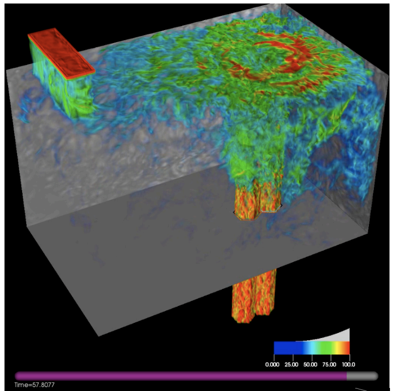
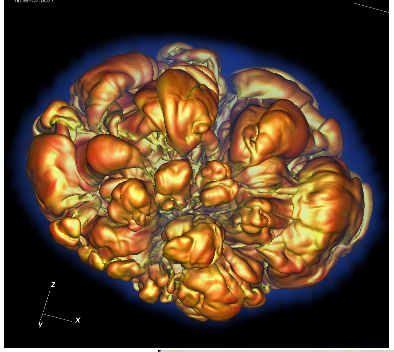
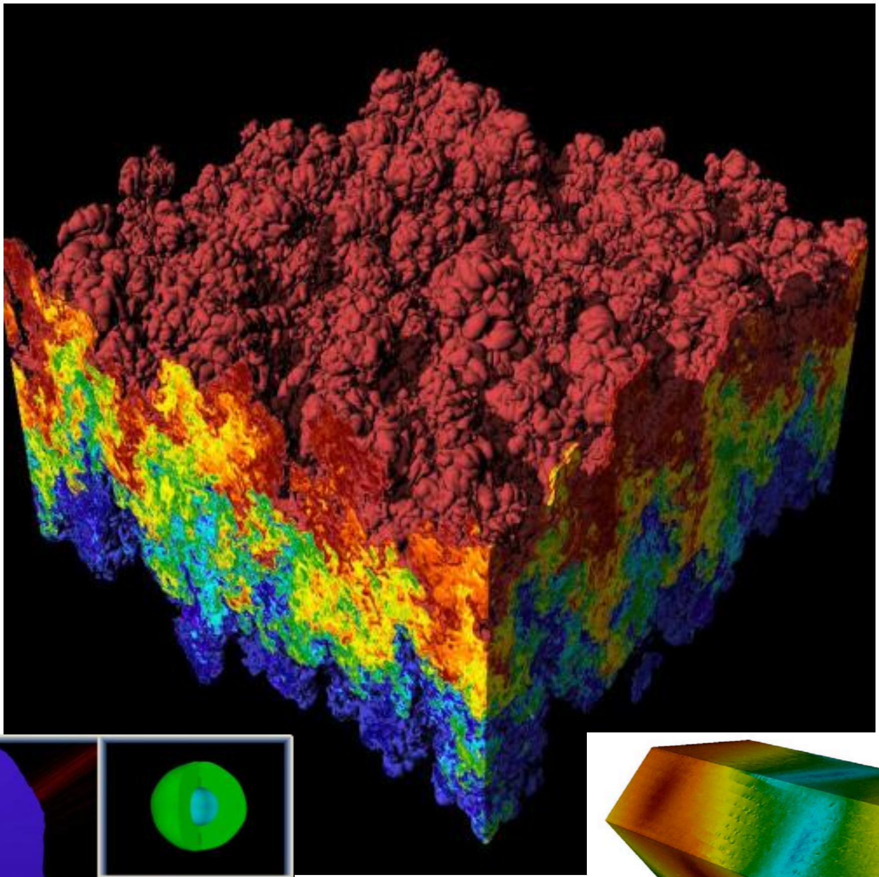
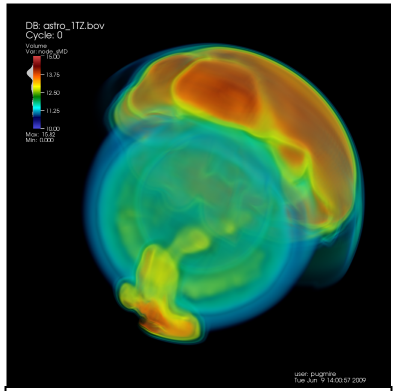
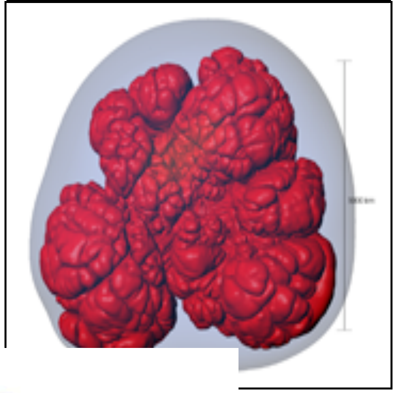
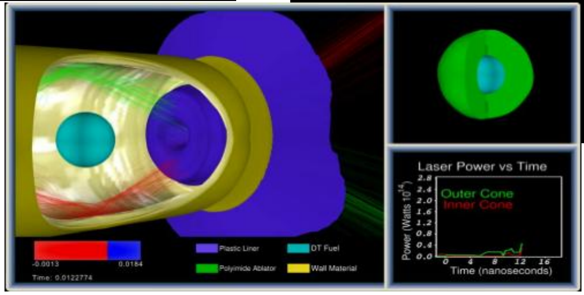
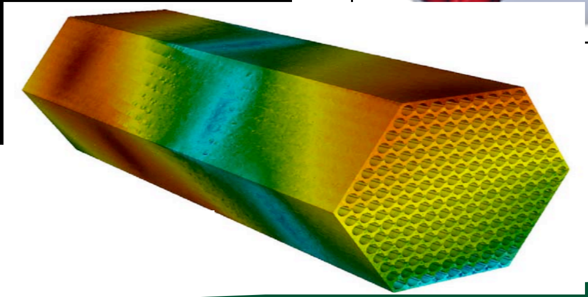
CIS 441/541: Intro to Computer Graphics
Lecture 10&11: Even More OpenGL!





February 19/21, 2019

Hank Childs, University of Oregon

Office Hours: Week 7

- Monday: 1-2 (Roscoe)
- Tuesday: 1230-115 (Hank)
- Tuesday: 1-2 (Roscoe)
- Wednesday: 1-3 (Roscoe)
- Thursday: 1130-1230 (Hank)
- Friday: 1130-1230 (Hank)

Timeline (1/2)

- 1F: assigned Feb 7th, due Feb 19th
  - → not as tough as 1E
- 2A: posted now, due ~~Feb 21st~~ Feb 23rd
- → you need to work on both 1F and 2A during Week 6 (Feb 11-15)
- 2B: posted now, due Feb 27th
- YouTube lectures for Feb 12th and 14th

UNIVERSITY OF OREGON

Timeline (2/2)

Sun	Mon	Tues	Weds	Thurs	Fri	Sat
	Feb 4	Feb 5 Lec 8	Feb 6 1E due	Feb 7 Begin 1F, begin 2A	Feb 8	Feb 9
Feb 10	Feb 11	~~YouTube~~	Feb 13	~~YouTube??~~	Feb 15	Feb 16
Feb 17	Feb 18	Feb 19 1F due	Feb 20	Feb 21 ~~2A due~~, begin 2B	Feb 22	Feb 23 2A due

Midterm

- Date: Tues Feb 26th
- Considering different plan: 25 & 5
  - Still no feedback received
- Details:
  - No notes.
  - Not expected to memorize Phong shading equation.
  - Will be derived directly from 1A-1F, 2A. Not 2B.

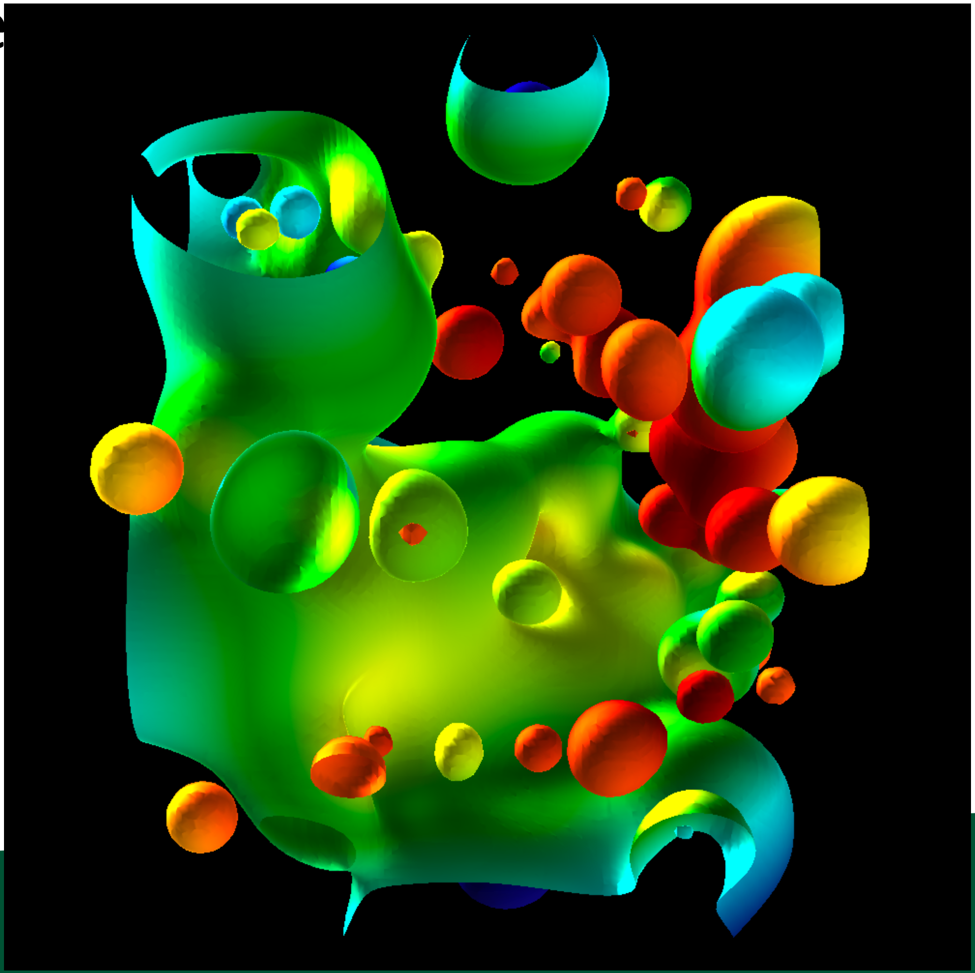
Midterm

- Questions will mostly derive from Project 1A, 1B, 1C, 1D, 1E, 1F, 2A
- Example: here's a triangle, what pixels does it contribute to?
  - Example: write GL program to do something
    - errors in syntax will receive minor deductions
- No notes, closed book, no calculators, internet, etc.

Questions on 1F?

Project #1F (8%), Due Feb 19th

- Goal: add shading, movie
- Extend your project1E code
- Important:
- add #define NORMALS



Changes to data structures

```
class Triangle
{
  public:
      double X[3], Y[3], Z[3];
      double colors[3][3];
      double normals[3][3];
};
```

→reader1e.cxx::GetTriangles() will not compile (with #define NORMALS) until you make these changes

→Now initializes normals at each vertex

More comments (1/3)

- This project in a nutshell:
  - Add method called “CalculateShading”.
    - My version of CalculateShading is about ten lines of code.
  - Call CalculateShading for each vertex
  - This is a new field, which you will LERP.
  - Modify RGB calculation to use shading.

More comments (2/3)

- Data to help debug
  - – I will make the shading value for each pixel available.
  - – I will also make it available for ambient, diffuse, specular.
- Don’t forget to do two-sided lighting for diffuse, one-sided lighting for specular

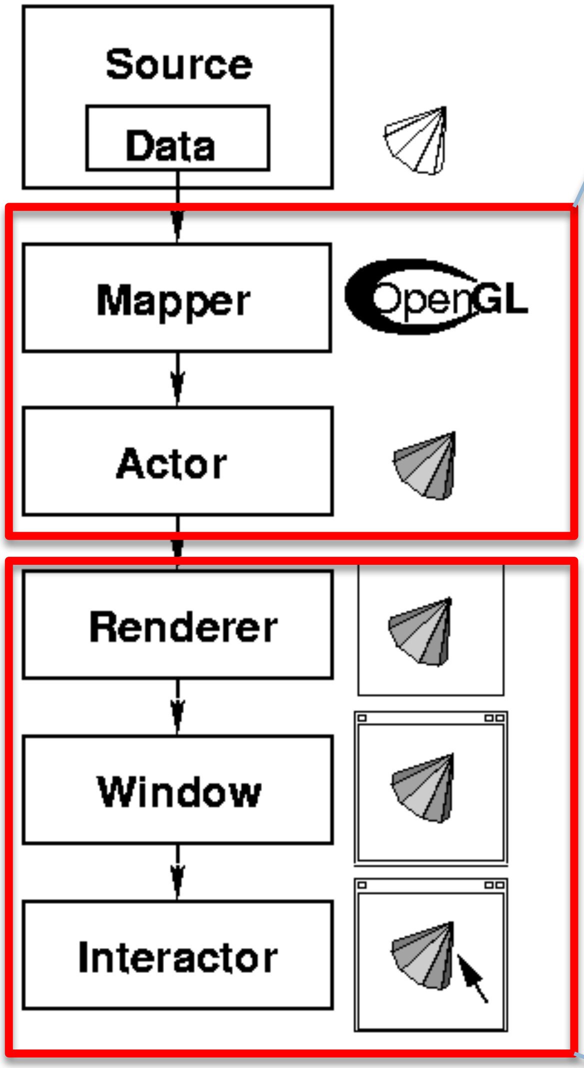
More comments (3/3)

- I haven’t said anything about movie encoders
- ffmpeg

Questions on 2A?

We will replace these and write our own GL calls.

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



```
from vtkpython import *
```

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

```
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)
```

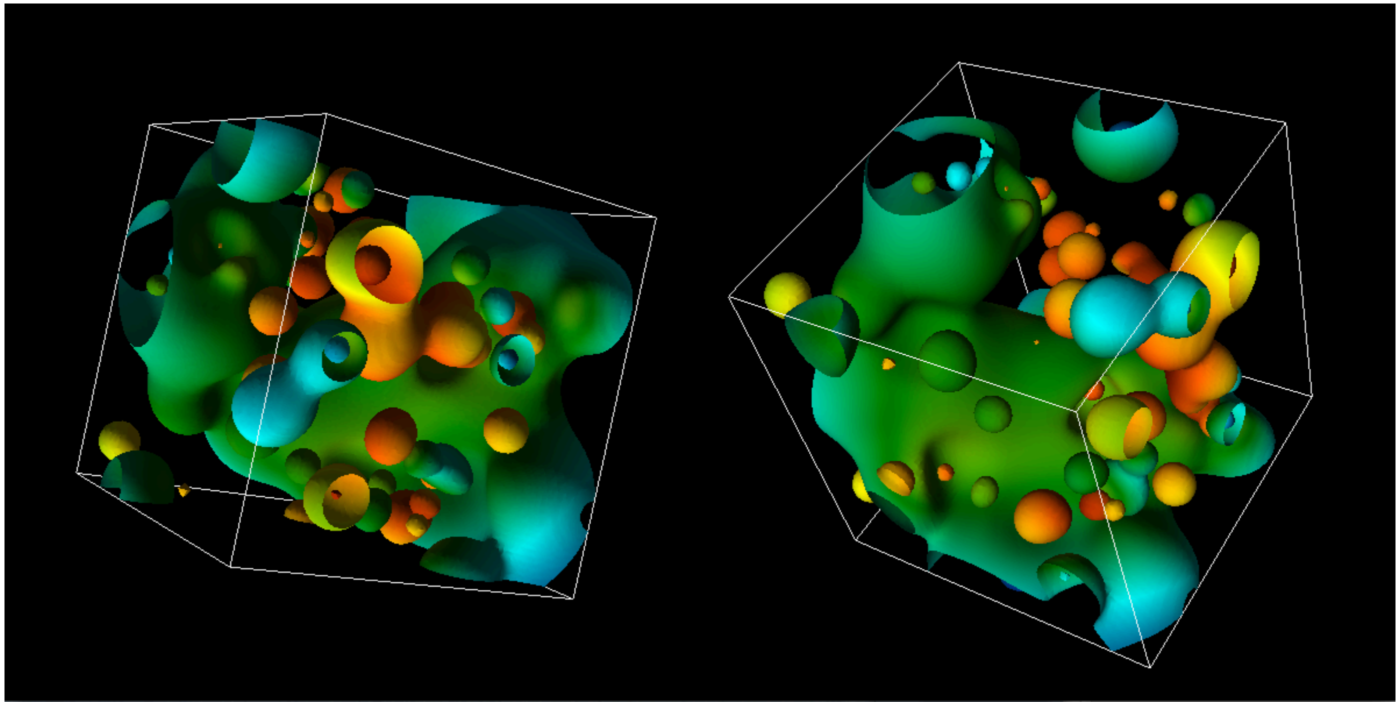
Allows the mouse to be used to interact wth the data.

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

We will re-use these.

Project #2A (8%), Due Feb. 23rd

- 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

- 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)

How to do colors (traditional)…

- The Triangle class now has a "fieldValue" data member, which ranges between 0 and 1.
- You will map this to a color using the GetColorMap function.
  - GetColorMap returns 256 colors.
- Mappings
  - A fieldValue value of 0 should be mapped to the first color
  - A fieldValue value of 1 should be mapped to the 255th color.
  - Each fieldValue in between should be mapped to the closest color of the 256, but interpolation of colors is not required.

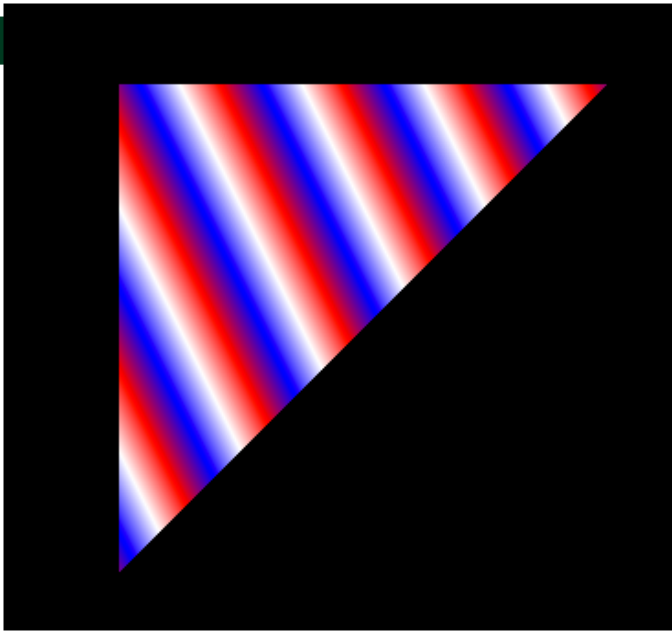
How to do colors (texture)…

- Same idea, but use texture infrastructure
- (easier)

More on Textures

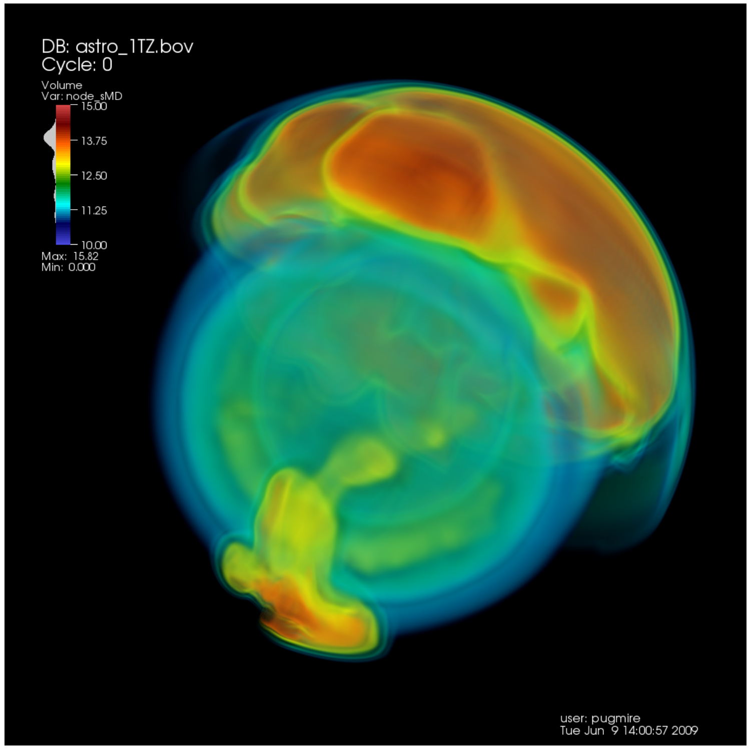
Textures with GL_REPEAT

```
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_REPEAT);
     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(-2);
     glVertex3f(0,0,0);
     glTexCoord1f(0);
     glVertex3f(0,1,0);
     glTexCoord1f(4.);
     glVertex3f(1,1,0);
     glEnd();
   }
};
```



1D, 2D, 3D textures

- 2D textures most common
- 1D textures: color maps (e.g., what we just did)
- 3D textures: "volume rendering"
  - Use combination of opacity and color (i.e., RGBA)



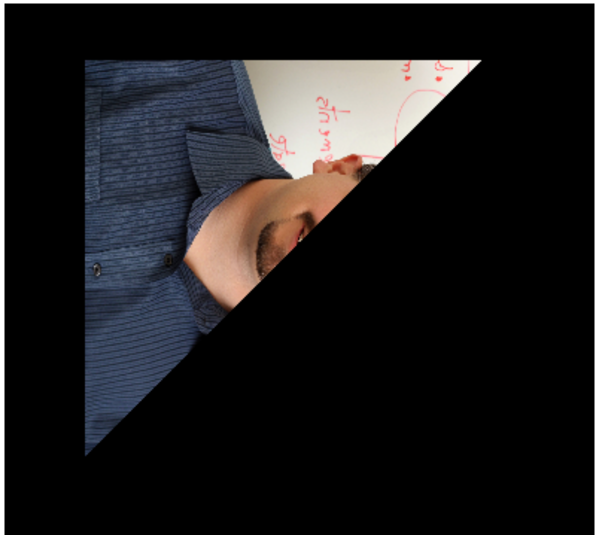
2D Textures

- Pre-rendered images painted onto geometry
- glTexImage1D → glTexImage2D
- GL_TEXTURE_WRAP_S → GL_TEXTURE_WRAP_S + GL_TEXTURE_WRAP_T
- glTexCoord1f → glTexCoord2f

