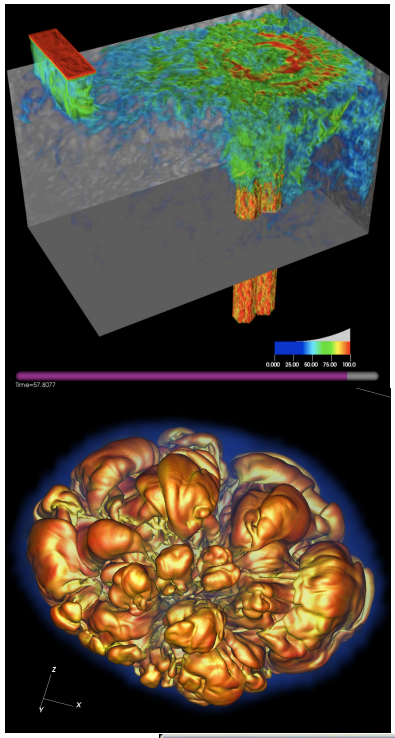
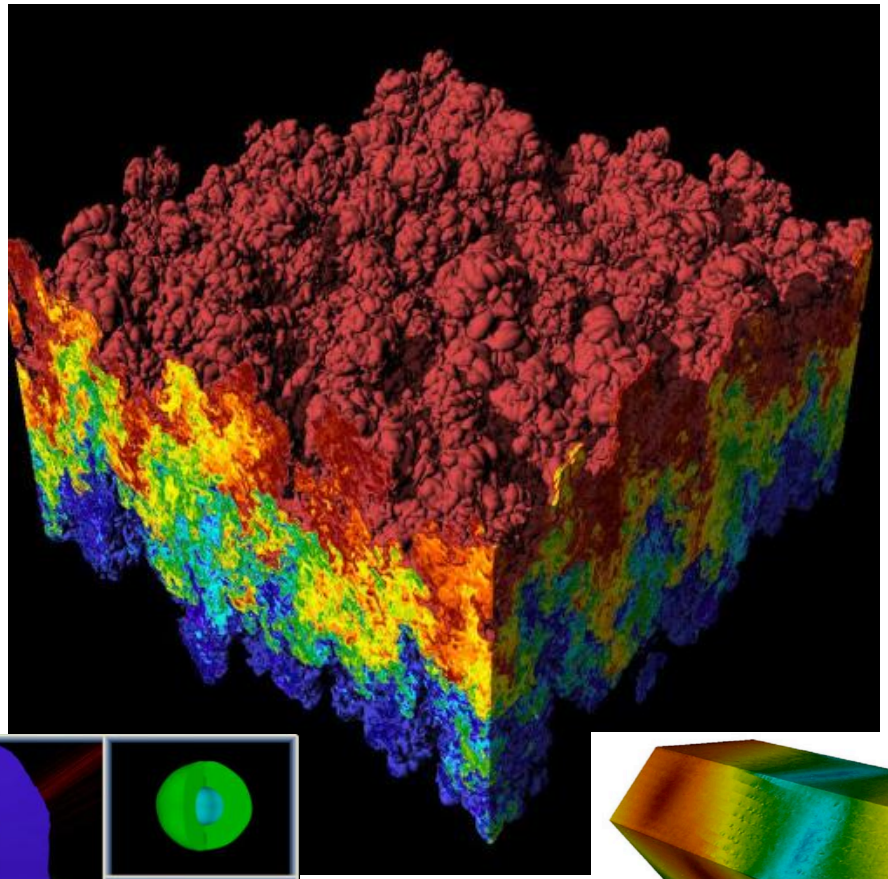
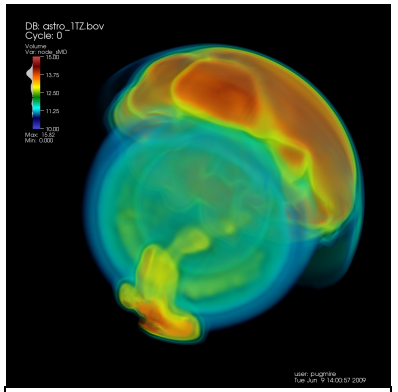
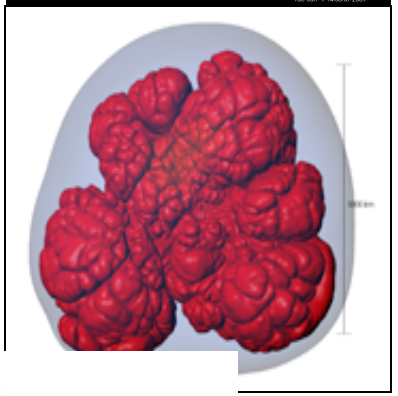
CIS 441/541: Intro to Computer Graphics

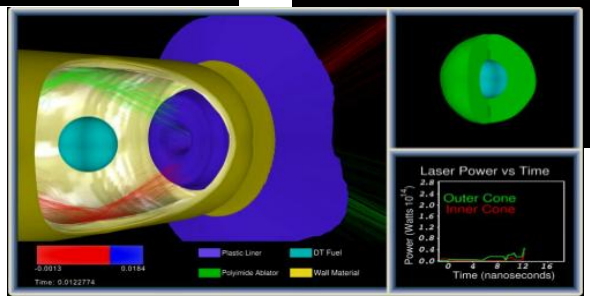
Lecture 11: Even More OpenGL!

