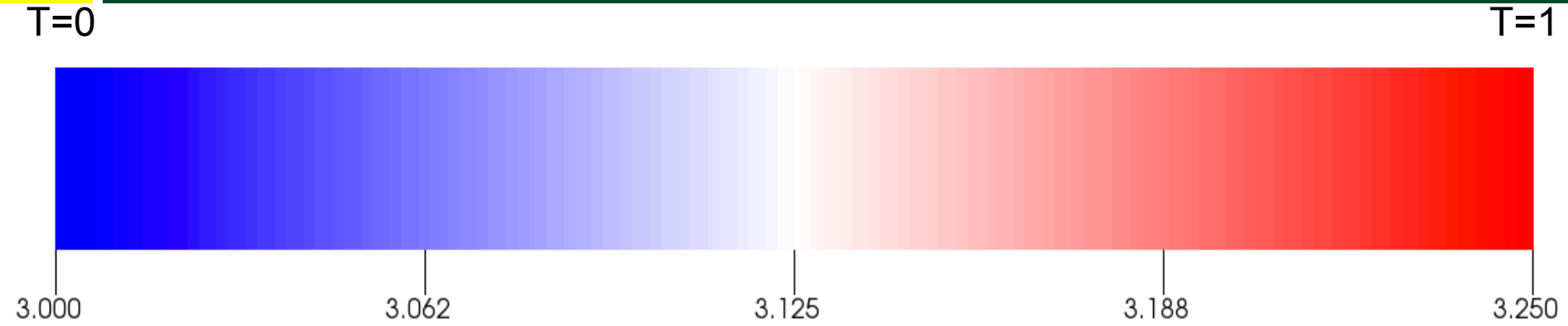
Textures

- "Textures" are a mechanism for adding "texture" to surfaces.
  - Think of texture of a cloth being applied to a surface
  - Typically used in 2D form
- We will start with a 1D form, and work our way up to 2D later.

1D textures: basic idea

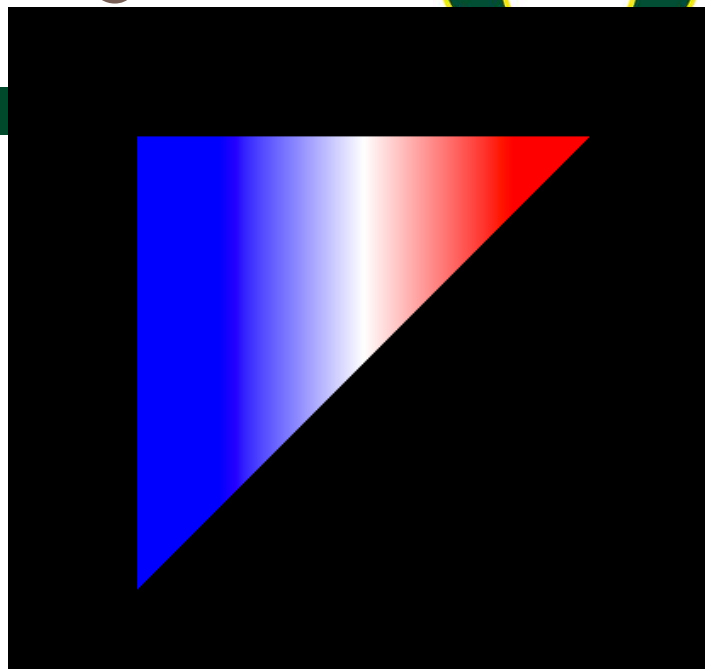
- Store color map on GPU as a texture
  - An array of colors
- Old color interpolation of fragment on a scanline:
  - For (int j = 0 ; j < 3 ; j++)
    - RGB[j] = leftRGB[j] + proportion*(rightRGB[j]-leftRGB[j])
- New color interpolation of fragment on a scanline:
  - textureVal = leftTextureVal
    + proportion*(rightTextureVal-leftTextureVal)
  - RGB ← textureLookup[textureVal]

Example



- Triangle with vertices with scalar values 2.9, 3.3, and 3.1.
- T for 2.9 = (2.9-3.0)/(3.25-3) = -0.4
- T for 3.1 = (3.1-3.0)/(3.25-3) = 0.4
- T for 3.3 = (3.3-3.0)/(3.25-3) = 1.2
- Fragment colors come from interpolating texture coordinates and applying texture