2D Texture Program

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

   virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
   {
     vtkJPEGReader *rdr = vtkJPEGReader::New();
     rdr->SetFileName("HankChilds_345.jpg");
     rdr->Update();
     vtkImageData *img = rdr->GetOutput();
     int dims[3];
     img->GetDimensions(dims);
     unsigned char *buffer = (unsigned char *) img->GetScalarPointer(0,0,0);
     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, dims[0], dims[1], 0, GL_RGB,
                  GL_UNSIGNED_BYTE, buffer);
     glEnable(GL_COLOR_MATERIAL);
     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

     glEnable(GL_TEXTURE_2D);
     float ambient[3] = { 1, 1, 1 };
     glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
     glBegin(GL_TRIANGLES);
     glTexCoord2f(0,0);
     glVertex3f(0,0,0);
     glTexCoord2f(1, 0);
     glVertex3f(0,1,0);
     glTexCoord2f(1., 1.);
     glVertex3f(1,1,0);
     glEnd();
   }
};
```

What do we expect the output to be?

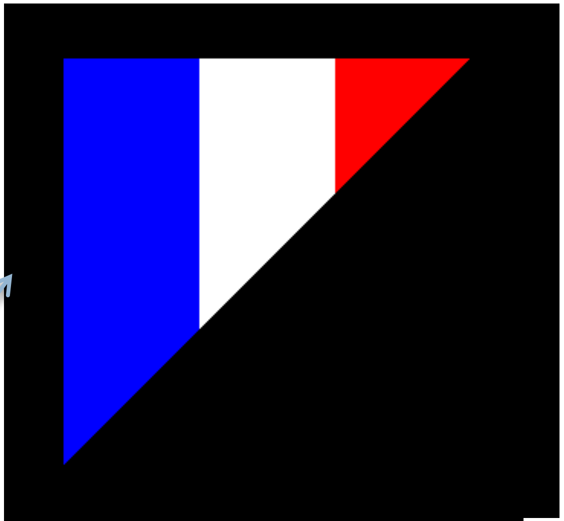
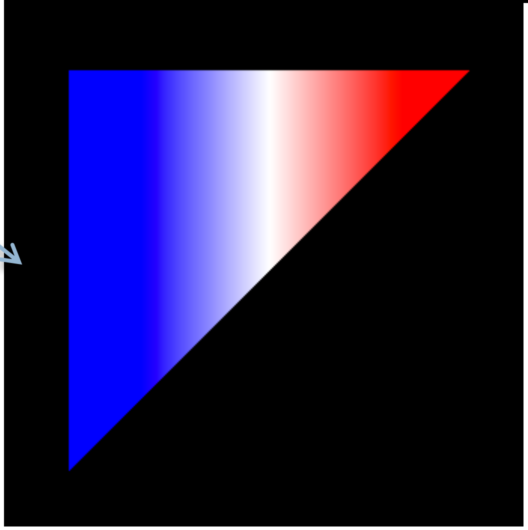


GL_TEXTURE_MIN_FILTER / GL_TEXTURE_MAG_FILTER

- Minifying: texture bigger than triangle.
  - How to map multiple texture elements onto a pixel?
    - GL_NEAREST: pick closest texture
    - GL_LINEAR: average neighboring textures
- Magnifying (GL_TEXTURE_MAG_FILTER): triangle bigger than texture
  - How to map single texture element onto multiple pixels?
    - GL_NEAREST: no interpolation
    - GL_LINEAR: interpolate with neighboring textures

GL_TEXTURE_MAG_FILTER with NEAREST and LINEAR

```
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_MAG_FILTER, GL_NEAREST);
    --- OR ---
     glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    ------
     glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

     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();
   }
};
```

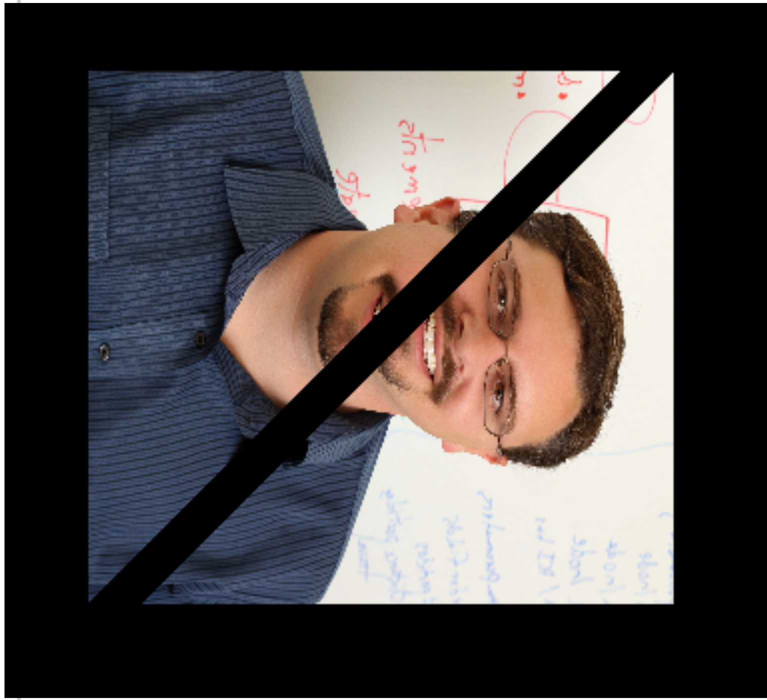



2D Texture Program

```
virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
{
   vtkJPEGReader *rdr = vtkJPEGReader::New();
   rdr->SetFileName("HankChilds_345.jpg");
   rdr->Update();
   vtkImageData *img = rdr->GetOutput();
   int dims[3];
   img->GetDimensions(dims);
   unsigned char *buffer = (unsigned char *) img->GetScalarP
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, dims[0], dims[1], 0,
                GL_UNSIGNED_BYTE, buffer);
   glEnable(GL_COLOR_MATERIAL);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

   glEnable(GL_TEXTURE_2D);
   float ambient[3] = { 1, 1, 1 };
   glMat
   glBeg
   glTex
   glVer
   glTex
   glVer
   glTex
   glVer

   glTex
   glVer
   glTexCo
   glVertex3f(1.1,0,0);
   glTexCoord2f(0, 0);
   glVertex3f(0.1,0,0);

   glEnd();
}
```

Texture is not 1:1, should probably scale geometry.

This is a terrible program … why?



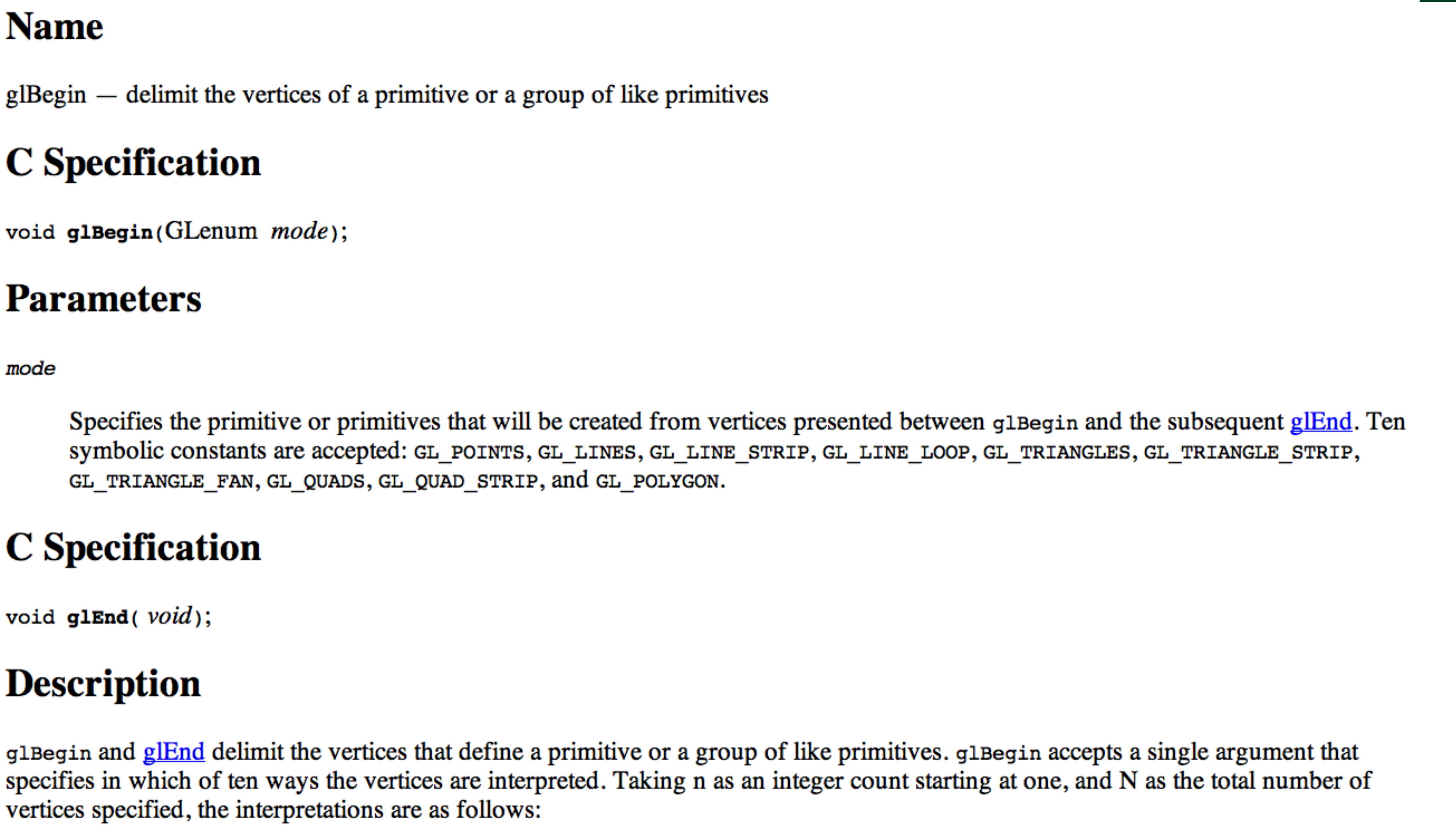
glBindTexture: tell the GPU about the texture once and re-use it!

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

   GLuint texture;
   bool   initialized;

   vtk441PolyDataMapper()
   {
     initialized = false;
   }
   void SetUpTexture()
   {
     glGenTextures(1, &texture);
     glBindTexture(GL_TEXTURE_2D, texture);

     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

     vtkJPEGReader *rdr = vtkJPEGReader::New();
     rdr->SetFileName("HankChilds_345.jpg");
     rdr->Update();
     vtkImageData *img = rdr->GetOutput();
     int dims[3];
     img->GetDimensions(dims);
     unsigned char *buffer = (unsigned char *) img->GetScalarPointer(0,0,0);
     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, dims[0], dims[1], 0, GL_RGB,
                  GL_UNSIGNED_BYTE, buffer);
     initialized = true;
   }
```

```
virtual void RenderPiece(vtkRenderer *ren, vtkActor *act
{
  if (!initialized)
     SetUpTexture();
  glEnable(GL_COLOR_MATERIAL);

  glBindTexture(GL_TEXTURE_2D, texture);
  glEnable(GL_TEXTURE_2D);
  float ambient[3] = { 1, 1, 1 };
  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
  glBegin(GL_TRIANGLES);
  glTexCoord2f(0,0);
  glVertex3f(0,0,0);
  glTexCoord2f(1, 0);
  glVertex3f(0,1,0);
  glTexCoord2f(1., 1.);
  glVertex3f(1,1,0);

  glTexCoord2f(1., 1.);
  glVertex3f(1.1,1,0);
  glTexCoord2f(0,1);
  glVertex3f(1.1,0,0);
  glTexCoord2f(0, 0);
  glVertex3f(0.1,0,0);

  glEnd();
}
};
```

More on Geometric Primitives

Geometric Specification: glBegin



Name

glBegin — delimit the vertices of a primitive or a group of like primitives

C Specification

`void` **`glBegin`**`(`GLenum *mode*`);`

Parameters

mode

> Specifies the primitive or primitives that will be created from vertices presented between `glBegin` and the subsequent glEnd. Ten symbolic constants are accepted: `GL_POINTS`, `GL_LINES`, `GL_LINE_STRIP`, `GL_LINE_LOOP`, `GL_TRIANGLES`, `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`, `GL_QUADS`, `GL_QUAD_STRIP`, and `GL_POLYGON`.

C Specification

`void` **`glEnd`**`(` *void*`);`

Description

`glBegin` and glEnd delimit the vertices that define a primitive or a group of like primitives. `glBegin` accepts a single argument that specifies in which of ten ways the vertices are interpreted. Taking n as an integer count starting at one, and N as the total number of vertices specified, the interpretations are as follows:

Geometric Primitives

`GL_POINTS`

Treats each vertex as a single point. Vertex n defines point n. N points are drawn.

`GL_LINES`

Treats each pair of vertices as an independent line segment. Vertices 2 ⊠ n - 1 and 2 ⊠ n define line n. N 2 lines are drawn.

`GL_LINE_STRIP`

Draws a connected group of line segments from the first vertex to the last. Vertices n and n + 1 define line n. N - 1 lines are drawn.

`GL_LINE_LOOP`

Draws a connected group of line segments from the first vertex to the last, then back to the first. Vertices n and n + 1 define line n. The last line, however, is defined by vertices N and 1 . N lines are drawn.

Geometric Primitives

`GL_TRIANGLES`

Treats each triplet of vertices as an independent triangle. Vertices 3 ☒ n - 2 , 3 ☒ n - 1 , and 3 ☒ n define triangle n. N 3 triangles are drawn.

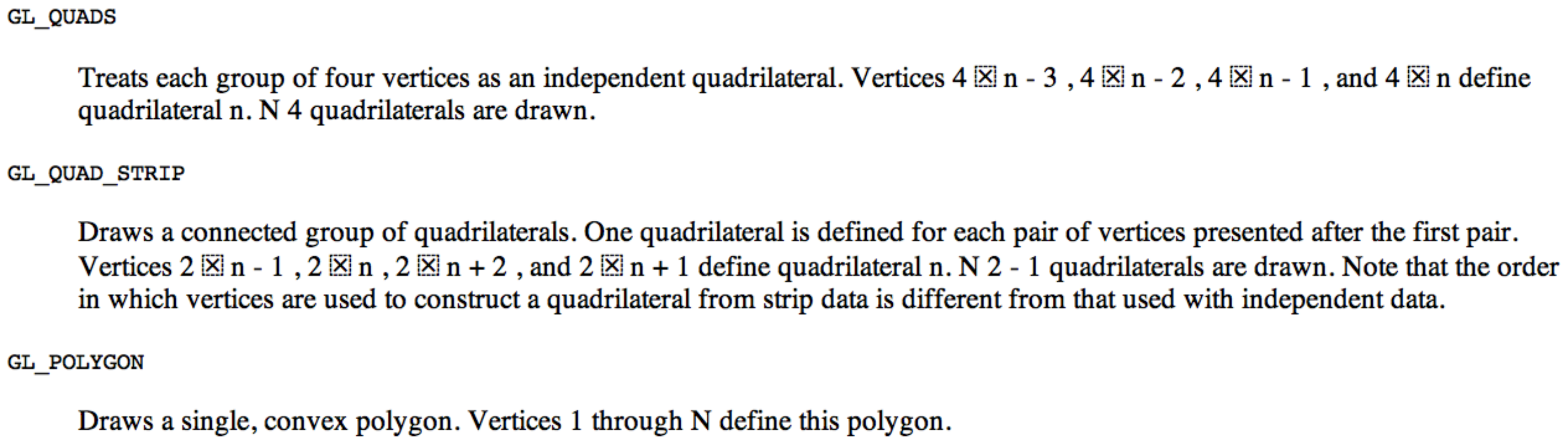
`GL_TRIANGLE_STRIP`

Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices. For odd n, vertices n, n + 1 , and n + 2 define triangle n. For even n, vertices n + 1 , n, and n + 2 define triangle n. N - 2 triangles are drawn.

`GL_TRIANGLE_FAN`

Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices. Vertices 1 , n + 1 , and n + 2 define triangle n. N - 2 triangles are drawn.

Geometric Primitives



```
GL_QUADS
```

Treats each group of four vertices as an independent quadrilateral. Vertices 4 ⊠ n - 3 , 4 ⊠ n - 2 , 4 ⊠ n - 1 , and 4 ⊠ n define quadrilateral n. N 4 quadrilaterals are drawn.

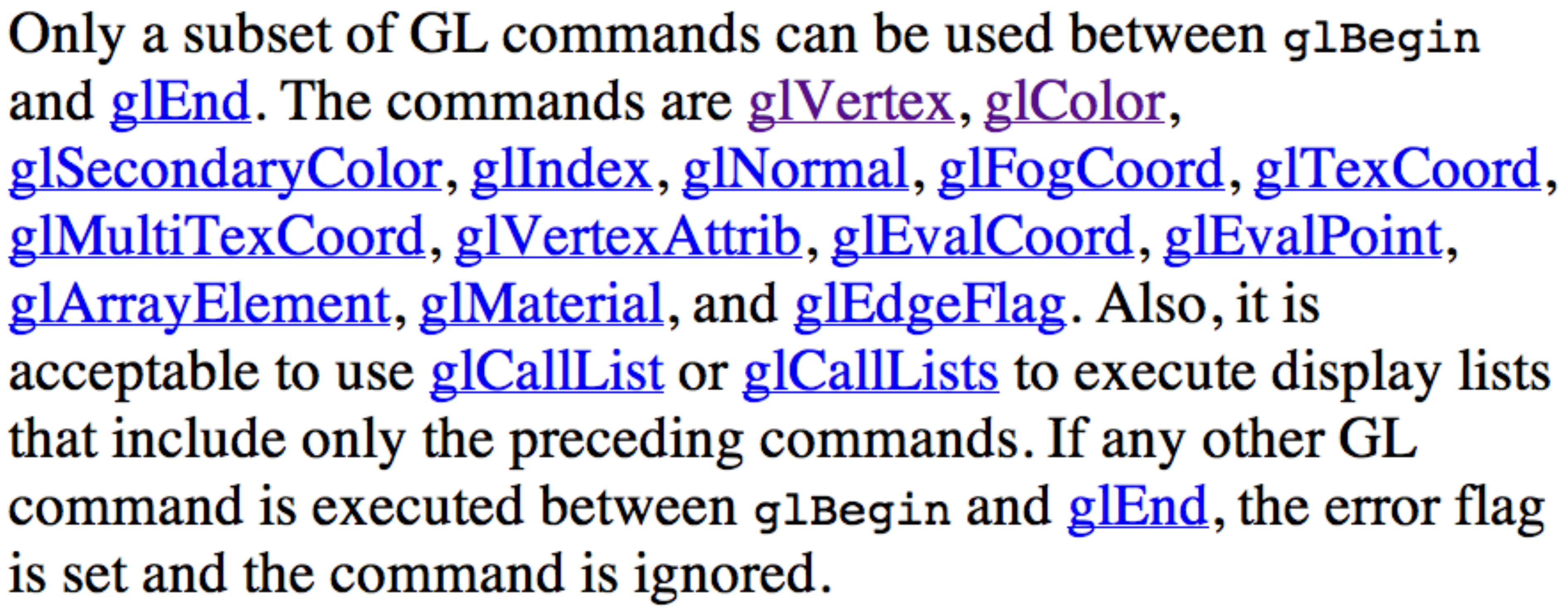
```
GL_QUAD_STRIP
```

Draws a connected group of quadrilaterals. One quadrilateral is defined for each pair of vertices presented after the first pair. Vertices 2 ⊠ n - 1 , 2 ⊠ n , 2 ⊠ n + 2 , and 2 ⊠ n + 1 define quadrilateral n. N 2 - 1 quadrilaterals are drawn. Note that the order in which vertices are used to construct a quadrilateral from strip data is different from that used with independent data.

```
GL_POLYGON
```

Draws a single, convex polygon. Vertices 1 through N define this polygon.

What can go inside a glBegin?



Only a subset of GL commands can be used between `glBegin` and glEnd. The commands are glVertex, glColor, glSecondaryColor, glIndex, glNormal, glFogCoord, glTexCoord, glMultiTexCoord, glVertexAttrib, glEvalCoord, glEvalPoint, glArrayElement, glMaterial, and glEdgeFlag. Also, it is acceptable to use glCallList or glCallLists to execute display lists that include only the preceding commands. If any other GL command is executed between `glBegin` and glEnd, the error flag is set and the command is ignored.

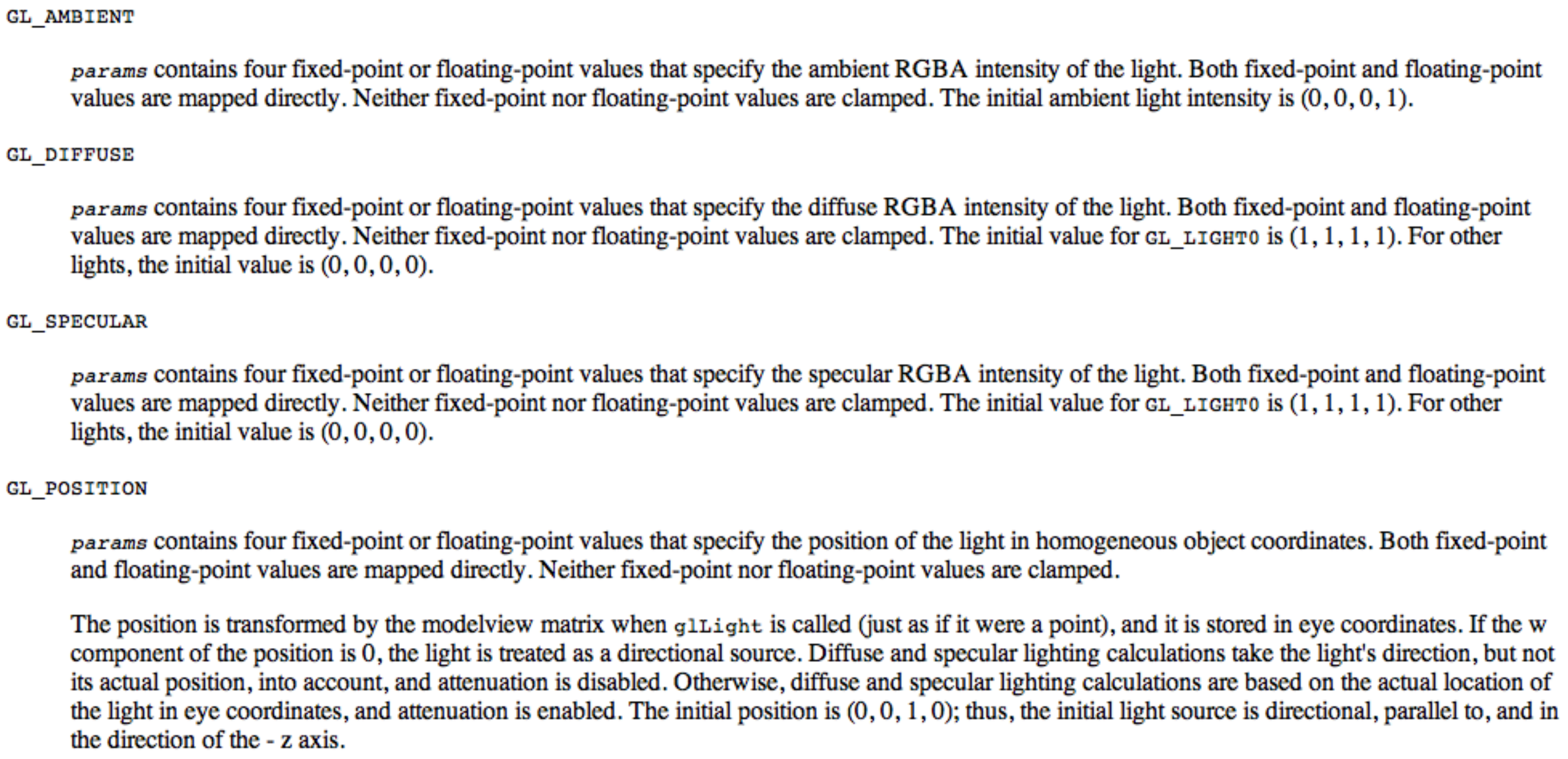
Lighting Model

Lighting

- glEnable(GL_LIGHTING);
  - Tells OpenGL you want to have lighting.
- Eight lights
  - Enable and disable individually
    - glEnable(GL_LIGHT0)
    - glDisable(GL_LIGHT7)
  - Set attributes individually
    - glLightfv(GL_LIGHTi, ARGUMENT, VALUES)

glLightfv parameters

- Ten parameters (ones you will use):



`GL_AMBIENT`

`params` contains four fixed-point or floating-point values that specify the ambient RGBA intensity of the light. Both fixed-point and floating-point values are mapped directly. Neither fixed-point nor floating-point values are clamped. The initial ambient light intensity is (0, 0, 0, 1).