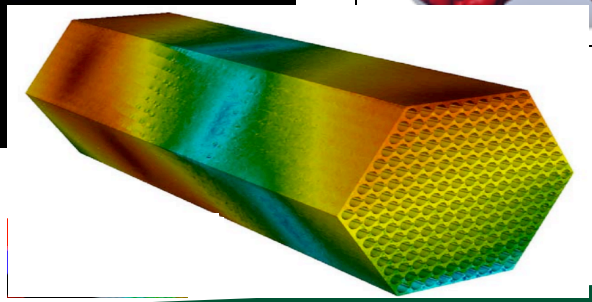












February 21, 2019

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

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~~
- Midterm worth 30 points, no quiz on Week 10
- Details:
  - No notes.
  - Not expected to memorize Phong shading equation.
  - Will be derived directly from 1A-1F, 2A. Not 2B.

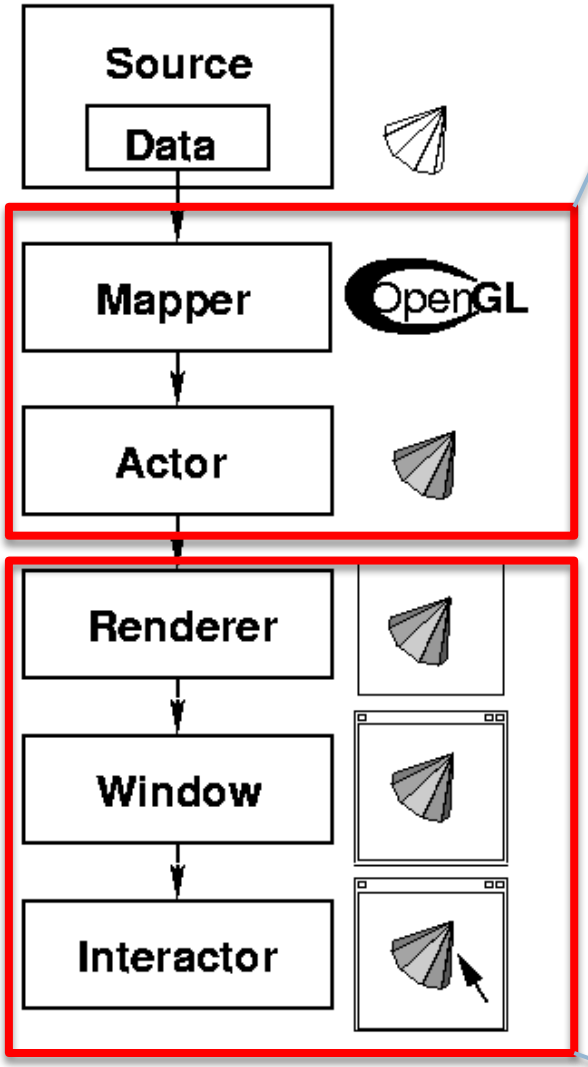
Midterm

- 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 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)
```

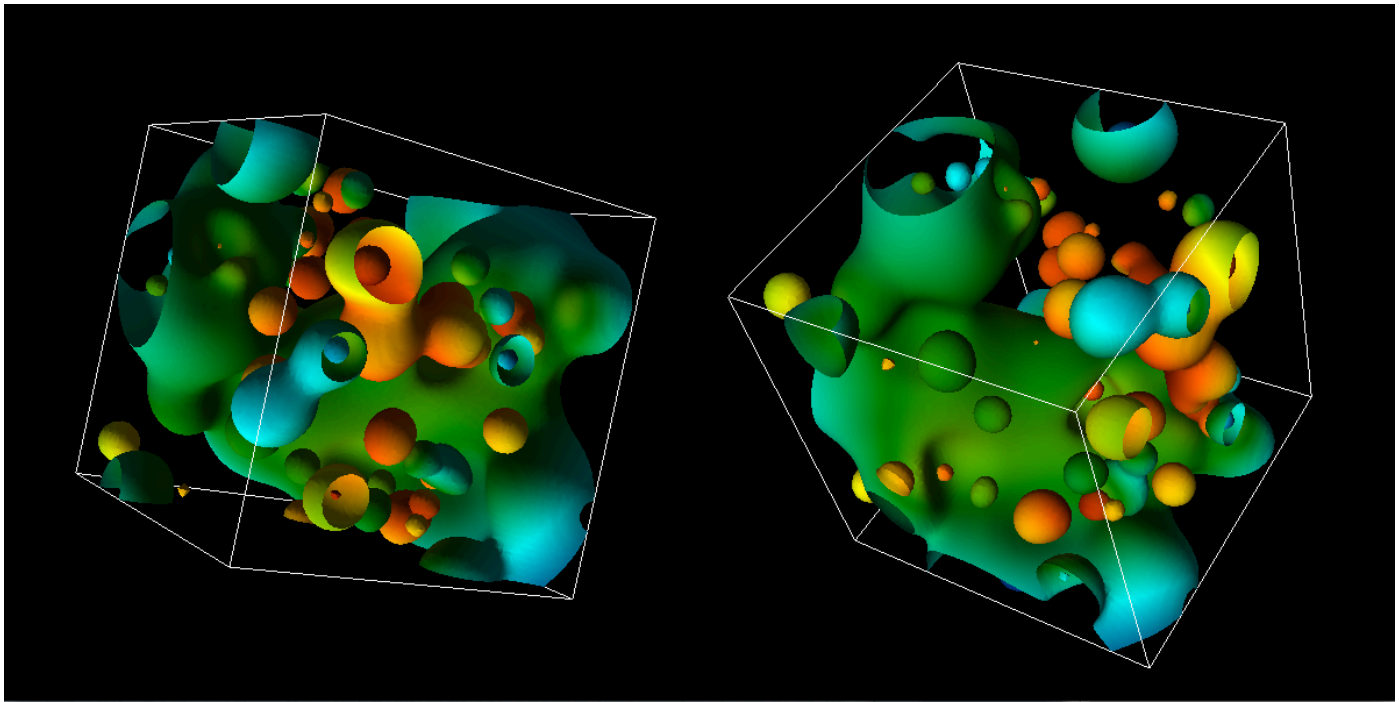
Allows the mouse to be used to interact wth the data.

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

We will re-use these.

Project #2A (8%), Due 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)

Final Projects

- 541: everyone should do a self-defined project
- 441: your choice
  - Do a self-defined project
  - Do a pre-defined project

(DRAFT) Pre-defined projects

- Hoping to have prompts up by Thursday Feb 28th
- Likely topics:
  - 2 WebGL projects
  - 3 shader programs
  - GPU programming
  - Blender
- Possible topics:
  - Vulcan
  - Computer vision (?)