First OpenGL Texture Program



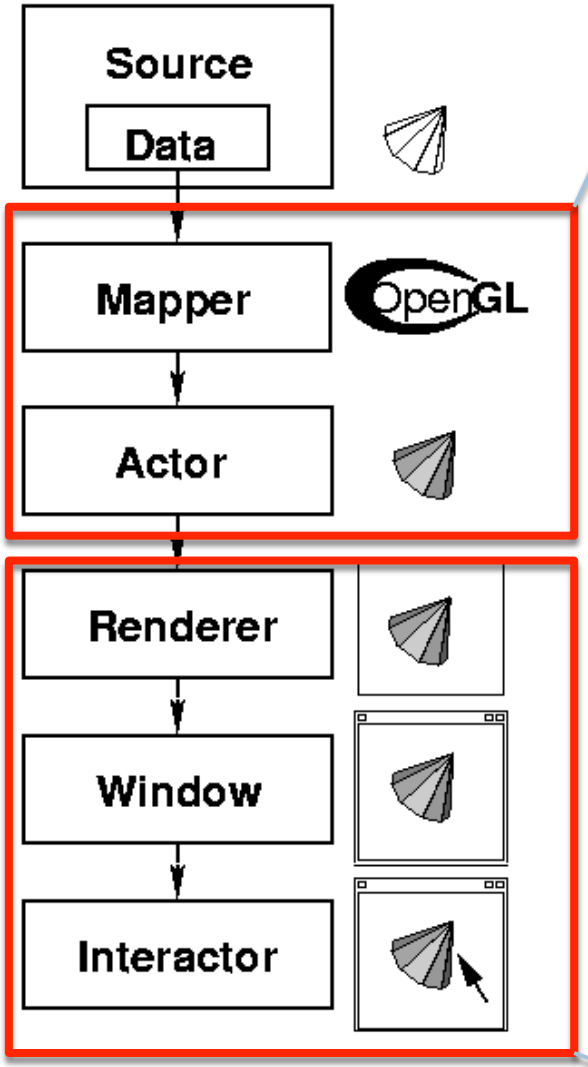
```
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
 public:
   static vtk441PolyDataMapper *New();

   virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
   {
     GLubyte Texture3[9] = {
          0, 0, 255,   // blue
          255, 255, 255,   // white
          255, 0, 0, // red
     };
     glTexImage1D(GL_TEXTURE_1D, 0, GL_RGB, 3, 0, GL_RGB,
                  GL_UNSIGNED_BYTE, Texture3);
     glEnable(GL_COLOR_MATERIAL);
     glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
     glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

     glEnable(GL_TEXTURE_1D);
     float ambient[3] = { 1, 1, 1 };
     glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
     glBegin(GL_TRIANGLES);
     glTexCoord1f(0);
     glVertex3f(0,0,0);
     glTexCoord1f(0.0);
     glVertex3f(0,1,0);
     glTexCoord1f(1.);
     glVertex3f(1,1,0);
     glEnd();
   }
};
```

(advanced texture features & 2D textures on Weds)

Project 2A

We will replace these and write our own GL calls.

Cone.py Pipeline Diagram (type "python Cone.py" to run)



Either reads the data from a file or creates the data from scratch.

```
from vtkpython import *

cone = vtkConeSource()
cone.SetResolution(10)
```

Moves the data from VTK into OpenGL.

```
coneMapper = vtkPolyDataMapper()
coneMapper.SetInput(cone.GetOutput())
```

For setting colors, surface properties, and the position of the object.

```
coneActor = vtkActor()
coneActor.SetMapper(coneMapper)
```

The rectangle of the computer screen that VTK draws into.

```
ren = vtkRenderer()
ren.AddActor(coneActor)
```

The window, including title bar and decorations.

```
renWin = vtkRenderWindow()
renWin.SetWindowName("Cone")
renWin.SetSize(300,300)
renWin.AddRenderer(ren)
```