`GL_DIFFUSE`

`params` contains four fixed-point or floating-point values that specify the diffuse RGBA intensity of the light. Both fixed-point and floating-point values are mapped directly. Neither fixed-point nor floating-point values are clamped. The initial value for `GL_LIGHT0` is (1, 1, 1, 1). For other lights, the initial value is (0, 0, 0, 0).

`GL_SPECULAR`

`params` contains four fixed-point or floating-point values that specify the specular RGBA intensity of the light. Both fixed-point and floating-point values are mapped directly. Neither fixed-point nor floating-point values are clamped. The initial value for `GL_LIGHT0` is (1, 1, 1, 1). For other lights, the initial value is (0, 0, 0, 0).

`GL_POSITION`

`params` contains four fixed-point or floating-point values that specify the position of the light in homogeneous object coordinates. Both fixed-point and floating-point values are mapped directly. Neither fixed-point nor floating-point values are clamped.

The position is transformed by the modelview matrix when `glLight` is called (just as if it were a point), and it is stored in eye coordinates. If the w component of the position is 0, the light is treated as a directional source. Diffuse and specular lighting calculations take the light's direction, but not its actual position, into account, and attenuation is disabled. Otherwise, diffuse and specular lighting calculations are based on the actual location of the light in eye coordinates, and attenuation is enabled. The initial position is (0, 0, 1, 0); thus, the initial light source is directional, parallel to, and in the direction of the - z axis.

glLightfv in action

For each light source, we can set an RGBA for the <u>diffuse, specular, and ambient components</u>:

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
Glfloat diffuse0[4] = { 0.7, 0.7, 0.7, 1 };
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
… // set ambient, specular, position
glDisable(GL_LIGHT1); // do we need to do this?
…
glDisable(GL_LIGHT7); // do we need to do this?
```

How do we tell OpenGL about the surface normals?

- Flat shading:

```
glNormal3f(0, 0.707, -0.707);
glVertex3f(0, 0, 0);
glVertex3f(1, 1, 0);
glVertex3f(1, 0, 0);
```

- Smooth shading:

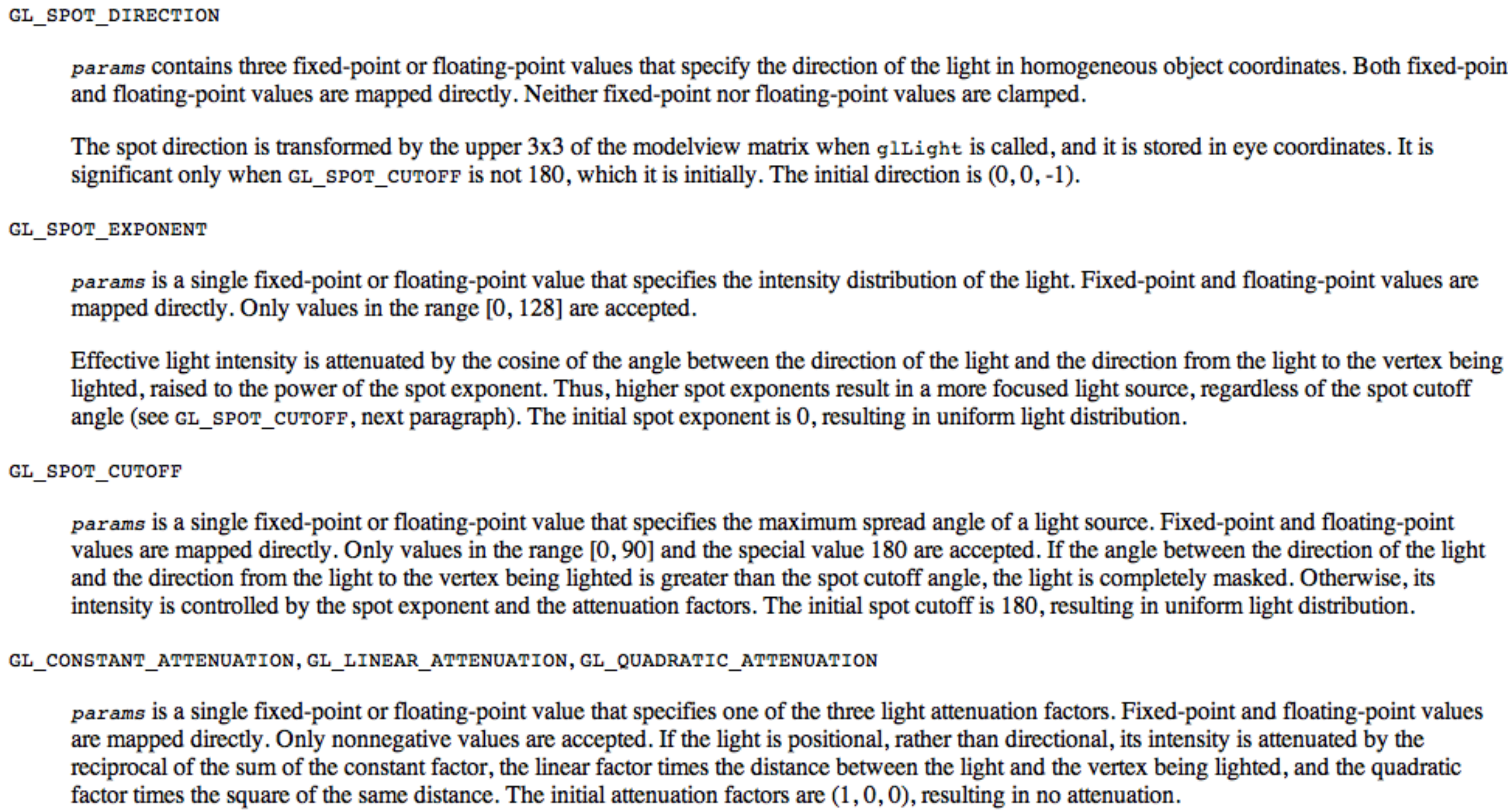
```
glNormal3f(0, 0.707, -0.707);
glVertex3f(0, 0, 0);
glNormal3f(0, 0.707, +0.707);
glVertex3f(1, 1, 0);
glNormal3f(1, 0, 0);
glVertex3f(1, 0, 0);
```

Distance and Direction

- The source colors are specified in RGBA
- The position is given in homogeneous coordinates
  - If w =1.0, we are specifying a finite location
  - If w =0.0, we are specifying a parallel source with the given direction vector

glLightfv parameters (2)

- Ten parameters (ones you will never use)



`GL_SPOT_DIRECTION`

`params` contains three fixed-point or floating-point values that specify the direction of the light in homogeneous object coordinates. Both fixed-poin and floating-point values are mapped directly. Neither fixed-point nor floating-point values are clamped.

The spot direction is transformed by the upper 3x3 of the modelview matrix when `glLight` is called, and it is stored in eye coordinates. It is significant only when `GL_SPOT_CUTOFF` is not 180, which it is initially. The initial direction is (0, 0, -1).

`GL_SPOT_EXPONENT`

`params` is a single fixed-point or floating-point value that specifies the intensity distribution of the light. Fixed-point and floating-point values are mapped directly. Only values in the range [0, 128] are accepted.

Effective light intensity is attenuated by the cosine of the angle between the direction of the light and the direction from the light to the vertex being lighted, raised to the power of the spot exponent. Thus, higher spot exponents result in a more focused light source, regardless of the spot cutoff angle (see `GL_SPOT_CUTOFF`, next paragraph). The initial spot exponent is 0, resulting in uniform light distribution.

`GL_SPOT_CUTOFF`

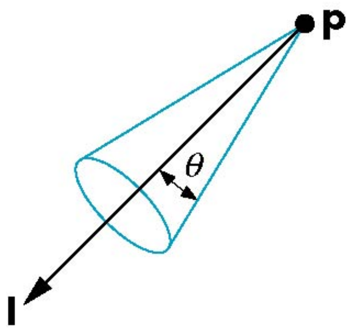
`params` is a single fixed-point or floating-point value that specifies the maximum spread angle of a light source. Fixed-point and floating-point values are mapped directly. Only values in the range [0, 90] and the special value 180 are accepted. If the angle between the direction of the light and the direction from the light to the vertex being lighted is greater than the spot cutoff angle, the light is completely masked. Otherwise, its intensity is controlled by the spot exponent and the attenuation factors. The initial spot cutoff is 180, resulting in uniform light distribution.

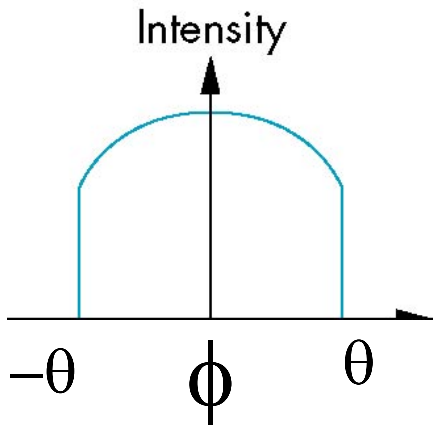
`GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, GL_QUADRATIC_ATTENUATION`

`params` is a single fixed-point or floating-point value that specifies one of the three light attenuation factors. Fixed-point and floating-point values are mapped directly. Only nonnegative values are accepted. If the light is positional, rather than directional, its intensity is attenuated by the reciprocal of the sum of the constant factor, the linear factor times the distance between the light and the vertex being lighted, and the quadratic factor times the square of the same distance. The initial attenuation factors are (1, 0, 0), resulting in no attenuation.

Spotlights

- Use `glLightv` to set
  - Direction `GL_SPOT_DIRECTION`
  - Cutoff `GL_SPOT_CUTOFF`
  - Attenuation `GL_SPOT_EXPONENT`
    - Proportional to $\cos^{\alpha}\phi$





What happens with multiple lights?