Self-defined projects

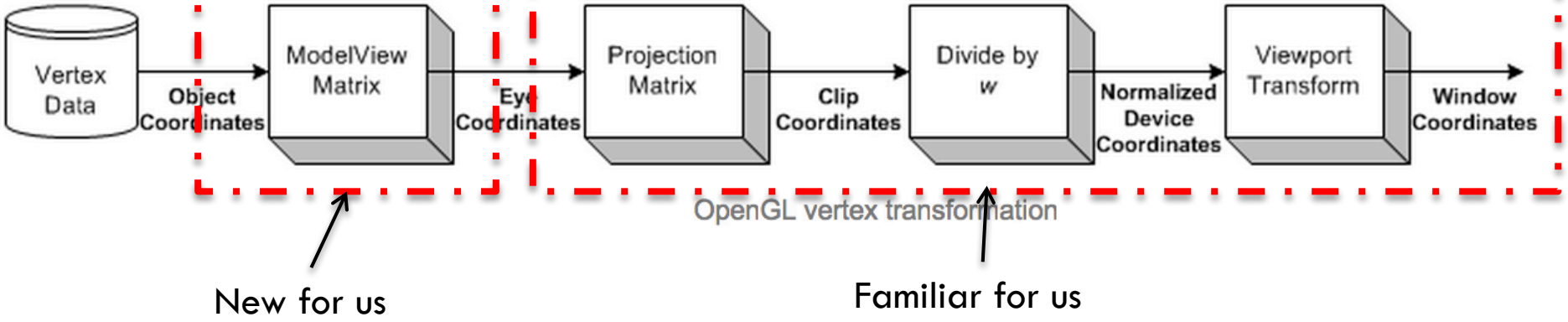
- Write a screen saver
- Write a SIMPLE video game
- Make a movie
- Model something
- Advanced computer graphics effects
  - Example: collision detection
  - Example: ray tracing
  - Example: volume rendering
  - Example: physics-based rendering

Process

- Send me a proposal by Saturday March 2nd
- The proposal should be for ~24 hours of work
- I will send feedback (too much, too little)
- If you don't send a proposal, then it is assumed that you are doing the pre-defined projects

Transforms in GL

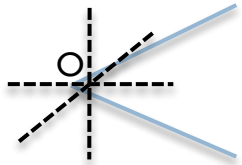
ModelView and Projection Matrices



- 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

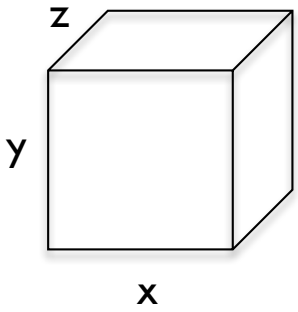
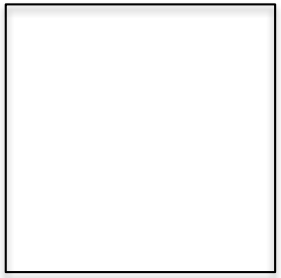


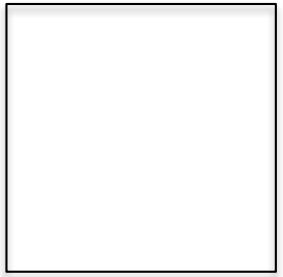
Image space:

All viewable objects within
-1 <= x,y,z <= +1



Screen space:

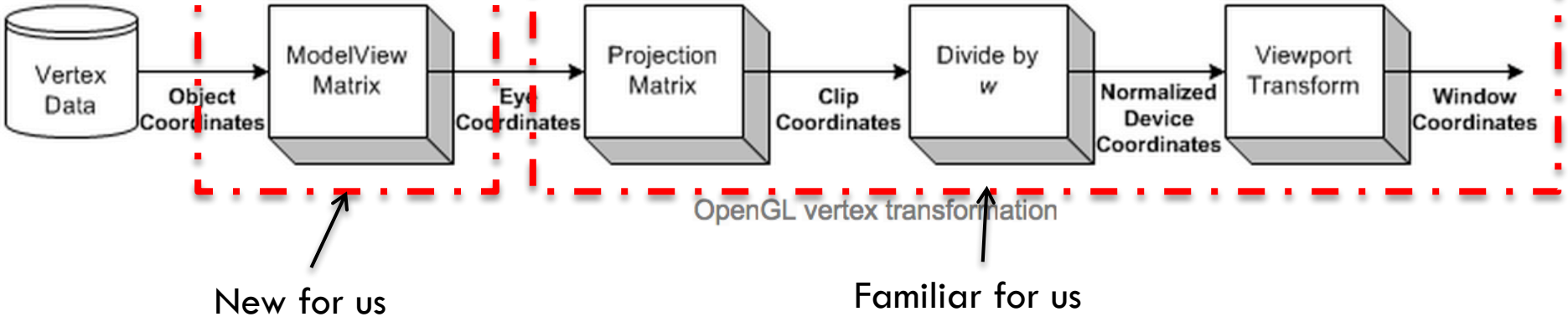
All viewable objects within
-1 <= x, y <= +1