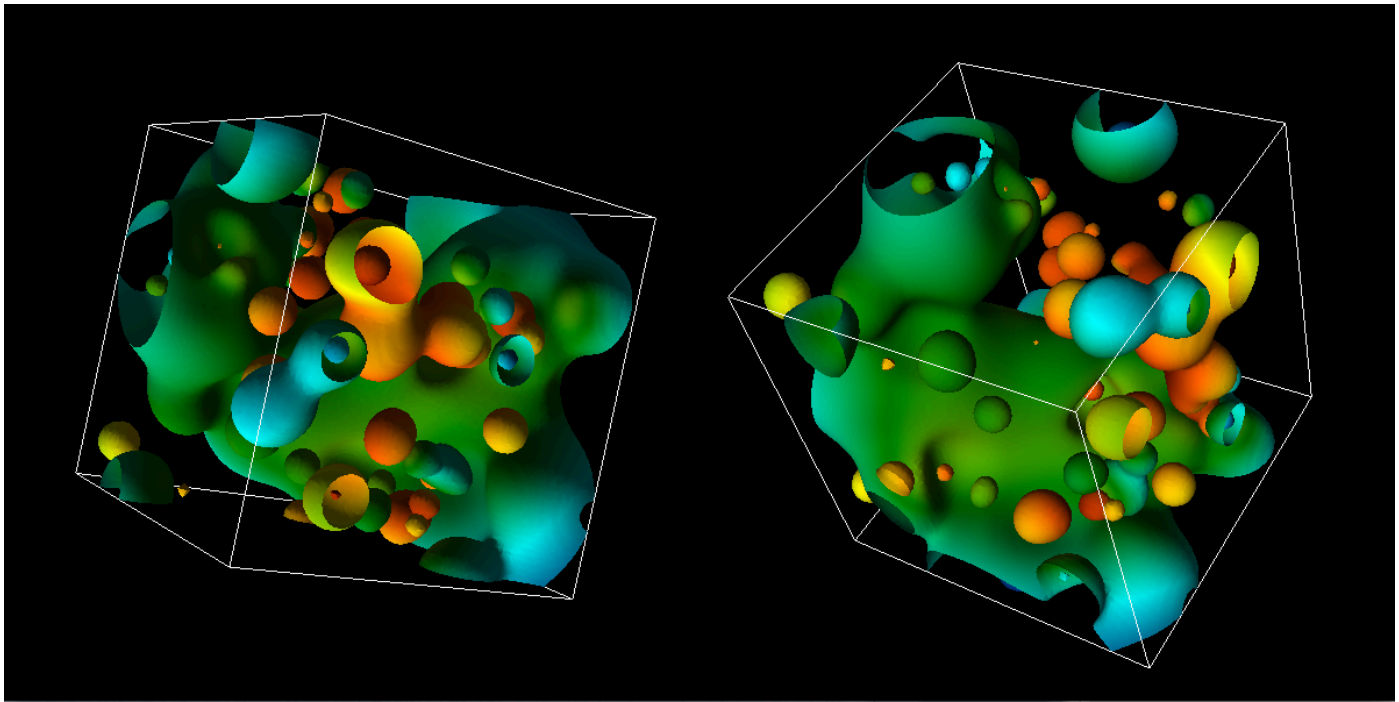
Allows the mouse to be used to interact wth the data.

```
iren = vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
iren.Start()
```

We will re-use these.

Project #2A (8%), Due Nov. 7th

- Goal: OpenGL program that does regular colors and textures
- New VTK-based project2A.cxx
- New CMakeLists.txt (but same as old ones)



Hints

- I recommend you “walk before you run” & “take small bites”. OpenGL can be very punishing. Get a picture up and then improve on it. Make sure you know how to retreat to your previously working version at every step.
- OpenGL “state thrashing” is common and tricky to debug.
  - Get one window working perfectly.
  - Then make the second one work perfectly.
  - Then try to get them to work together.
    - Things often go wrong, when one program leaves the OpenGL state in a way that doesn’t suit another renderer.

Hints

- MAKE MANY BACKUPS OF YOUR PROGRAM

- If the program doesn't run with VTK 7, use VTK 6
- If you are having issues on your laptop with a GL program, then use Room 100
  - (There's only 2 of these projects)