```
glEnable(GL_LIGHT0);
glEnable(GL_LIGHT1);
```

- → the effects of these lights are additive.
  - Individual shading factors are added and combined
  - Effect is to make objects brighter and brighter
    - Same as handling of high specular factors for 1E

Global Ambient Light

- Ambient light depends on color of light sources
  - A red light in a white room will cause a red ambient term that disappears when the light is turned off
- OpenGL also allows a global ambient term that is often helpful for testing
  - `glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient)`
    - **VTK turns this on by default!**
      - **Affects lighting of materials colored with glColor, but not glTexCoord1f!!**

Shading Model

glShadeModel

glShadeModel — select flat or smooth shading

C Specification

`void glShadeModel(GLenum mode);`

Parameters

mode

Specifies a symbolic value representing a shading technique. Accepted values are `GL_FLAT` and `GL_SMOOTH`. The initial value is `GL_SMOOTH`.

Description

GL primitives can have either flat or smooth shading. Smooth shading, the default, causes the computed colors of vertices to be interpolated as the primitive is rasterized, typically assigning different colors to each resulting pixel fragment. Flat shading selects the computed color of just one vertex and assigns it to all the pixel fragments generated by rasterizing a single primitive. In either case, the computed color of a vertex is the result of lighting if lighting is enabled, or it is the current color at the time the vertex was specified if lighting is disabled.

Flat and smooth shading are indistinguishable for points. Starting when glBegin is issued and counting vertices and primitives from 1, the GL gives each flat-shaded line segment i the computed color of vertex i + 1 , its second vertex. Counting similarly from 1, the GL gives each flat-shaded polygon the computed color of the vertex listed in the following table. This is the last vertex to specify the polygon in all cases except single polygons, where the first vertex specifies the flat-shaded color.

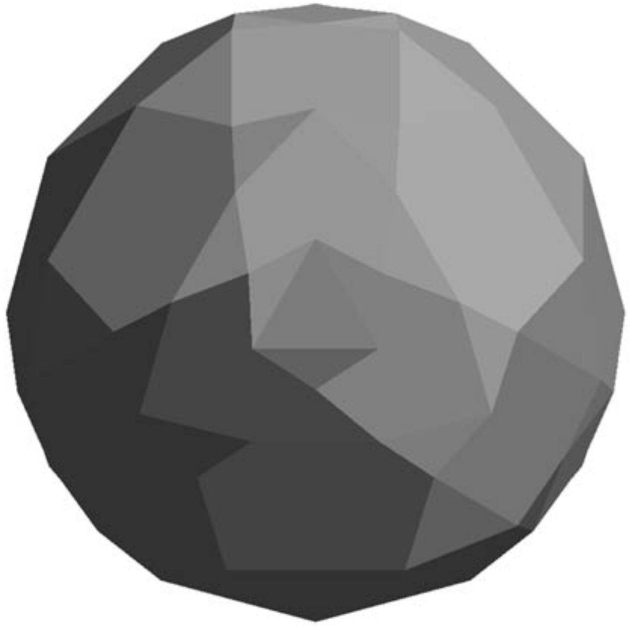
Primitive Type of Polygon i	**Vertex**
Single polygon (i == 1)	1
Triangle strip	i + 2
Triangle fan	i + 2
Independent triangle	3 ☒ i
Quad strip	2 ☒ i + 2
Independent quad	4 ☒ i

Polygonal Shading

- Shading calculations are done for each vertex
  - Vertex colors become vertex shades
- By default, vertex shades are interpolated across the polygon
  - `glShadeModel(GL_SMOOTH);`
- If we use `glShadeModel(GL_FLAT);` the color at the first vertex will determine the shade of the whole polygon
  - We will come back to this in a few slides

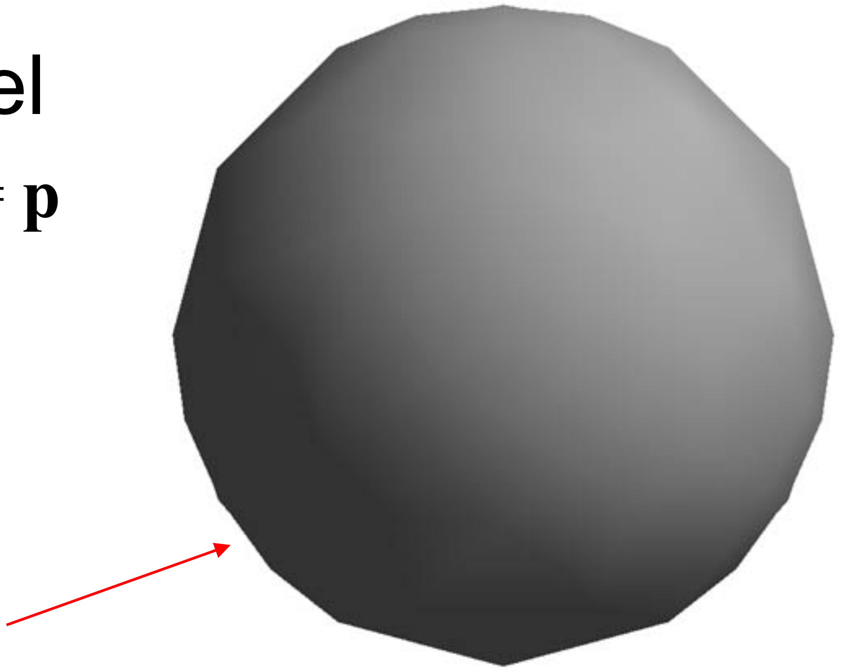
Flat Shading

- *IF* polygons have a single normal
  - Shades at the vertices as computed by the Phong model can be almost same
  - Identical for a distant viewer (default) or if there is no specular component
- Example: model of sphere



Smooth Shading

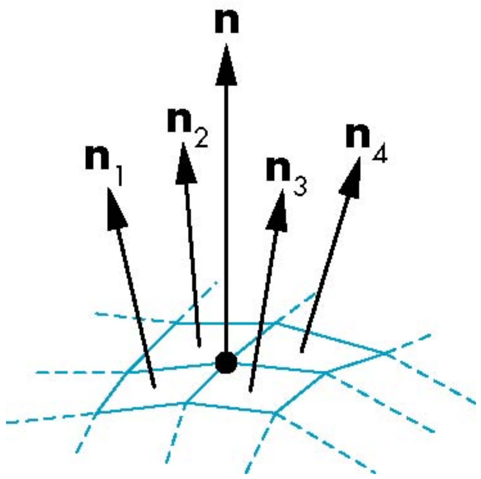
- Normal at each vertex
- Easy for sphere model
  - If centered at origin $\mathbf{n} = \mathbf{p}$
- Now smooth shading works
- Note *silhouette edge*



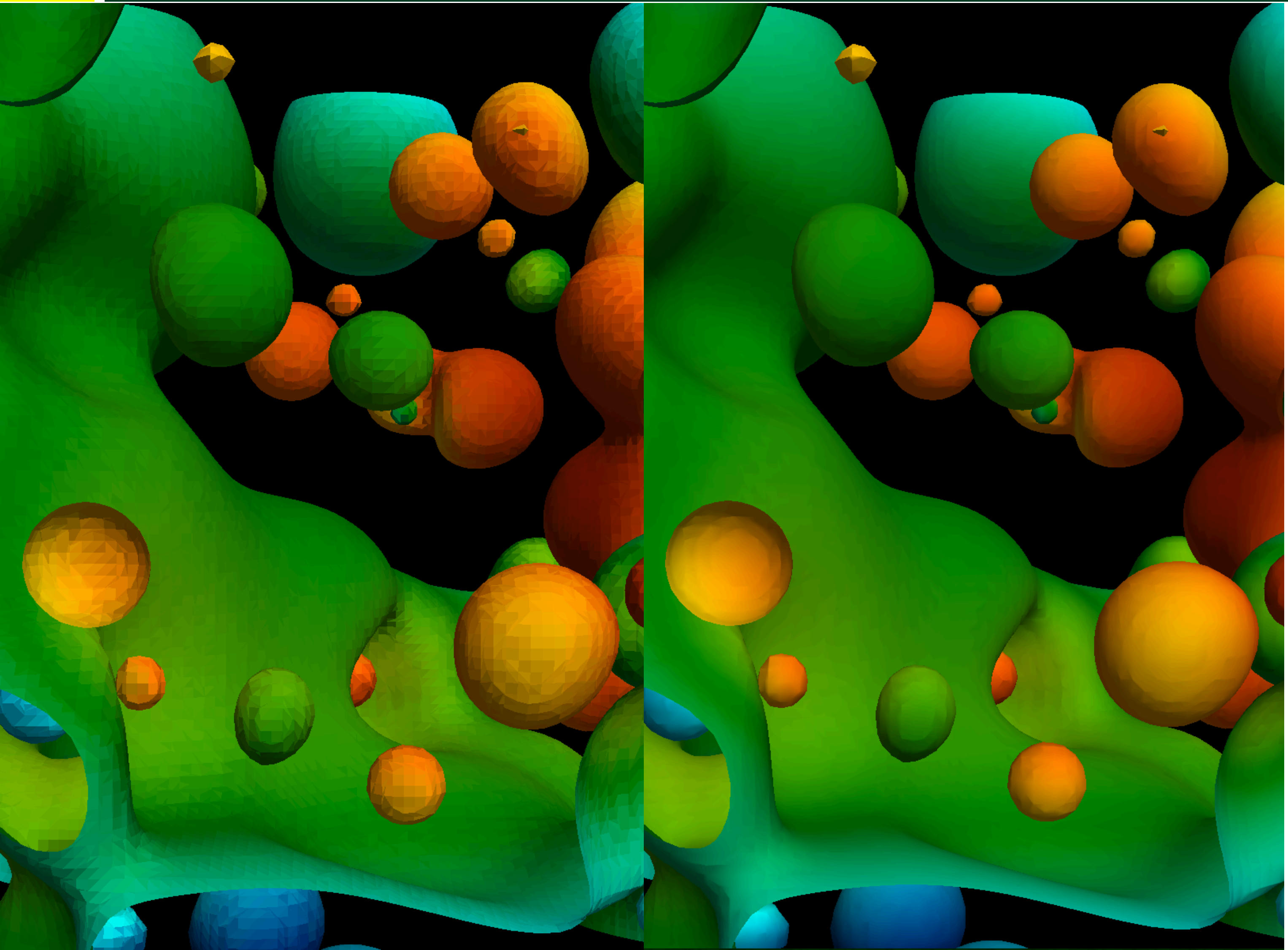
Mesh Shading

- The previous example is not general because we knew the normal at each vertex analytically
- For polygonal models, Gouraud proposed we use the average of the normals around a mesh vertex

$$\mathbf{n} = (\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4)/\ |\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4|$$



glShadeModel



glShadeModel affects normals and colors

Material Model

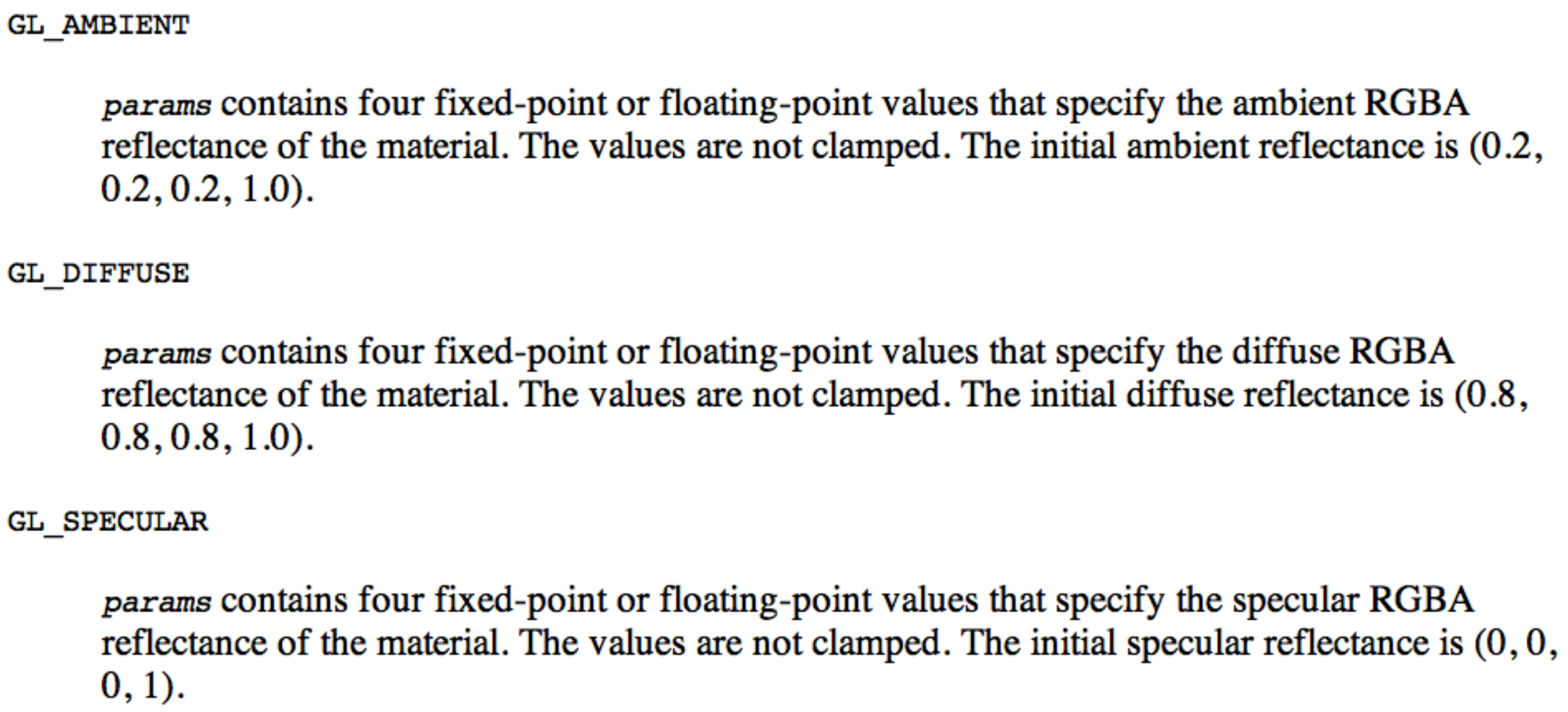
glMaterial: coarser controls for color

- You specify how much light is reflected for the material type.
- Command:

  glMaterialfv(FACE_TYPE, PARAMETER, VALUE(S))
- FACE_TYPE =
  - GL_FRONT_AND_BACK
  - ~~GL_FRONT~~
  - ~~GL_BACK~~

(We will talk about this later)

glMaterialfv Parameters



GL_AMBIENT

params contains four fixed-point or floating-point values that specify the ambient RGBA reflectance of the material. The values are not clamped. The initial ambient reflectance is (0.2, 0.2, 0.2, 1.0).

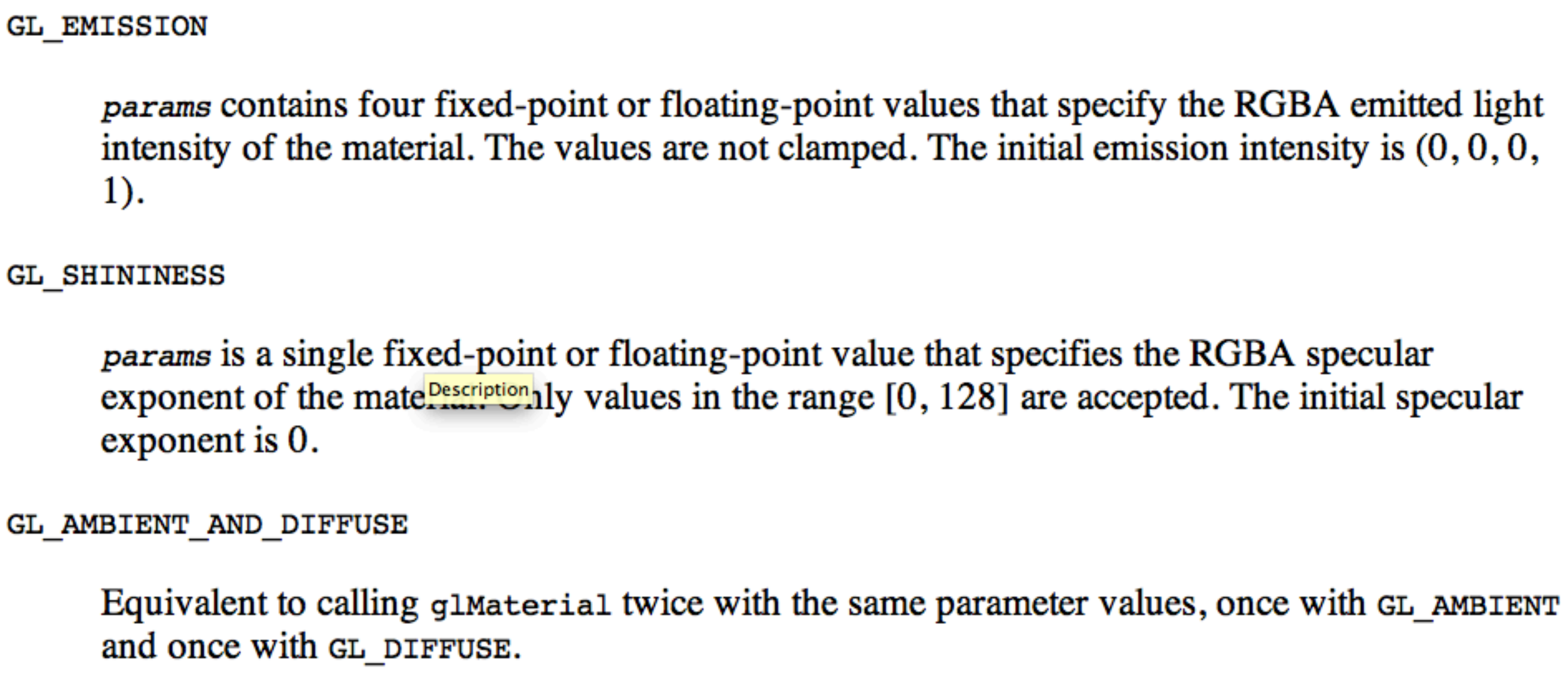
GL_DIFFUSE

params contains four fixed-point or floating-point values that specify the diffuse RGBA reflectance of the material. The values are not clamped. The initial diffuse reflectance is (0.8, 0.8, 0.8, 1.0).

GL_SPECULAR

params contains four fixed-point or floating-point values that specify the specular RGBA reflectance of the material. The values are not clamped. The initial specular reflectance is (0, 0, 0, 1).

glMaterialfv Parameters



`GL_EMISSION`

params contains four fixed-point or floating-point values that specify the RGBA emitted light intensity of the material. The values are not clamped. The initial emission intensity is (0, 0, 0, 1).

`GL_SHININESS`

params is a single fixed-point or floating-point value that specifies the RGBA specular exponent of the mate Description nly values in the range [0, 128] are accepted. The initial specular exponent is 0.

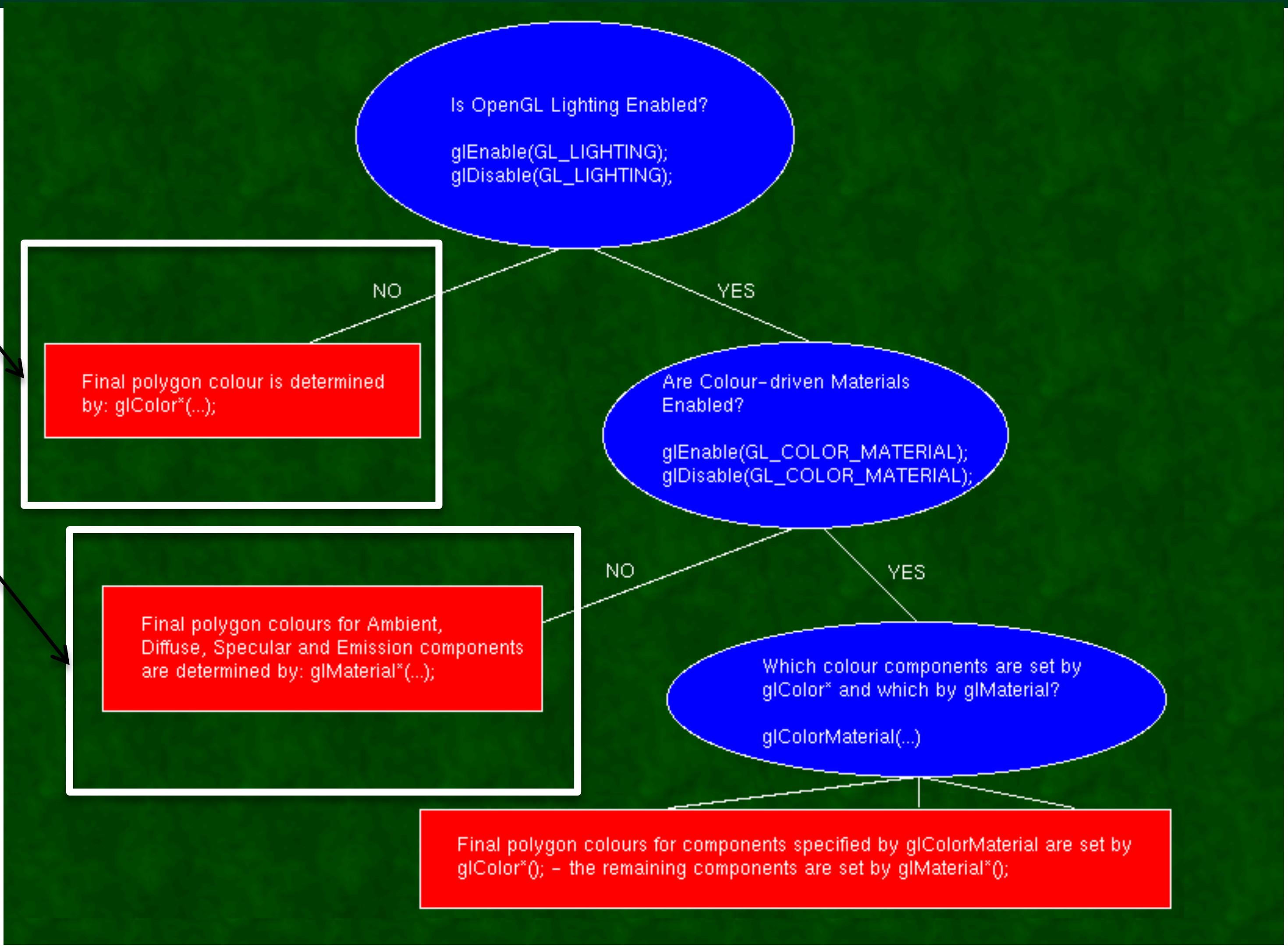
`GL_AMBIENT_AND_DIFFUSE`

Equivalent to calling `glMaterial` twice with the same parameter values, once with `GL_AMBIENT` and once with `GL_DIFFUSE`.

OpenGL: very complex model for lighting and colors

glMaterial not used

glColor is no-op


Is OpenGL Lighting Enabled?
glEnable(GL_LIGHTING);
glDisable(GL_LIGHTING);
NO
YES
Final polygon colour is determined by: glColor*(...);
Are Colour-driven Materials Enabled?
glEnable(GL_COLOR_MATERIAL);
glDisable(GL_COLOR_MATERIAL);
NO
YES
Final polygon colours for Ambient, Diffuse, Specular and Emission components are determined by: glMaterial*(...);
Which colour components are set by glColor* and which by glMaterial?
glColorMaterial(...)
Final polygon colours for components specified by glColorMaterial are set by glColor*(); - the remaining components are set by glMaterial*();

http://www.sjbaker.org/steve/omniv/opengl_lighting.html

Basic OpenGL light model

- The OpenGL light model presumes that the light that reaches your eye from the polygon surface arrives by four different mechanisms:
  - AMBIENT
  - DIFFUSE
  - SPECULAR
  - EMISSION - in this case, the light is actually emitted by the polygon - equally in all directions.

http://www.sjbaker.org/steve/omniv/opengl_lighting.html

Difference between lights and materials

- There are three light colors for each light:
  - Ambient, Diffuse and Specular (set with glLight)
- There are four colors for each surface:
  - Same three + Emission(set with glMaterial)
- All OpenGL implementations support at least eight light sources - and the glMaterial can be changed at will for each polygon

http://www.sjbaker.org/steve/omniv/opengl_lighting.html

Interactions between lights and materials

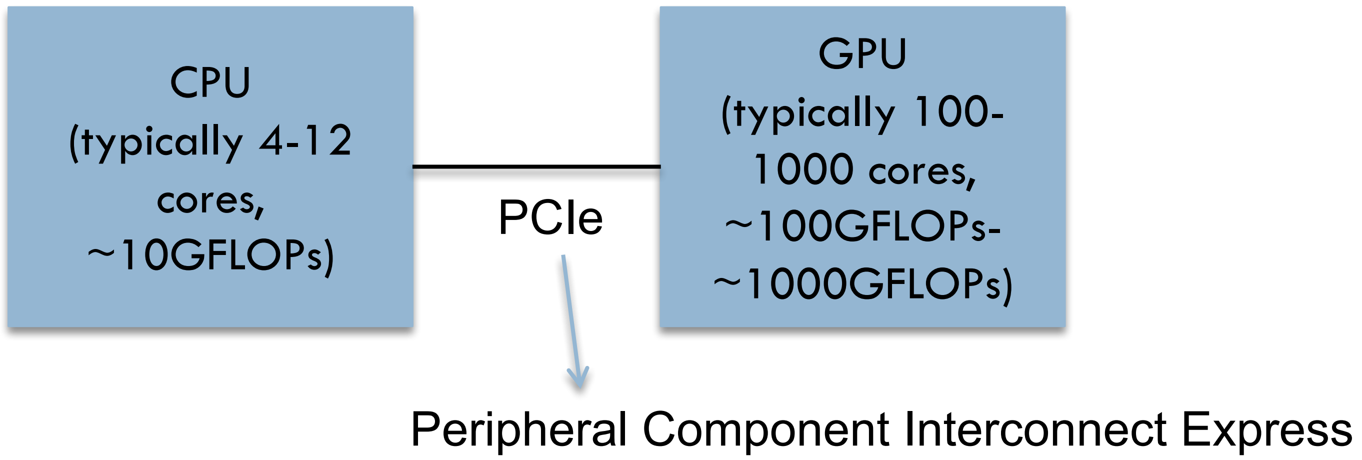
- The final polygon colour is the sum of all four light components, each of which is formed by multiplying the glMaterial colour by the glLight colour (modified by the directionality in the case of Diffuse and Specular).
- Since there is no Emission colour for the glLight, that is added to the final colour without modification.

http://www.sjbaker.org/steve/omniv/opengl_lighting.html

Display Lists

CPU and GPU

- Most common configuration has CPU and GPU on separate dies
  - I.e., plug GPU in CPU


CPU (typically 4-12 cores, ~10GFLOPs)
PCIe
GPU (typically 100-1000 cores, ~100GFLOPs-~1000GFLOPs)
Peripheral Component Interconnect Express

What are the performance ramifications of this architecture?

Display lists

- Idea:
  - send geometry and settings to GPU once, give it an identifier
  - GPU stores geometry and settings
  - Just pass the identifier for every subsequent render

Display lists

- Generate an idenfitier:

```
GLUint displayList = glGenLists(1);
```

- Tell GPU that all subsequent geometry is part of the list:

```
glNewList(displayList,GL_COMPILE);
```

- Specify geometry (i.e., glVertex, etc)
- Tell GPU we are done specifying geometry:

```
glEndList();
```

- Later on, tell GPU to render all the geometry and settings associated with our list:

```
glCallList(displayList);
```

Display lists in action

```
for (int frame = 0 ; frame < nFrames ; frame++)
{
   SetCamera(frame, nFrames);
   glBegin(GL_TRIANGLES);
   for (int i = 0 ; i < triangles.size() ; i++)
   {
       for (int j = 0 ; j < 3 ; j++)
       {
           glColor3ubv(triangles[i].colors[j]);
           glColor3fv(triangles[i].vertices[j]);
   }
   glEnd();
}
```

```
GLUint displayList = glGenLists(1);
glNewList(displayList, GL_COMPILE);
glBegin(GL_TRIANGLES);
for (int i = 0 ; i < triangles.size() ; i++)
{
    for (int j = 0 ; j < 3 ; j++)
    {
        glColor3ubv(triangles[i].colors[j]);
        glColor3fv(triangles[i].vertices[j]);
}
glEnd();
glEndList();

for (int frame = 0 ; frame < nFrames ; frame++)
{
   SetCamera(frame, nFrames);
   glCallList(displayList);
}
```

What are the performance ramifications between the two?

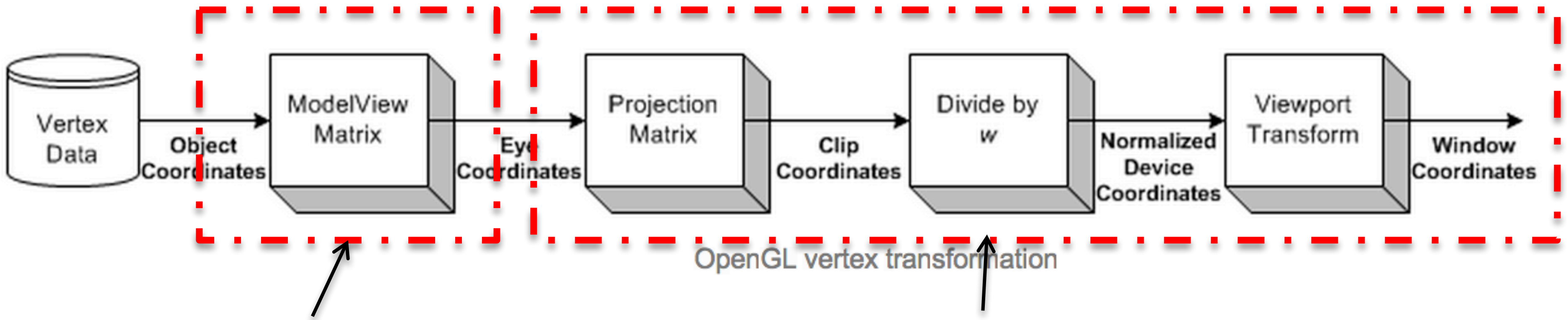
glNewList

- GL_COMPILE
  - Make the display list for later use.
- GL_COMPILE_AND_EXECUTE
  - Make the display list and also execute it as you go.

Why is My 2A Rendering Slow?

Transforms in GL

ModelView and Projection Matrices



New for us

Familiar for us

- ModelView idea: two purposes … model and view
  - Model: extra matrix, just for rotating, scaling, and translating geometry.
    - How could this be useful?
  - View: Cartesian to Camera transform
- (We will focus on the model part of the modelview matrix now & come back to others later)

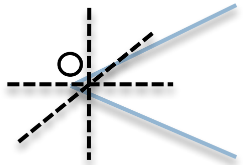
SLIDE REPEAT: Our goal

Add additional transforms here….

World space:

Triangles in native Cartesian coordinates

Camera located anywhere



Camera space:

Camera located at origin, looking down -Z

Triangle coordinates relative to camera frame

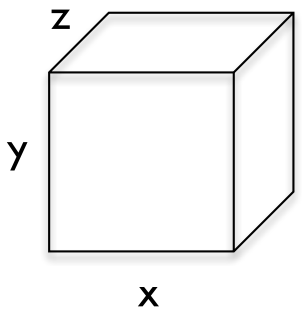
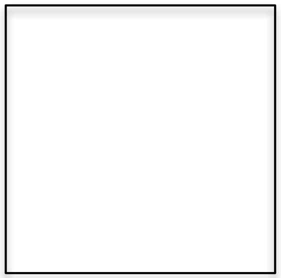


Image space:

All viewable objects within

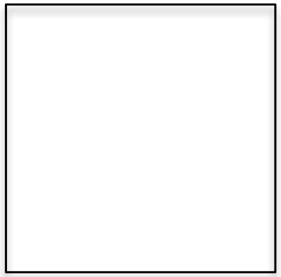
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



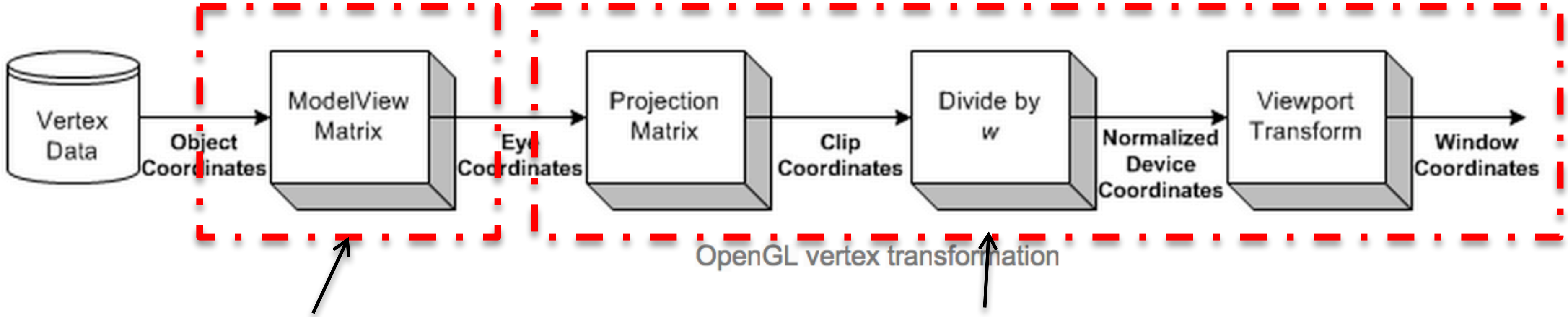
Device space:

All viewable objects within

0<=x<=width, 0

<=y<=height

ModelView and Projection Matrices



New for us

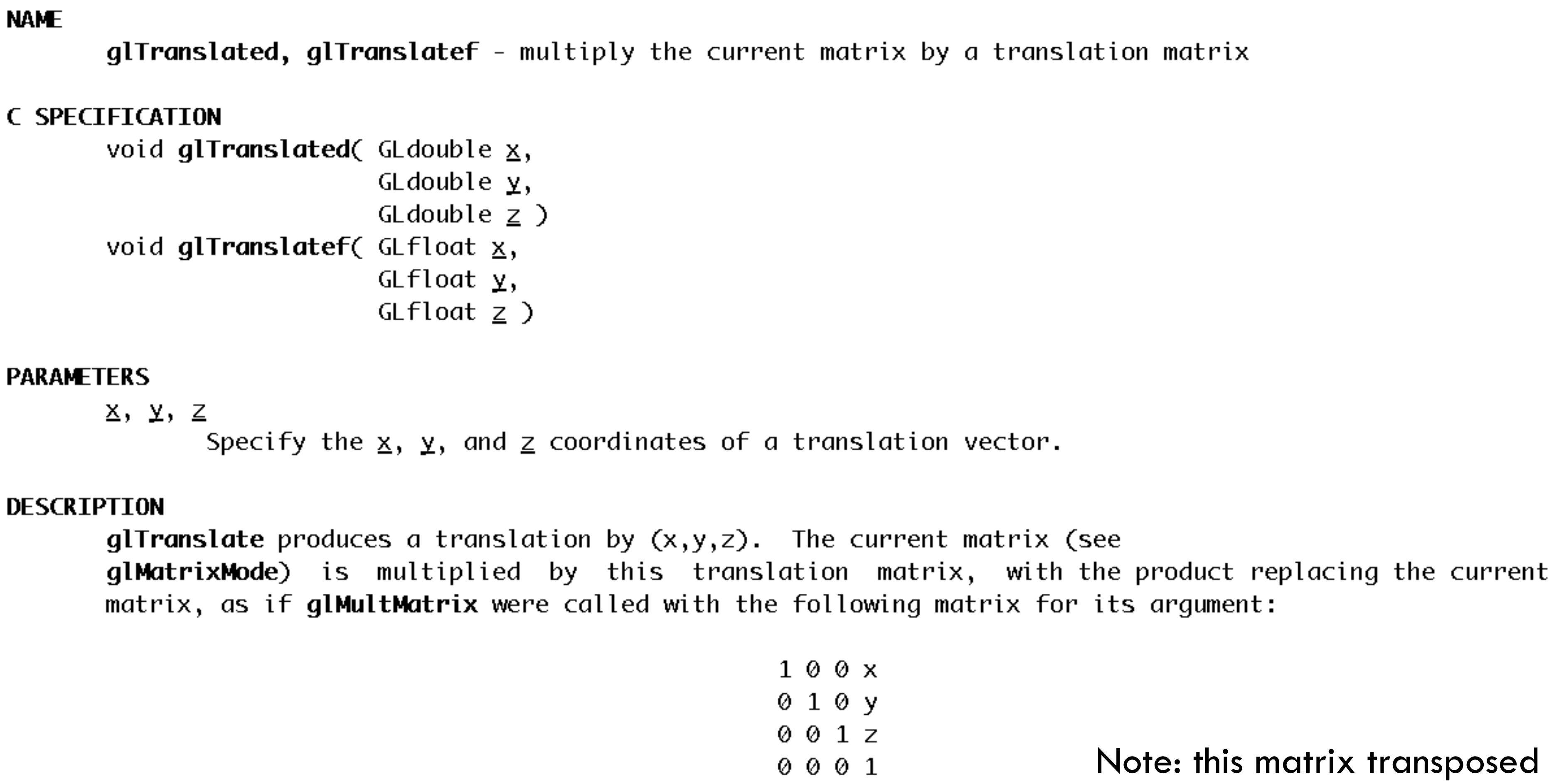
Familiar for us

- ModelView idea: two purposes … model and view
  - Model: extra matrix, just for rotating, scaling, and translating geometry.
    - How could this be useful?
  - View: Cartesian to Camera transform
- (We will focus on the model part of the modelview matrix now & come back to others later)

Common commands for modifying model part of ModelView matrix

- glTranslate
- glRotate
- glScale

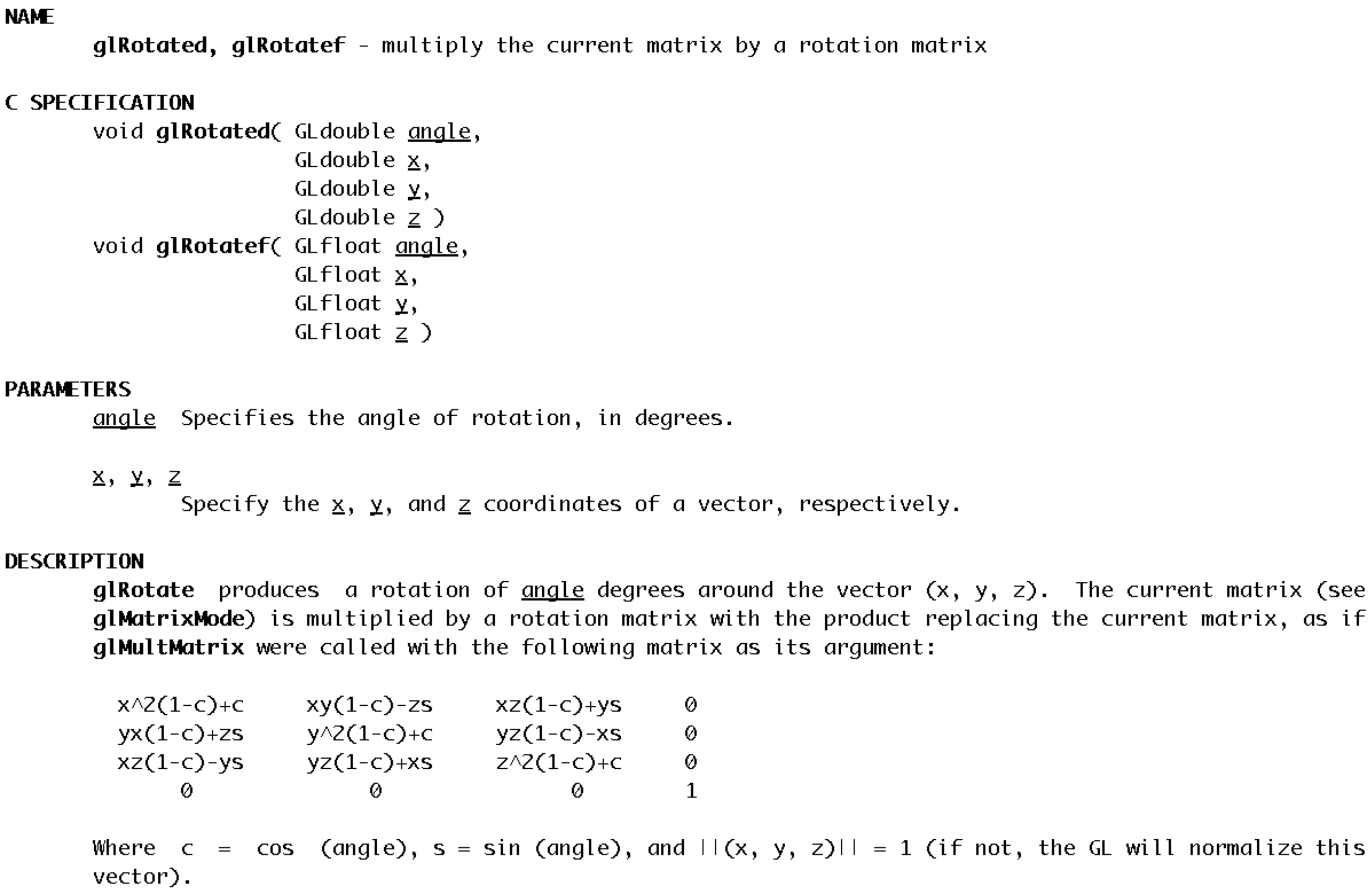
glTranslate



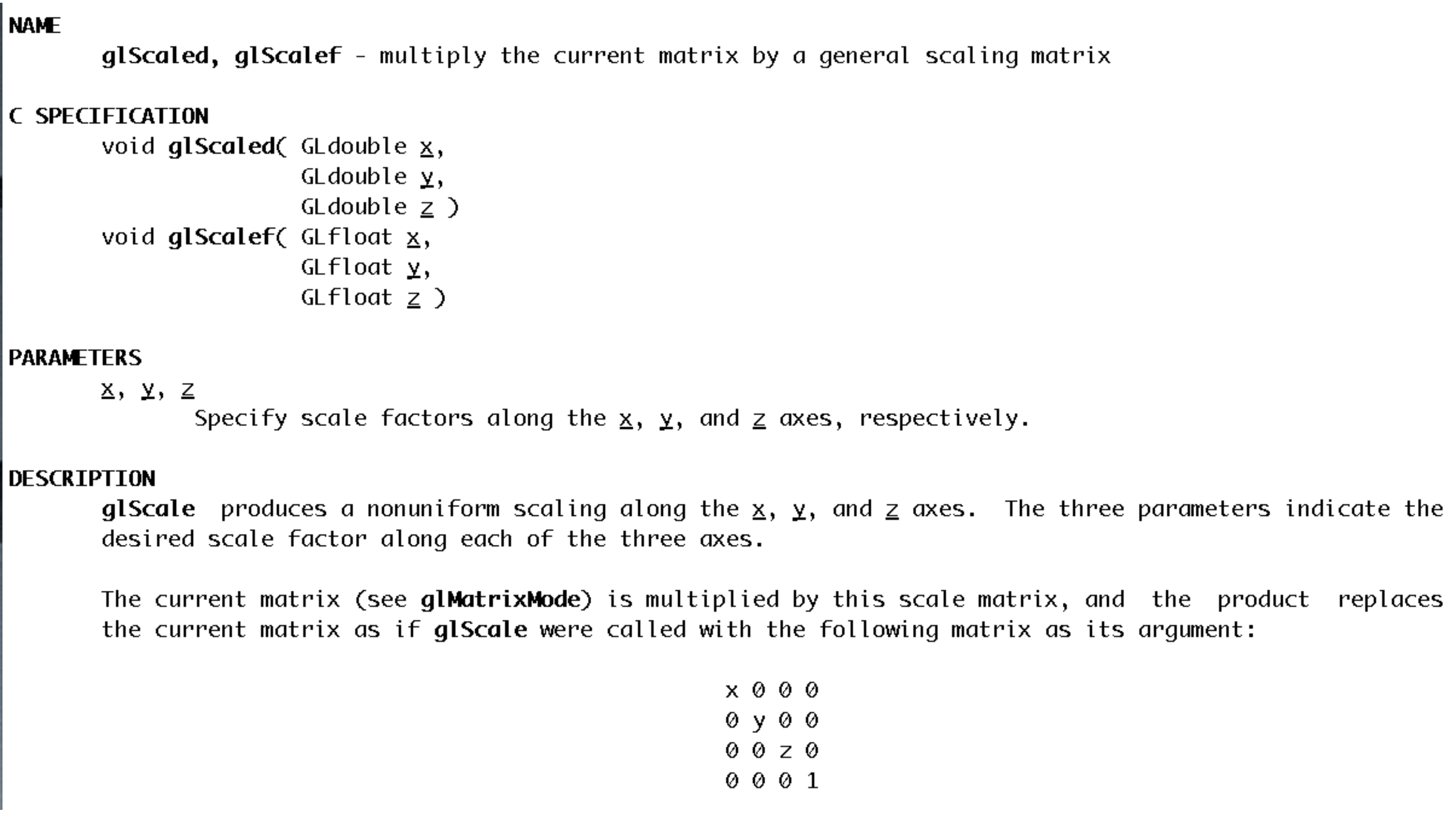
```
NAME
       glTranslated, glTranslatef - multiply the current matrix by a translation matrix

C SPECIFICATION
       void glTranslated( GLdouble x,
                          GLdouble y,
                          GLdouble z )
       void glTranslatef( GLfloat x,
                          GLfloat y,
                          GLfloat z )

PARAMETERS
       x, y, z
              Specify the x, y, and z coordinates of a translation vector.

DESCRIPTION
       glTranslate produces a translation by (x,y,z).  The current matrix (see
       glMatrixMode)  is  multiplied  by  this  translation  matrix,  with the product replacing the current
       matrix, as if glMultMatrix were called with the following matrix for its argument:

                                                 1 0 0 x
                                                 0 1 0 y
                                                 0 0 1 z
                                                 0 0 0 1
```

Note: this matrix transposed from what we did earlier

glRotate


```
NAME
       glRotated, glRotatef - multiply the current matrix by a rotation matrix

C SPECIFICATION
       void glRotated( GLdouble angle,
                       GLdouble x,
                       GLdouble y,
                       GLdouble z )
       void glRotatef( GLfloat angle,
                       GLfloat x,
                       GLfloat y,
                       GLfloat z )

PARAMETERS
       angle  Specifies the angle of rotation, in degrees.

       x, y, z
              Specify the x, y, and z coordinates of a vector, respectively.

DESCRIPTION
       glRotate  produces  a rotation of angle degrees around the vector (x, y, z).  The current matrix (see
       glMatrixMode) is multiplied by a rotation matrix with the product replacing the current matrix, as if
       glMultMatrix were called with the following matrix as its argument:

         x^2(1-c)+c     xy(1-c)-zs     xz(1-c)+ys     0
         yx(1-c)+zs     y^2(1-c)+c     yz(1-c)-xs     0
         xz(1-c)-ys     yz(1-c)+xs     z^2(1-c)+c     0
              0              0               0        1

       Where  c  =  cos  (angle), s = sin (angle), and ||(x, y, z)|| = 1 (if not, the GL will normalize this
       vector).
```

glScale



```
NAME
       glScaled, glScalef - multiply the current matrix by a general scaling matrix

C SPECIFICATION
       void glScaled( GLdouble x,
                      GLdouble y,
                      GLdouble z )
       void glScalef( GLfloat x,
                      GLfloat y,
                      GLfloat z )

PARAMETERS
       x, y, z
              Specify scale factors along the x, y, and z axes, respectively.

DESCRIPTION
       glScale  produces a nonuniform scaling along the x, y, and z axes.  The three parameters indicate the
       desired scale factor along each of the three axes.

       The current matrix (see glMatrixMode) is multiplied by this scale matrix, and  the  product  replaces
       the current matrix as if glScale were called with the following matrix as its argument:

                                              x 0 0 0
                                              0 y 0 0
                                              0 0 z 0
                                              0 0 0 1
```

How do transformations combine?

```
glScale(2, 2, 2)
glTranslate(1, 0, 0)
glRotate(45, 0, 1, 0)
```

→ Rotate by 45 degrees around (0,1,0), then translate in X by 1, then scale by 2 in all dimensions.

→ (the last transformation is applied first)

Which of two of these three are the same?

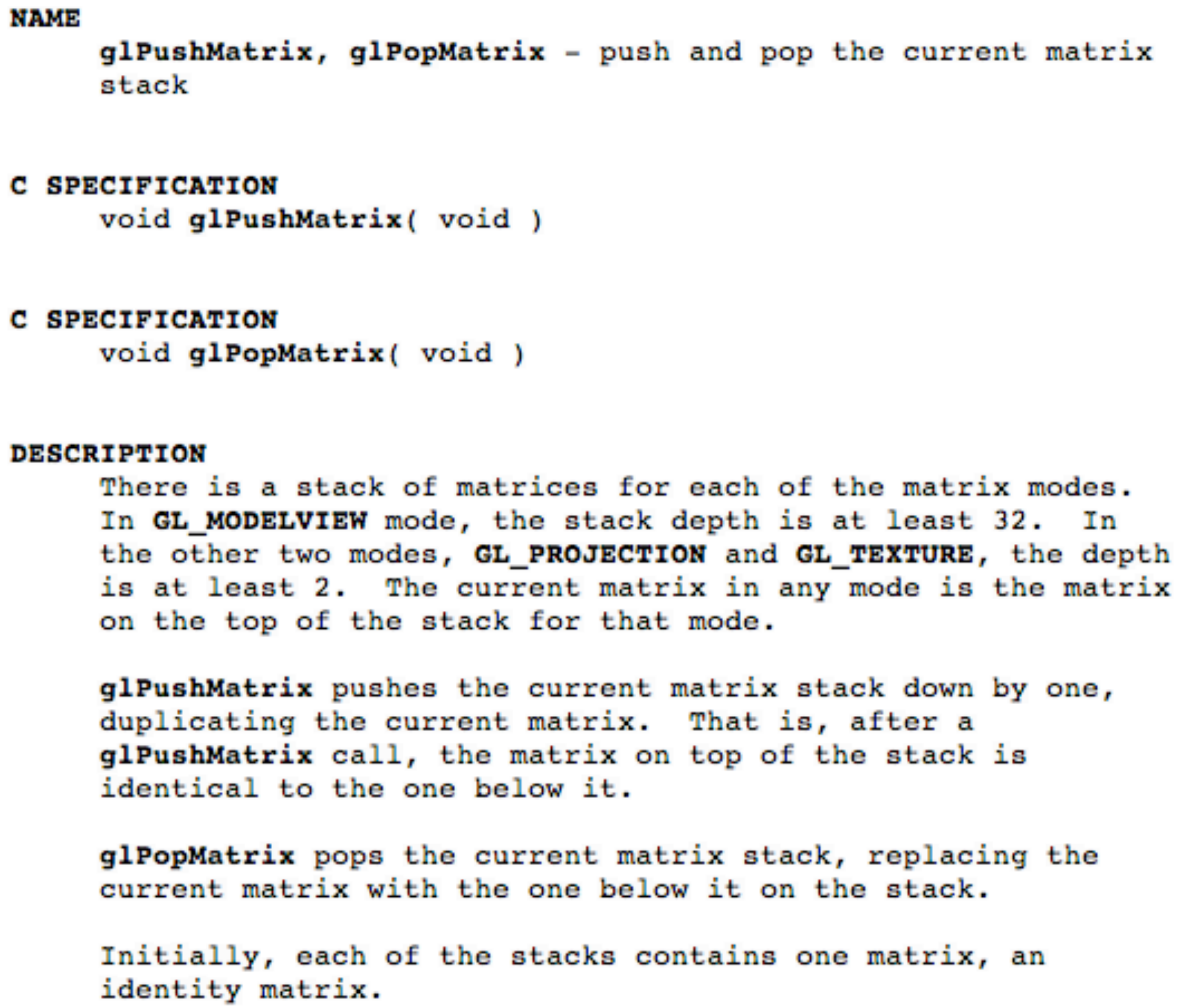
- Choice A:
  - glScalef(2, 2, 2);
  - glTranslate(1, 0, 0);
- Choice B:
  - glTranslate(1, 0, 0);
  - glScalef(2, 2, 2);
- Choice C:
  - glTranslate(2, 0, 0);
  - glScalef(2, 2, 2);

ModelView usage

```
dl = GenerateTireGeometry();
glCallList(dl);  // place tire at (0, 0, 0)
glTranslatef(10, 0, 0);
glCallList(dl); // place tire at (10, 0, 0)
glTranslatef(0, 0, 10)
glCallList(dl); // pl
glTranslatef(-10, 0,
glCallList(dl);  // pl
```

Each glTranslatef call updates the state of the ModelView matrix.

glPushMatrix, glPopMatrix



NAME

glPushMatrix, glPopMatrix - push and pop the current matrix stack

C SPECIFICATION

void **glPushMatrix**(void)

C SPECIFICATION

void **glPopMatrix**(void)

DESCRIPTION

There is a stack of matrices for each of the matrix modes. In **GL_MODELVIEW** mode, the stack depth is at least 32. In the other two modes, **GL_PROJECTION** and **GL_TEXTURE**, the depth is at least 2. The current matrix in any mode is the matrix on the top of the stack for that mode.

glPushMatrix pushes the current matrix stack down by one, duplicating the current matrix. That is, after a **glPushMatrix** call, the matrix on top of the stack is identical to the one below it.

glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack.

Initially, each of the stacks contains one matrix, an identity matrix.

glPushMatrix and glPopMatrix

```
dl = GenerateTireGeometry();
glCallList(dl);  // place tire at (0, 0, 0)
glPushMatrix();
glTranslatef(10, 0, 0);
glCallList(dl); // place tire at (10, 0, 0)
glPopMatrix();
glPushMatrix();
glTranslatef(0, 0, 10);
glCallList(dl); // place tire at ~~(10, 0, 10)~~  (0, 0, 10)
glPopMatrix();
```

Why is this useful?

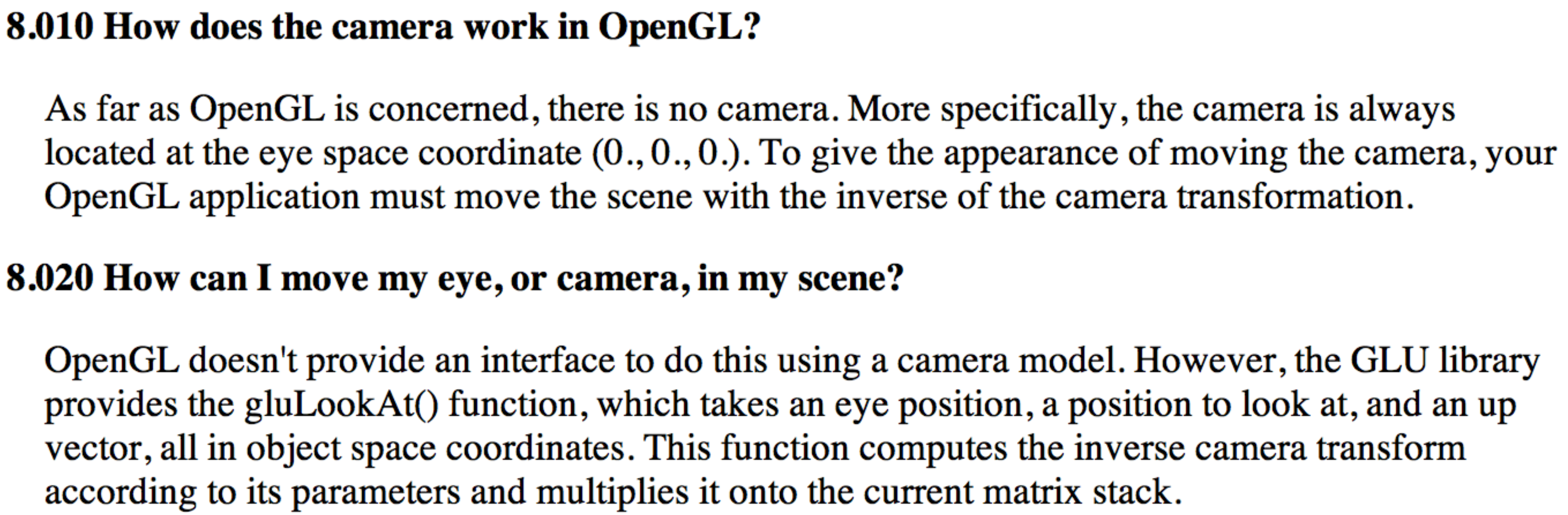
Matrices in OpenGL

- OpenGL maintains matrices for you and provides functions for setting matrices.
- There are four different modes you can use:
  - Modelview
  - Projection
  - Texture
  - Color (rarely used, often not supported)
- You control the mode using glMatrixMode.

Matrices in OpenGL (cont'd)

- The matrices are the identity matrix by default and you can modify them by:
  - 1) setting the matrix explicitly
  - 2) using OpenGL commands for appending to the matrix
- You can have >= 32 matrices for modelview, >=2 for others

The Camera Transformation



8.010 How does the camera work in OpenGL?

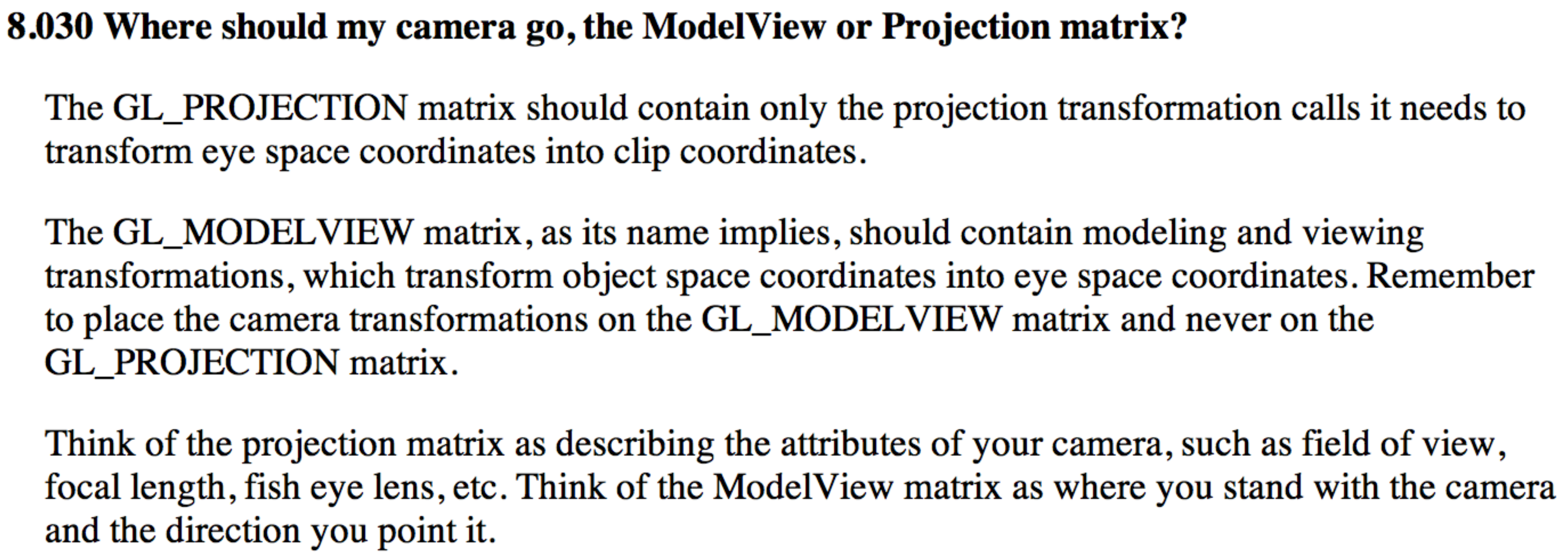
As far as OpenGL is concerned, there is no camera. More specifically, the camera is always located at the eye space coordinate (0., 0., 0.). To give the appearance of moving the camera, your OpenGL application must move the scene with the inverse of the camera transformation.

8.020 How can I move my eye, or camera, in my scene?

OpenGL doesn't provide an interface to do this using a camera model. However, the GLU library provides the gluLookAt() function, which takes an eye position, a position to look at, and an up vector, all in object space coordinates. This function computes the inverse camera transform according to its parameters and multiplies it onto the current matrix stack.

Source: www.opengl.org/archives/resources/faq/technical/viewing.htm

The Camera Transformation



8.030 Where should my camera go, the ModelView or Projection matrix?

The GL_PROJECTION matrix should contain only the projection transformation calls it needs to transform eye space coordinates into clip coordinates.

The GL_MODELVIEW matrix, as its name implies, should contain modeling and viewing transformations, which transform object space coordinates into eye space coordinates. Remember to place the camera transformations on the GL_MODELVIEW matrix and never on the GL_PROJECTION matrix.

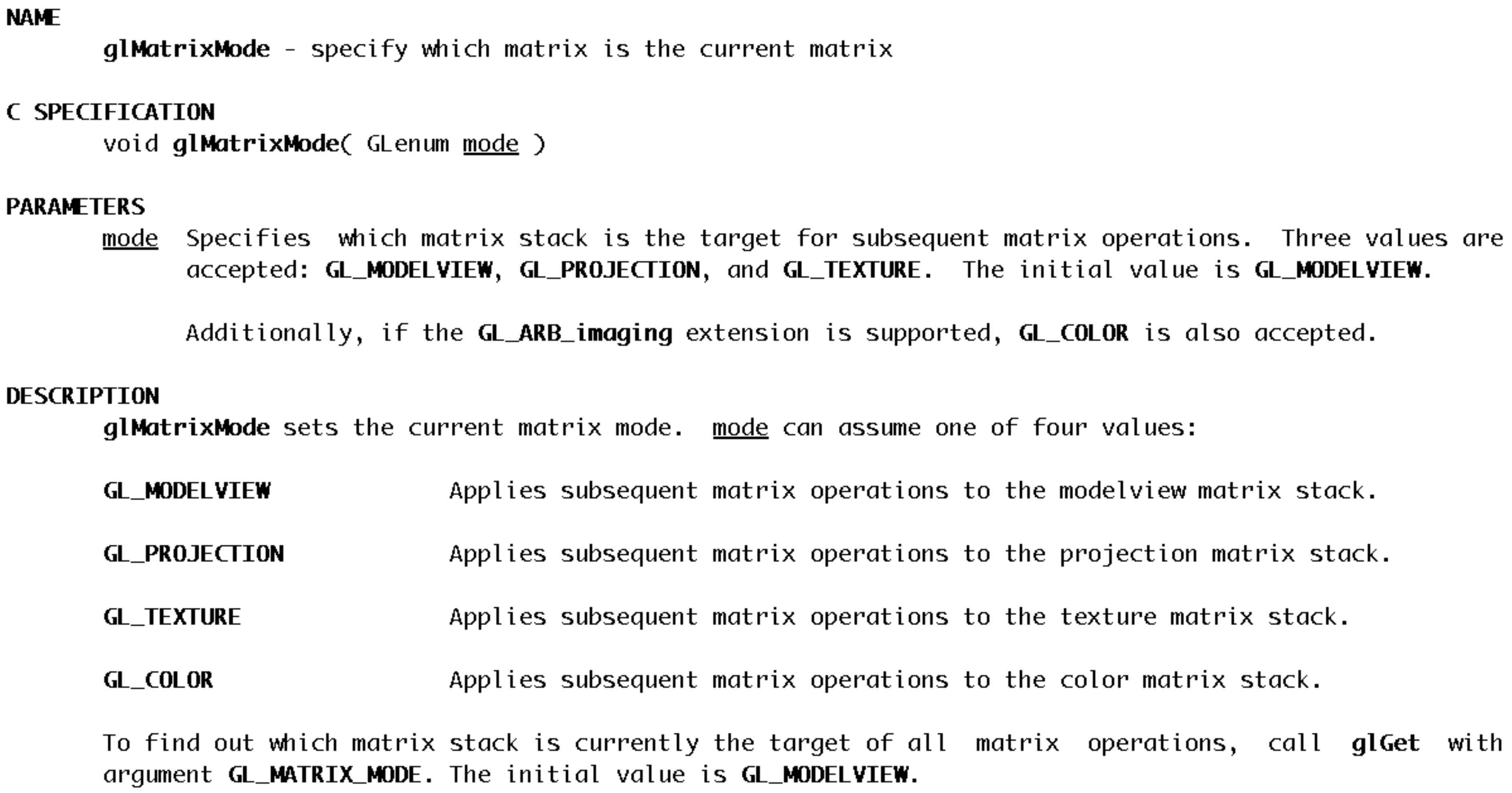
Think of the projection matrix as describing the attributes of your camera, such as field of view, focal length, fish eye lens, etc. Think of the ModelView matrix as where you stand with the camera and the direction you point it.

Source: www.opengl.org/archives/resources/faq/technical/viewing.htm

How do you put the Camera Transform in the ModelView matrix?

- No single GL call.
- Options are:
  - (1) you do it yourself (i.e., calculate matrix and load it into OpenGL)
  - (2) you use somebody's code, i.e., gluLookAt
  - (3) you use a combination of glRotatef, glScalef, and glTranslatef commands.

glMatrixMode



NAME

glMatrixMode - specify which matrix is the current matrix

C SPECIFICATION

void **glMatrixMode**(GLenum mode)

PARAMETERS

mode Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: **GL_MODELVIEW**, **GL_PROJECTION**, and **GL_TEXTURE**. The initial value is **GL_MODELVIEW.**

Additionally, if the **GL_ARB_imaging** extension is supported, **GL_COLOR** is also accepted.

DESCRIPTION

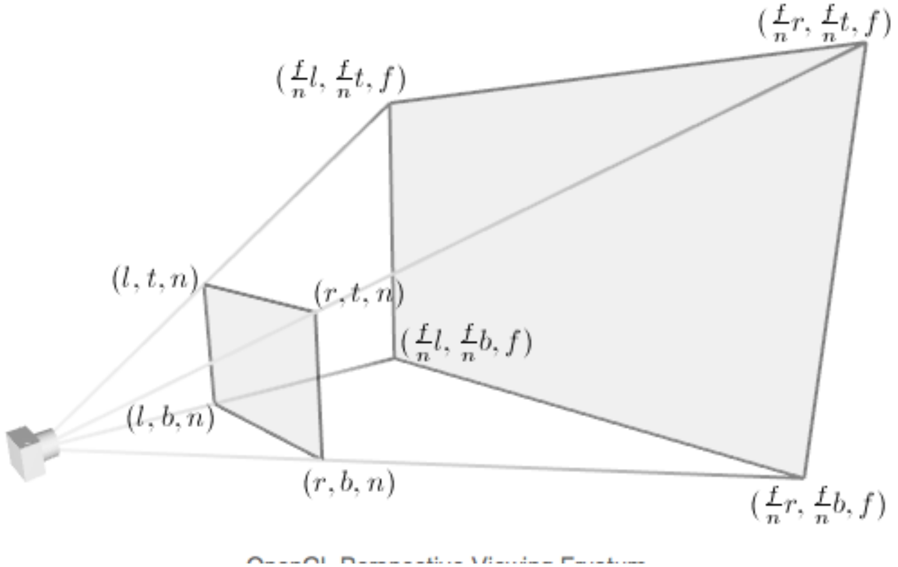
glMatrixMode sets the current matrix mode. mode can assume one of four values:

GL_MODELVIEW	Applies subsequent matrix operations to the modelview matrix stack.
GL_PROJECTION	Applies subsequent matrix operations to the projection matrix stack.
GL_TEXTURE	Applies subsequent matrix operations to the texture matrix stack.
GL_COLOR	Applies subsequent matrix operations to the color matrix stack.

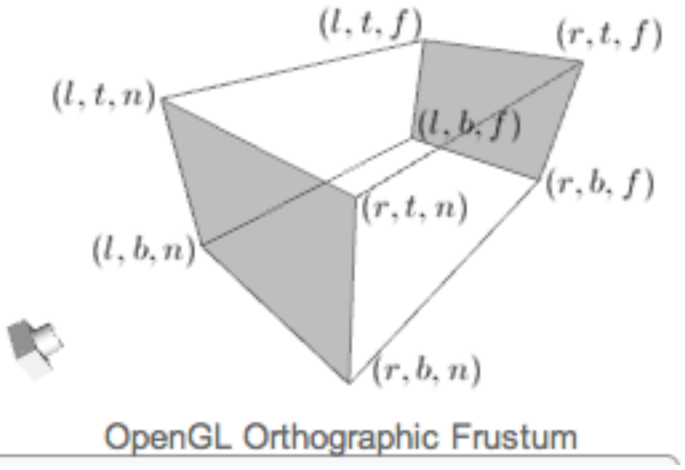
To find out which matrix stack is currently the target of all matrix operations, call **glGet** with argument **GL_MATRIX_MODE**. The initial value is **GL_MODELVIEW.**

How do you put the projection transformation in GL_PROJECTION?

- Two options:
  - glFrustum() (perspective projection)
  - glOrtho() (orthographic projection)

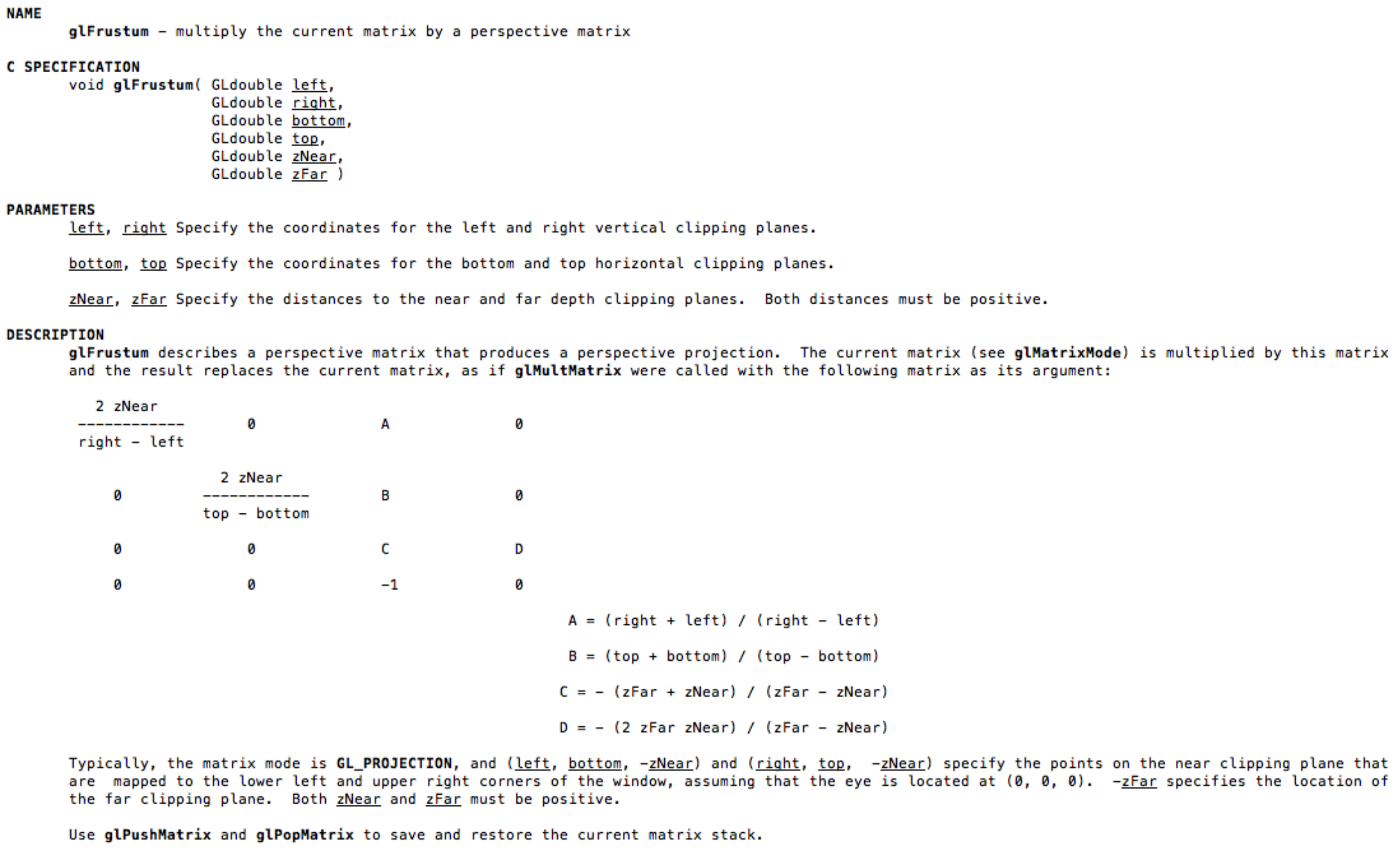


$(\frac{f}{n}r, \frac{f}{n}t, f)$
$(\frac{f}{n}l, \frac{f}{n}t, f)$
(l, t, n)
(r, t, n)
$(\frac{f}{n}l, \frac{f}{n}b, f)$
(l, b, n)
(r, b, n)
$(\frac{f}{n}r, \frac{f}{n}b, f)$



OpenGL Orthographic Frustum

glFrustum

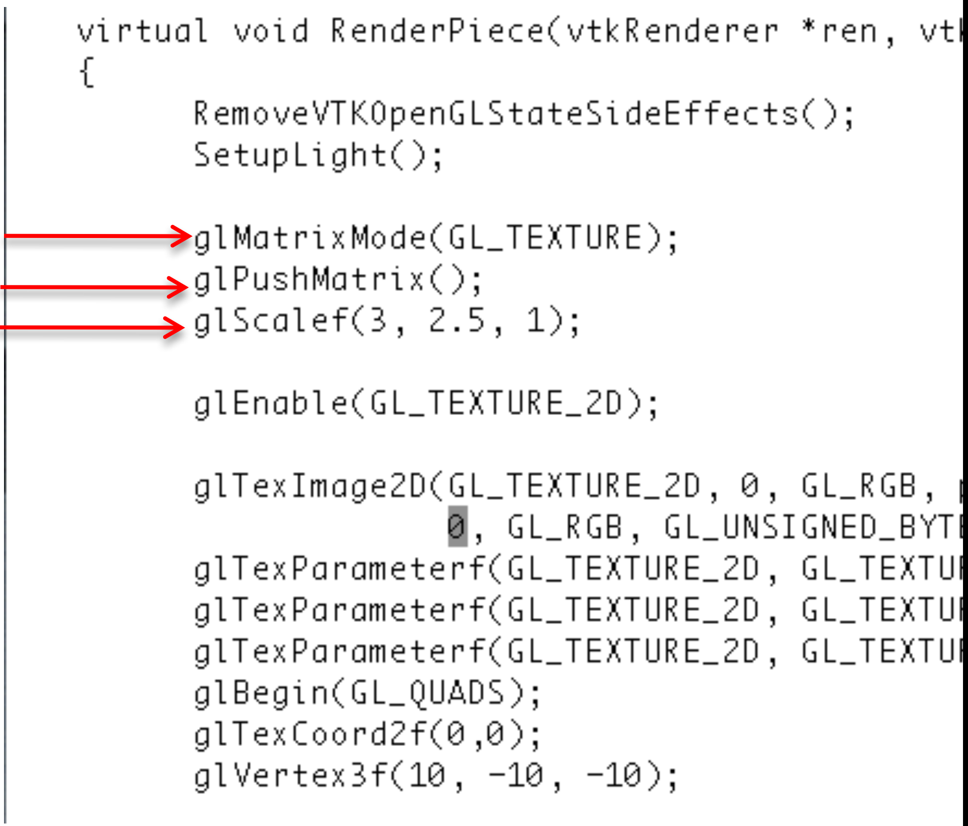


```
NAME
       glFrustum - multiply the current matrix by a perspective matrix

C SPECIFICATION
       void glFrustum( GLdouble left,
                       GLdouble right,
                       GLdouble bottom,
                       GLdouble top,
                       GLdouble zNear,
                       GLdouble zFar )

PARAMETERS
       left, right Specify the coordinates for the left and right vertical clipping planes.

       bottom, top Specify the coordinates for the bottom and top horizontal clipping planes.

       zNear, zFar Specify the distances to the near and far depth clipping planes.  Both distances must be positive.

DESCRIPTION
       glFrustum describes a perspective matrix that produces a perspective projection.  The current matrix (see glMatrixMode) is multiplied by this matrix
       and the result replaces the current matrix, as if glMultMatrix were called with the following matrix as its argument:

          2 zNear
        ------------       0              A              0
        right - left

                         2 zNear
             0         ------------       B              0
                       top - bottom

             0             0              C              D

             0             0              -1             0

                                                             A = (right + left) / (right - left)

                                                             B = (top + bottom) / (top - bottom)

                                                            C = - (zFar + zNear) / (zFar - zNear)

                                                            D = - (2 zFar zNear) / (zFar - zNear)

       Typically, the matrix mode is GL_PROJECTION, and (left, bottom, -zNear) and (right, top,  -zNear) specify the points on the near clipping plane that
       are  mapped to the lower left and upper right corners of the window, assuming that the eye is located at (0, 0, 0).  -zFar specifies the location of
       the far clipping plane.  Both zNear and zFar must be positive.

       Use glPushMatrix and glPopMatrix to save and restore the current matrix stack.
```

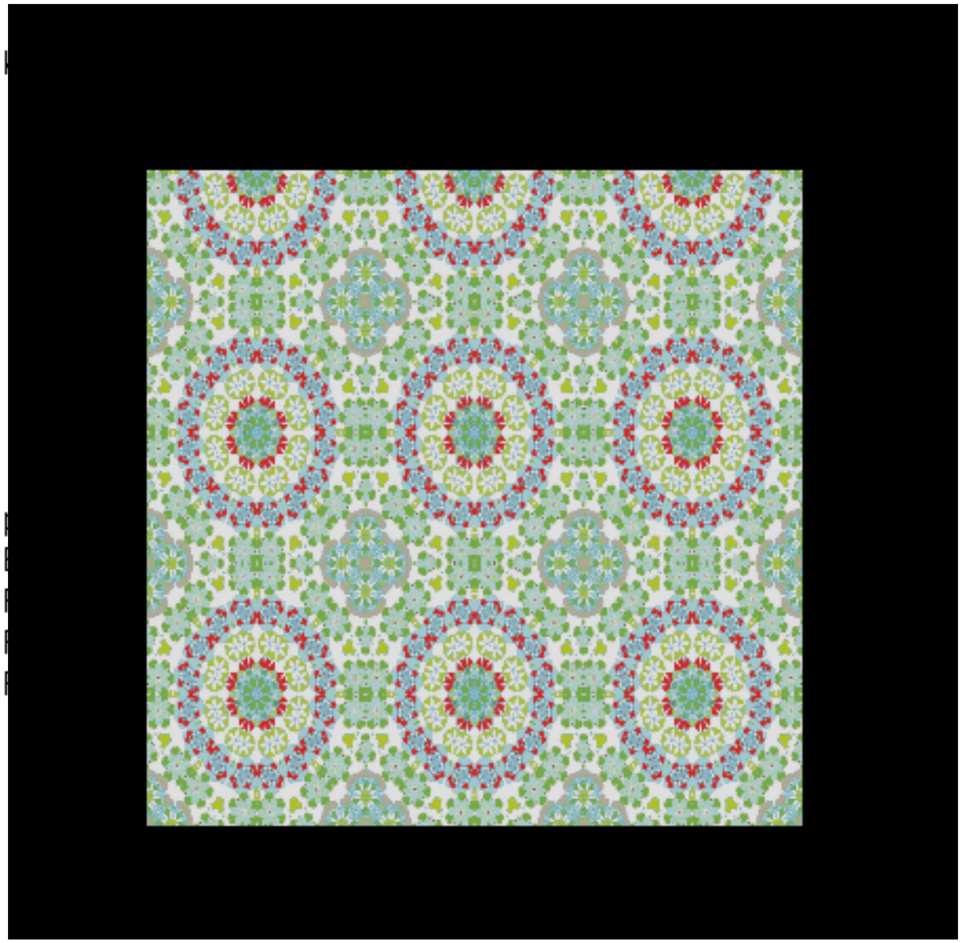
glOrtho

```
NAME
       glOrtho - multiply the current matrix with an orthographic matrix

C SPECIFICATION
       void glOrtho( GLdouble left,
                     GLdouble right,
                     GLdouble bottom,
                     GLdouble top,
                     GLdouble zNear,
                     GLdouble zFar )

PARAMETERS
       left, right Specify the coordinates for the left and right vertical clipping planes.

       bottom, top Specify the coordinates for the bottom and top horizontal clipping planes.

       zNear, zFar Specify  the  distances  to  the  nearer  and farther depth clipping planes.  These values are negative if the plane is to be behind the
                   viewer.

DESCRIPTION
       glOrtho describes a transformation that produces a parallel projection.  The current matrix (see glMatrixMode) is multiplied by this matrix and  the
       result replaces the current matrix, as if glMultMatrix were called with the following matrix as its argument:

                  2
        ------------       0               0               tx
        right - left

                               2
            0            ------------      0               ty
                         top - bottom

                                          -2
            0              0         ------------          tz
                                      zFar-zNear

            0              0               0               1

       where

                                                              tx = - (right + left) / (right - left)

                                                              ty = - (top + bottom) / (top - bottom)

                                                              tz = - (zFar + zNear) / (zFar - zNear)

       Typically,  the  matrix  mode  is GL_PROJECTION, and (left, bottom,  -zNear) and (right, top,  -zNear) specify the points on the near clipping plane
       that are mapped to the lower left and upper right corners of the window, respectively, assuming that the eye is located at (0, 0, 0).  -zFar  speci-
       fies the location of the far clipping plane.  Both zNear and zFar can be either positive or negative.

       Use glPushMatrix and glPopMatrix to save and restore the current matrix stack.
```

glMatrixMode(GL_TEXTURE)



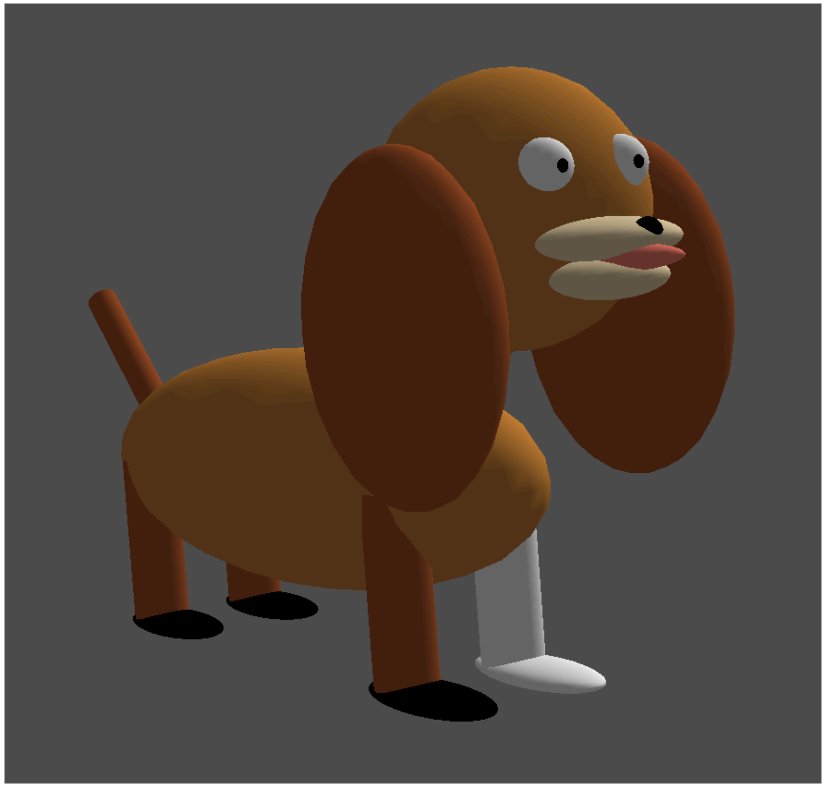
```
virtual void RenderPiece(vtkRenderer *ren, vt
{
        RemoveVTKOpenGLStateSideEffects();
        SetupLight();

        glMatrixMode(GL_TEXTURE);
        glPushMatrix();
        glScalef(3, 2.5, 1);

        glEnable(GL_TEXTURE_2D);

        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 
                         0, GL_RGB, GL_UNSIGNED_BYT
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTU
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTU
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTU
        glBegin(GL_QUADS);
        glTexCoord2f(0,0);
        glVertex3f(10, -10, -10);
```



Project 2B

Project #2B (7%), Due Weds Feb 27th

- Goal: modify ModelView matrix to create dog out of spheres and cylinders
- New code skeleton: "project2B.cxx"
- No geometry file needed.
- You will be able to do this w/ glPush/PopMatrix, glRotatef, glTranslatef, and glScalef.



Project #2B (7%), Due Weds Feb 27th

- G
  m
  of
- N
  "p
- N
- Yo
  w
  gl
  ar

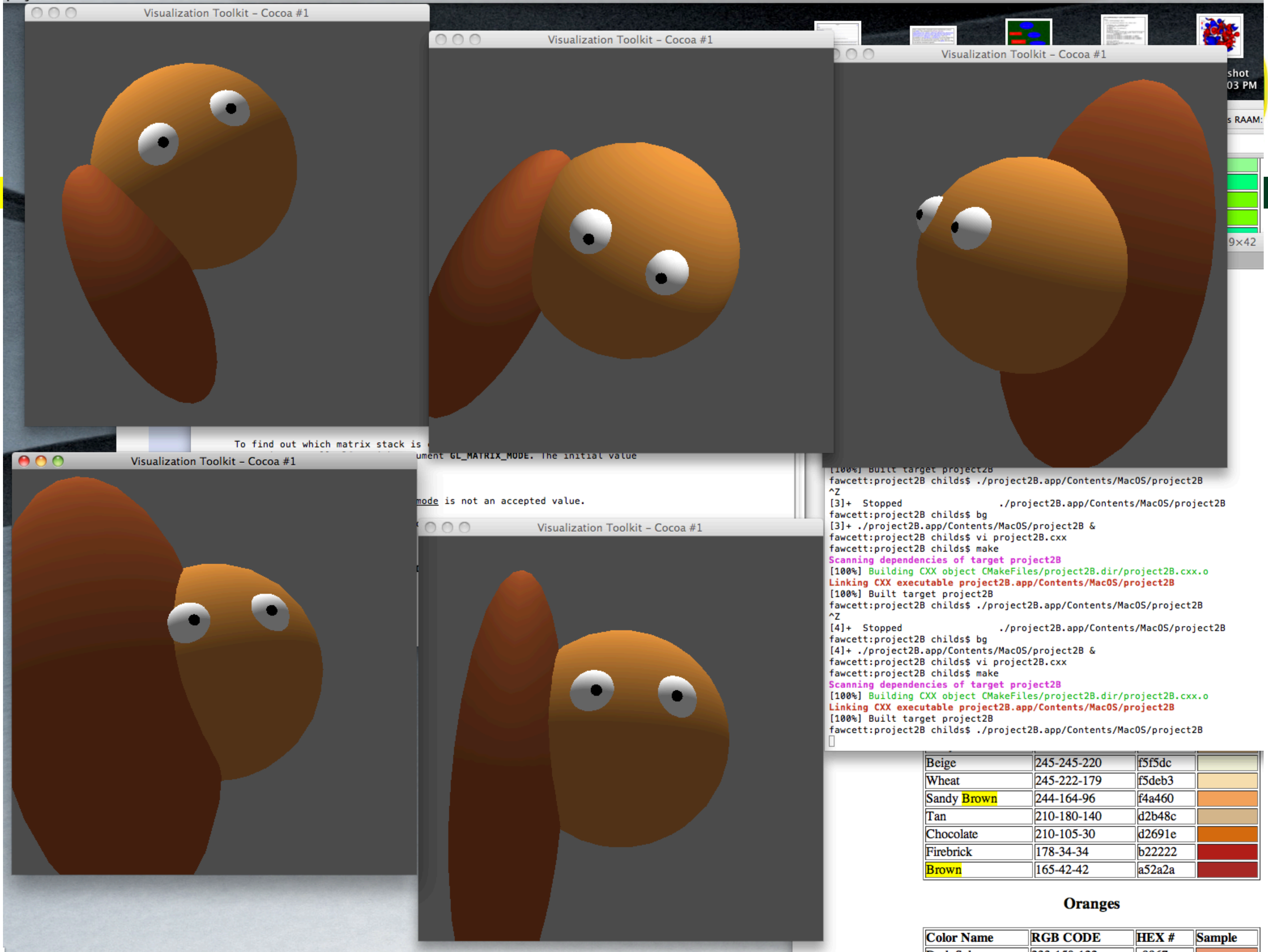
Contents of project2B.cxx

- Routine for generating spheres
- Routine for generating cylinders
- Routine for generating head, eyes, and pupils

What is the correct answer?

- The correct answer is:
  - Something that looks like a dog
    - No obvious problems with output geometry.
  - Something that uses the sphere and cylinder classes.
    - If you use something else, please clear it with me first.
      - I may fail your project if I think you are using outside resources that make the project too easy.
  - Something that uses rotation for the neck and tail.

- Aside from that, feel free to be as creative as you want … color, breed, etc.


Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
shot
03 PM
s RAAM:
9×42

To find out which matrix stack is

ument GL_MATRIX_MODE. The initial value

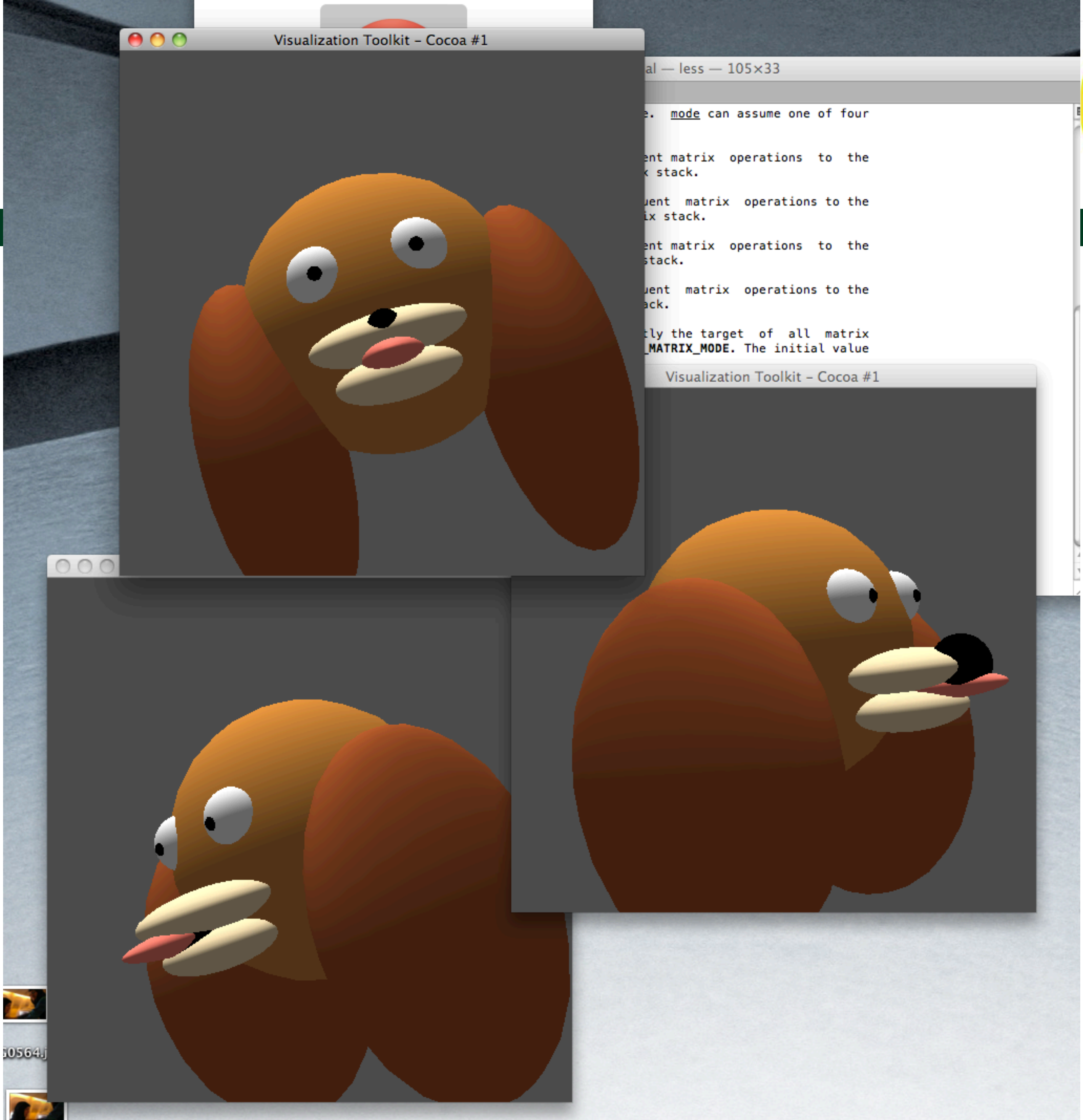
mode is not an accepted value.

```
[100%] Built target project2B
fawcett:project2B childs$ ./project2B.app/Contents/MacOS/project2B
^Z
[3]+  Stopped                 ./project2B.app/Contents/MacOS/project2B
fawcett:project2B childs$ bg
[3]+ ./project2B.app/Contents/MacOS/project2B &
fawcett:project2B childs$ vi project2B.cxx
fawcett:project2B childs$ make
Scanning dependencies of target project2B
[100%] Building CXX object CMakeFiles/project2B.dir/project2B.cxx.o
Linking CXX executable project2B.app/Contents/MacOS/project2B
[100%] Built target project2B
fawcett:project2B childs$ ./project2B.app/Contents/MacOS/project2B
^Z
[4]+  Stopped                 ./project2B.app/Contents/MacOS/project2B
fawcett:project2B childs$ bg
[4]+ ./project2B.app/Contents/MacOS/project2B &
fawcett:project2B childs$ vi project2B.cxx
fawcett:project2B childs$ make
Scanning dependencies of target project2B
[100%] Building CXX object CMakeFiles/project2B.dir/project2B.cxx.o
Linking CXX executable project2B.app/Contents/MacOS/project2B
[100%] Built target project2B
fawcett:project2B childs$ ./project2B.app/Contents/MacOS/project2B
```

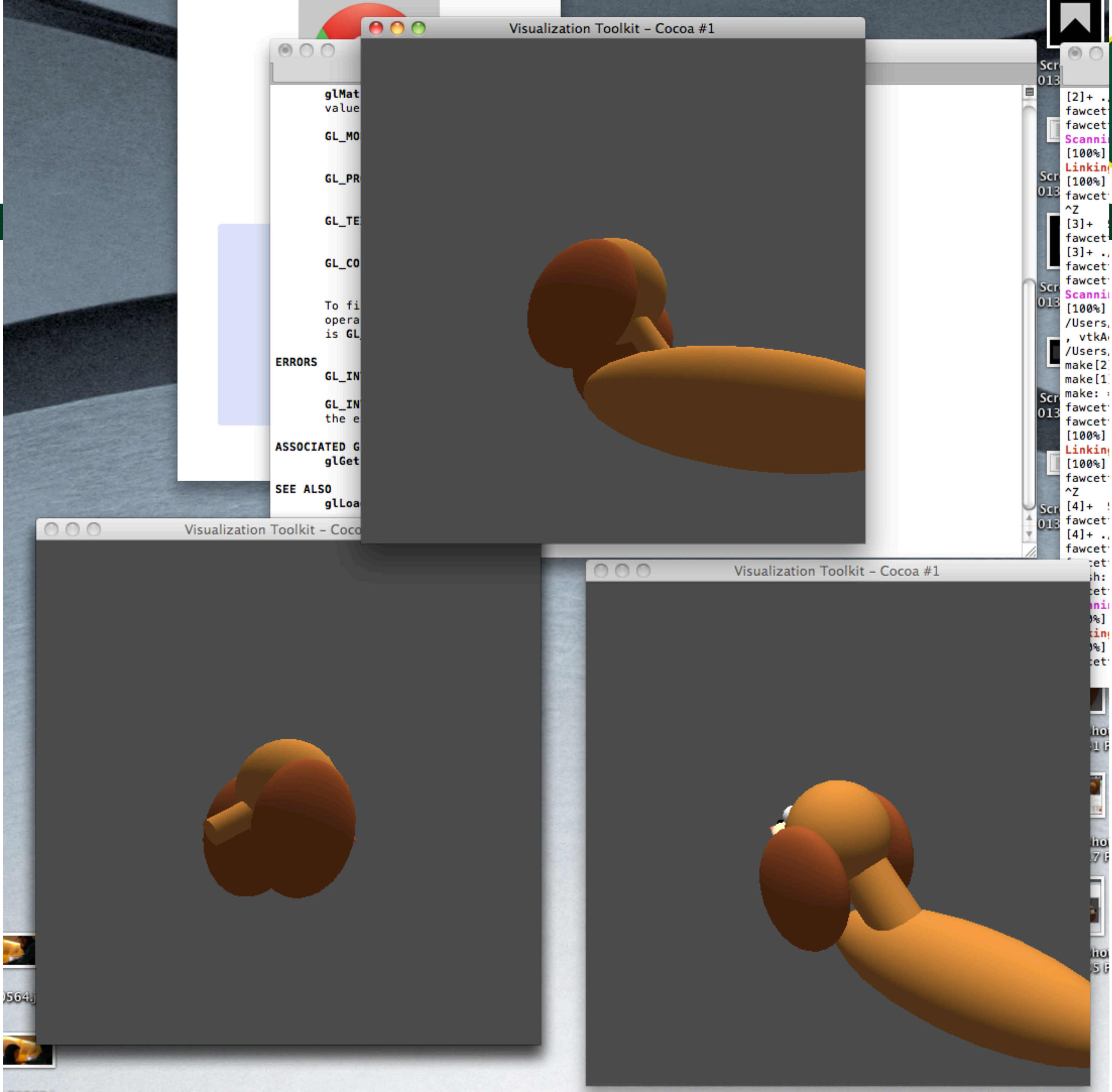
Beige	245-245-220	f5f5dc	⬜
Wheat	245-222-179	f5deb3	🟨
Sandy Brown	244-164-96	f4a460	🟧
Tan	210-180-140	d2b48c	🟫
Chocolate	210-105-30	d2691e	🟧
Firebrick	178-34-34	b22222	🟥
Brown	165-42-42	a52a2a	🟥

Oranges

Color Name	RGB CODE	HEX #	Sample

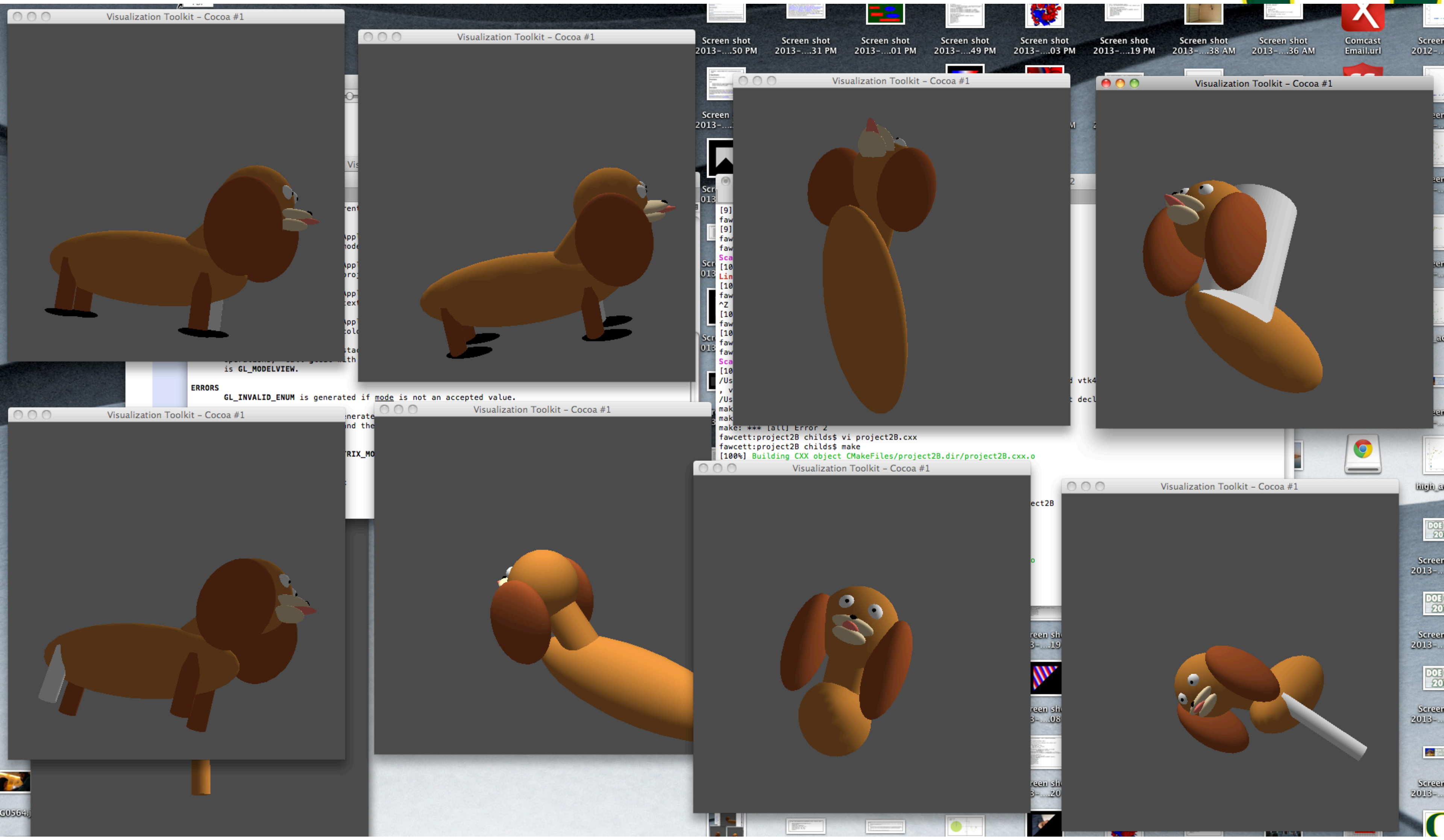

Visualization Toolkit – Cocoa #1
al — less — 105×33
Visualization Toolkit – Cocoa #1
i0564.j

```
e.  mode can assume one of four

ent matrix  operations  to  the
k stack.

uent  matrix  operations to the
ix stack.

ent matrix  operations  to  the
stack.

uent  matrix  operations to the
ack.

tly the target  of  all  matrix
_MATRIX_MODE. The initial value
```


Visualization Toolkit – Cocoa #1
Visualization Toolkit – Coco
Visualization Toolkit – Cocoa #1

```
        glMat
        value

        GL_MO


        GL_PR


        GL_TE


        GL_CO


        To fi
        opera
        is GL

ERRORS
        GL_IN

        GL_IN
        the e

ASSOCIATED G
        glGet

SEE ALSO
        glLoa
```

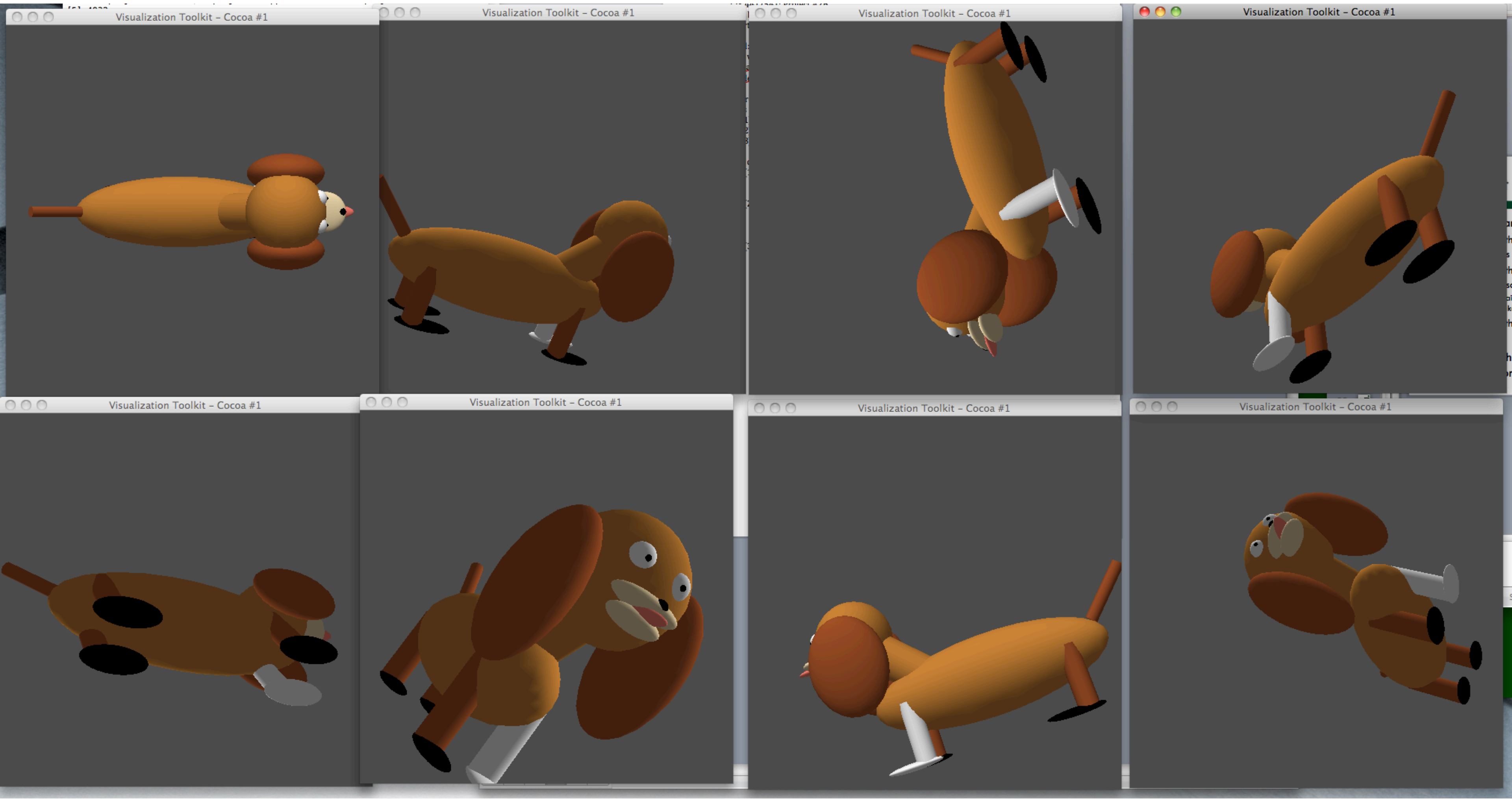
```
[2]+ .
fawcet
fawcet
Scanni
[100%]
Linkin
[100%]
fawcet
^Z
[3]+  S
fawcet
[3]+ .
fawcet
fawcet
Scanni
[100%]
/Users
, vtkA
/Users
make[2
make[1
make: 
fawcet
fawcet
[100%]
Linkin
[100%]
fawcet
^Z
[4]+  S
fawcet
[4]+ .
fawcet
```


Visualization Toolkit – Cocoa #1
Screen shot 2013-....50 PM
Screen shot 2013-....31 PM
Screen shot 2013-....01 PM
Screen shot 2013-....49 PM
Screen shot 2013-....03 PM
Screen shot 2013-....19 PM
Screen shot 2013-....38 AM
Screen shot 2013-....36 AM
Comcast Email.url

```
        is GL_MODELVIEW.

ERRORS
        GL_INVALID_ENUM is generated if mode is not an accepted value.
```

```
make: *** [all] Error 2
fawcett:project2B childs$ vi project2B.cxx
fawcett:project2B childs$ make
[100%] Building CXX object CMakeFiles/project2B.dir/project2B.cxx.o
```

For your reference: my dog


Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1

Transparent Geometry

COMPOSITING AND BLENDING

Ed Angel

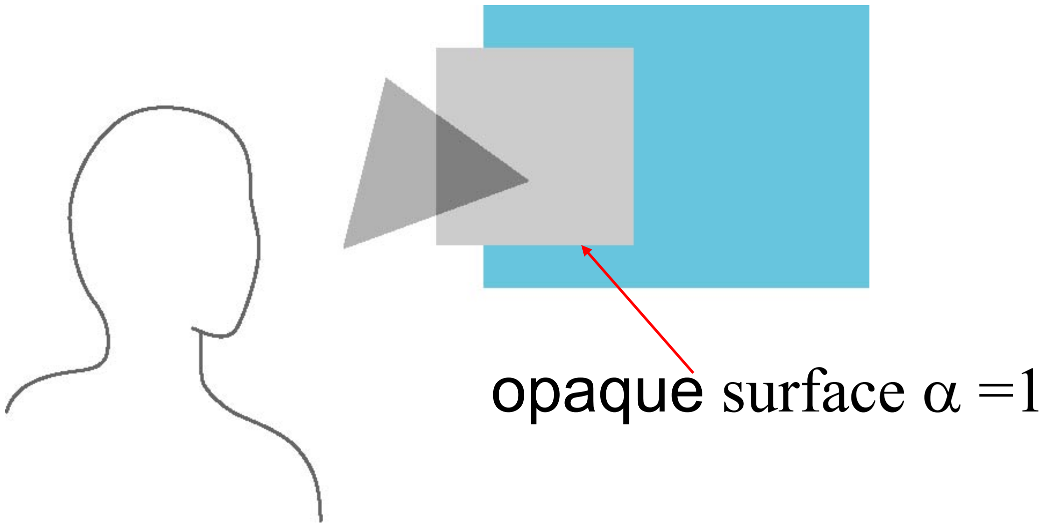
Professor of Computer Science, Electrical and Computer Engineering, and Media Arts

University of New Mexico

Opacity and Transparency

- Opaque surfaces permit no light to pass through
- Transparent surfaces permit all light to pass
- Translucent surfaces pass some light

  translucency = 1 – opacity (α)



Transparency

- If you have an opaque red square in front of a blue square, what color would you see?
  - Red
- If you have a 50% transparent red square in front of a blue square, what color would you see?
  - Purple
- If you have a 100% transparent red square in front of a blue square, what color would you see?
  - Blue

(One) Formula For Transparency

- Front = (Fr,Fg,Fb,Fa)
  - a = alpha, transparency factor
    - Sometimes percent
    - Typically 0-255, with 255 = 100%, 0 = 0%
- Back = (Br,Bg,Bb,Ba)
- Equation = (Fa*Fr+(1-Fa)*Br,

  Fa*Fg+(1-Fa)*Bg,

  Fa*Fb+(1-Fa)*Bb,

  Fa+(1-Fa)*Ba)

Transparency

- If you have an 25% transparent red square (255,0,0) in front of a blue square (0,0,255), what color would you see (in RGB)?
  - (192,0,64)
- If you have an 25% transparent blue square (0,0,255) in front of a red square (255,0,0), what color would you see (in RGB)?
  - (64,0,192)

Implementation

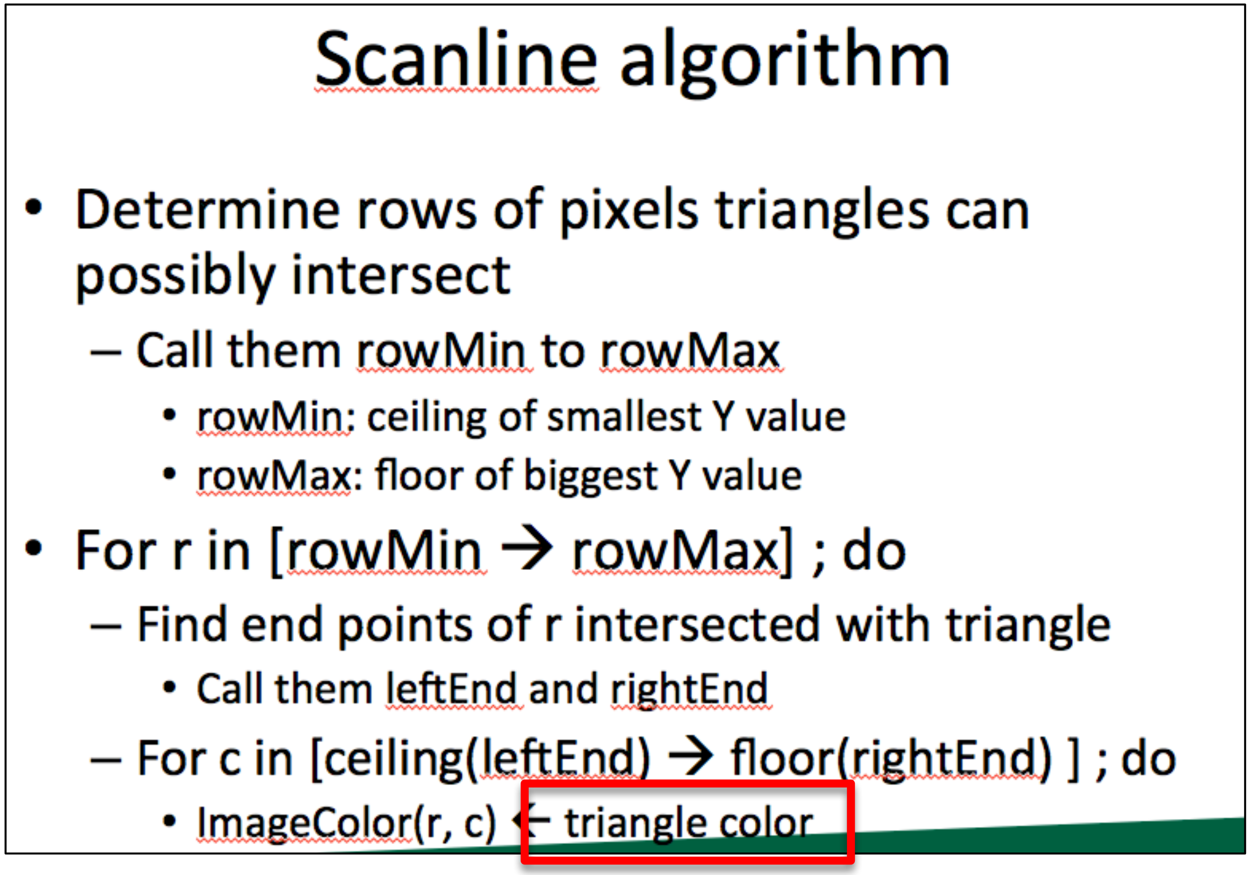
- Per pixel storage:
  - RGB: 3 bytes
  - Alpha: 1 byte
  - Z: 4 bytes

- Alpha used to control blending of current color and new colors

Vocab term reminder: fragment

- Fragment is the contribution of a triangle to a single pixel

Almost certain to use term "fragment" on midterm and expect that you know what it means



Scanline algorithm

- Determine rows of pixels triangles can possibly intersect
  - Call them rowMin to rowMax
    - rowMin: ceiling of smallest Y value
    - rowMax: floor of biggest Y value
- For r in [rowMin → rowMax] ; do
  - Find end points of r intersected with triangle
    - Call them leftEnd and rightEnd
  - For c in [ceiling(leftEnd) → floor(rightEnd)] ; do
    - ImageColor(r, c) ← $\boxed{\text{triangle color}}$

Examples

- Imagine pixel (i, j) has:
  - RGB = 255/255/255
  - Alpha=255
  - Depth = -0.5
- And we contribute fragment:
  - RGB=0/0/0
  - Alpha=128
  - Depth = -0.25
- What do we get?
- Answer: 128/128/128, Z = -0.25
- What's the alpha?

Examples

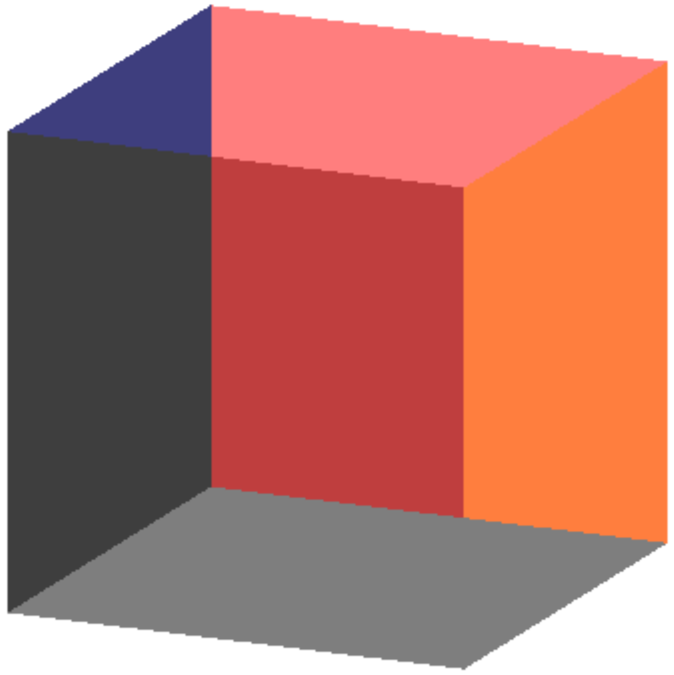
- Imagine pixel (i, j) has:
  - RGB = 255/255/255
  - Alpha=128
  - Depth = -0.25
- And we contribute fragment:
  - RGB=0/0/0
  - Alpha=255
  - Depth = -0.5
- What do we get?
- Answer: (probably) 128/128/128, Z = -0.25
- What's the alpha?

System doesn’t work well for transparency

- Contribute fragments in this order:
  - Z=-0.1
  - Z=-0.9
  - Z=-0.5
  - Z=-0.4
  - Z=-0.6
- Model is too simple. Not enough info to resolve!

Order Dependency

- Is this image correct?
  - Probably not
  - Polygons are rendered in the order they pass down the pipeline
  - Blending functions are order dependent

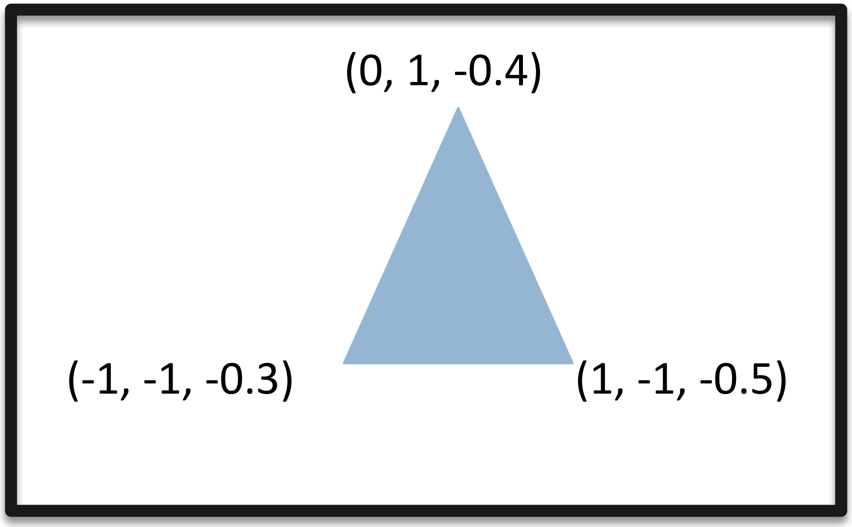


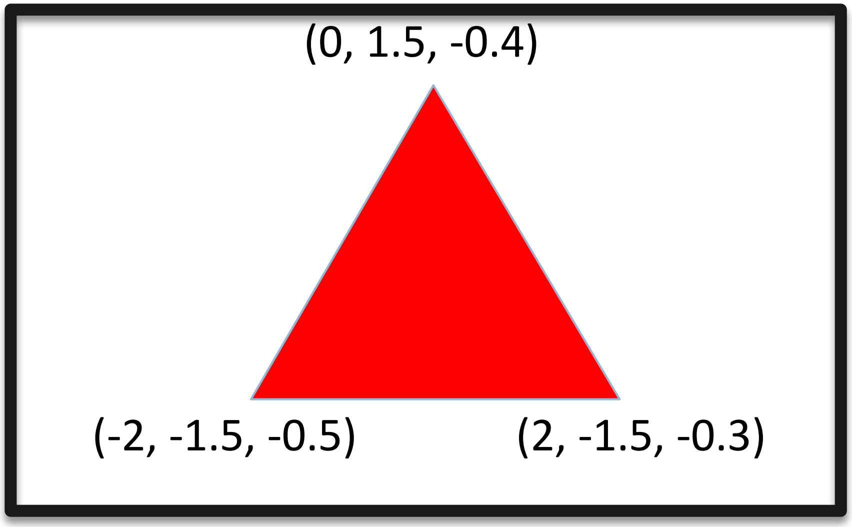
How do you sort?

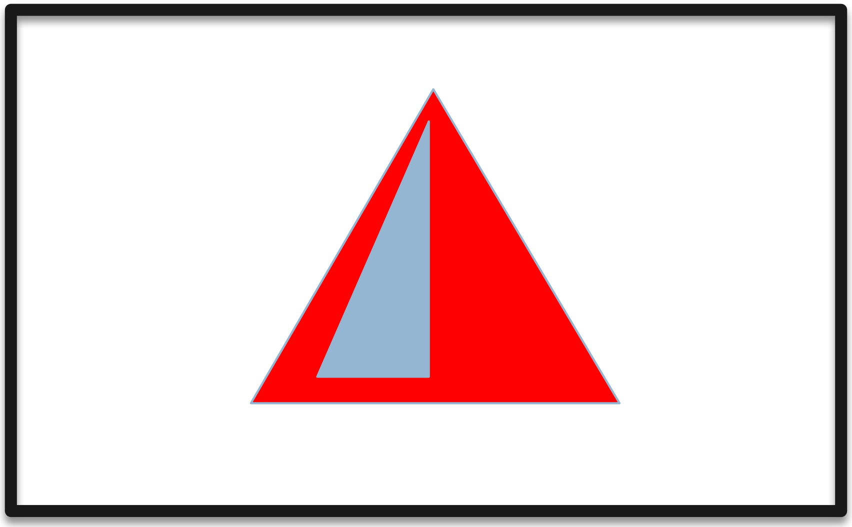
- 1) Calculate depth of each triangle center.
- 2) Sort based on depth
  - Not perfect, but good

- In practice: sort along X, Y, and Z and use “dominant axis” and only do “perfect sort” when rotation stops

But there is a problem…







Depth Peeling

- a multi-pass technique that renders transparent polygonal geometry without sorting
- Pass #1:
  - render as opaque, but note opacity of pixels placed on top
  - treat this as “top layer”
  - save Z-buffer and treat this as “max”
- Pass #2:
  - render as opaque, but ignore fragments beyond “max”
- repeat, repeat…