Device space:

All viewable objects within
0<=x<=width, 0
<=y<=height

ModelView and Projection Matrices



- 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:

$$\begin{matrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{matrix}$$

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:

$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

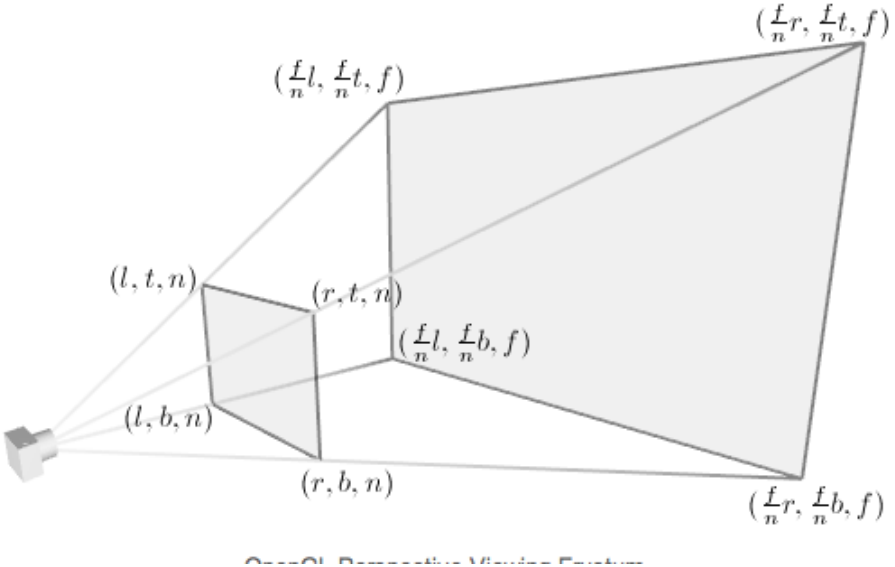
glMatrixMode sets the current matrix mode. mode can assume one of four values:

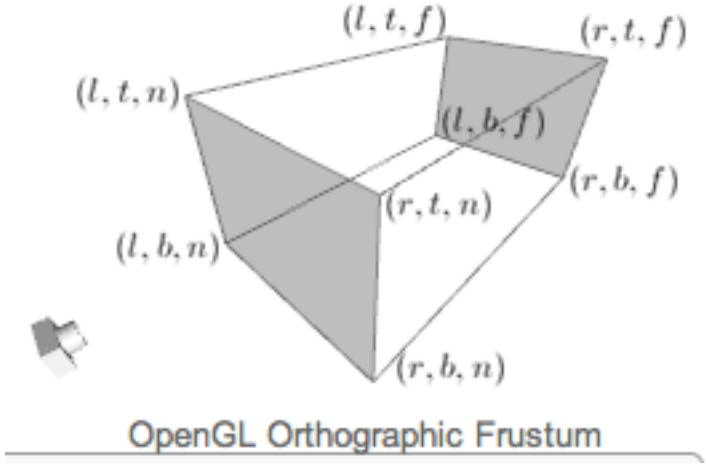
GL_MODELVIEW	Applies subsequent matrix operations to the modelview matrix stack.
GL_PROJECTION	Applies subsequent matrix operations to the projection matrix stack.
GL_TEXTURE	Applies subsequent matrix operations to the texture matrix stack.
GL_COLOR	Applies subsequent matrix operations to the color matrix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





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:

$$\begin{bmatrix} \frac{2\,zNear}{right - left} & 0 & A & 0 \\ 0 & \frac{2\,zNear}{top - bottom} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

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

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

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

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

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:

$$\begin{bmatrix} \frac{2}{right - left} & 0 & 0 & tx \\ 0 & \frac{2}{top - bottom} & 0 & ty \\ 0 & 0 & \frac{-2}{zFar - zNear} & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

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

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

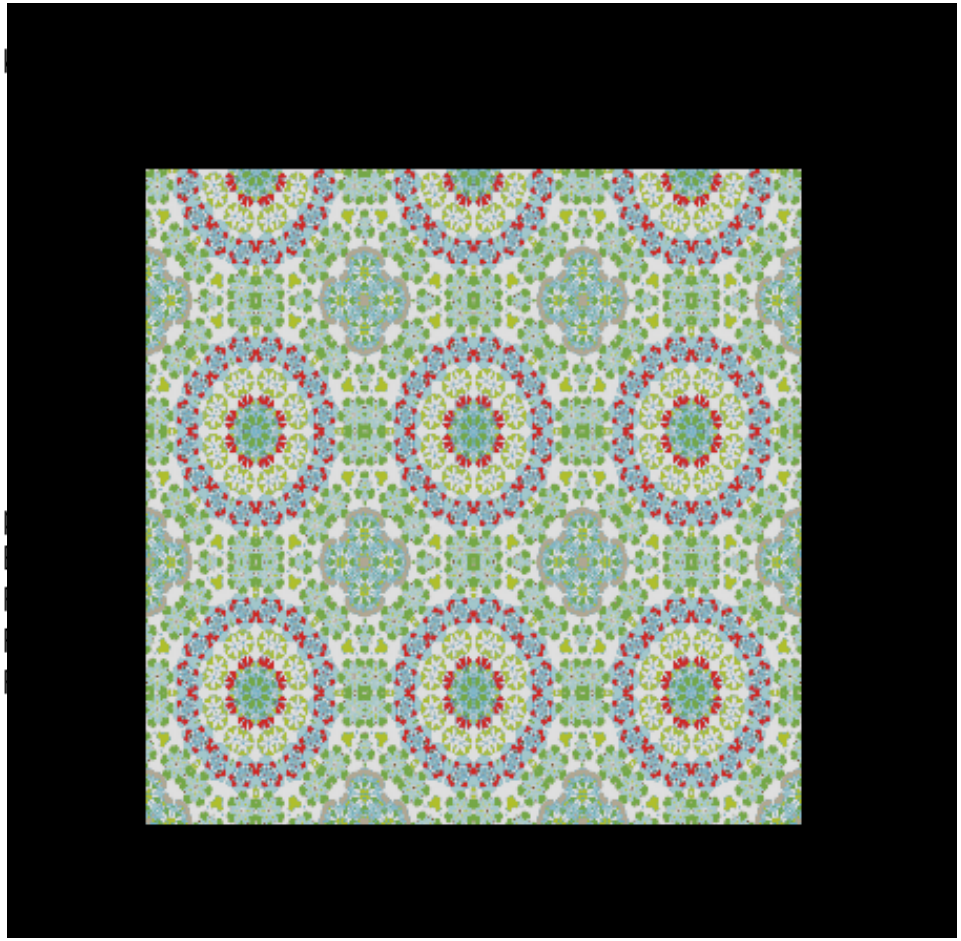
$$tz = -(zFar + zNear) / (zFar - zNear)$$

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 specifies 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.

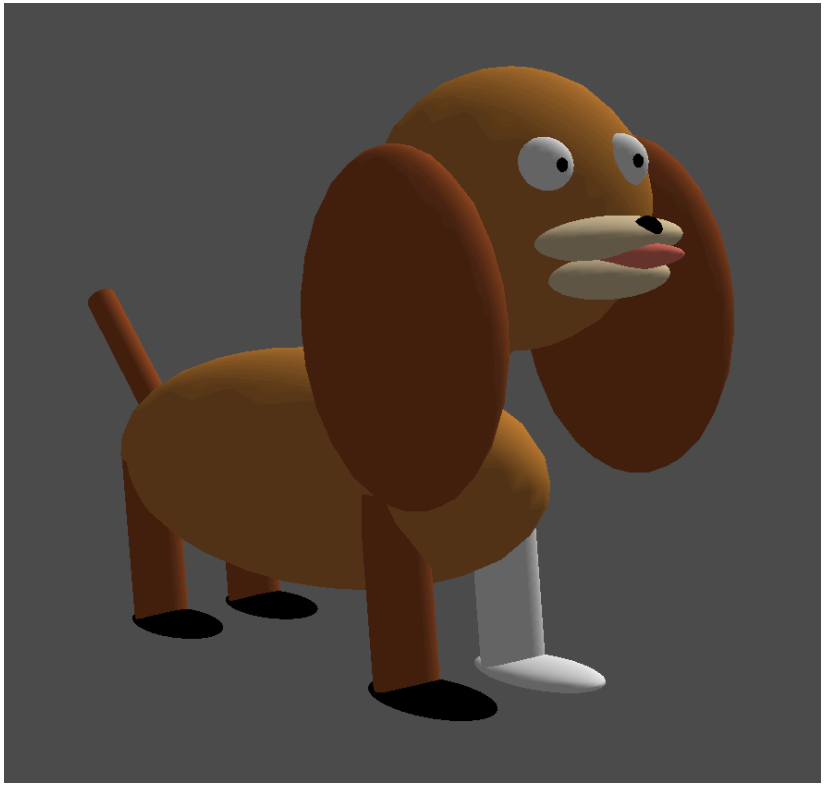
glMatrixMode(GL_TEXTURE)

```
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

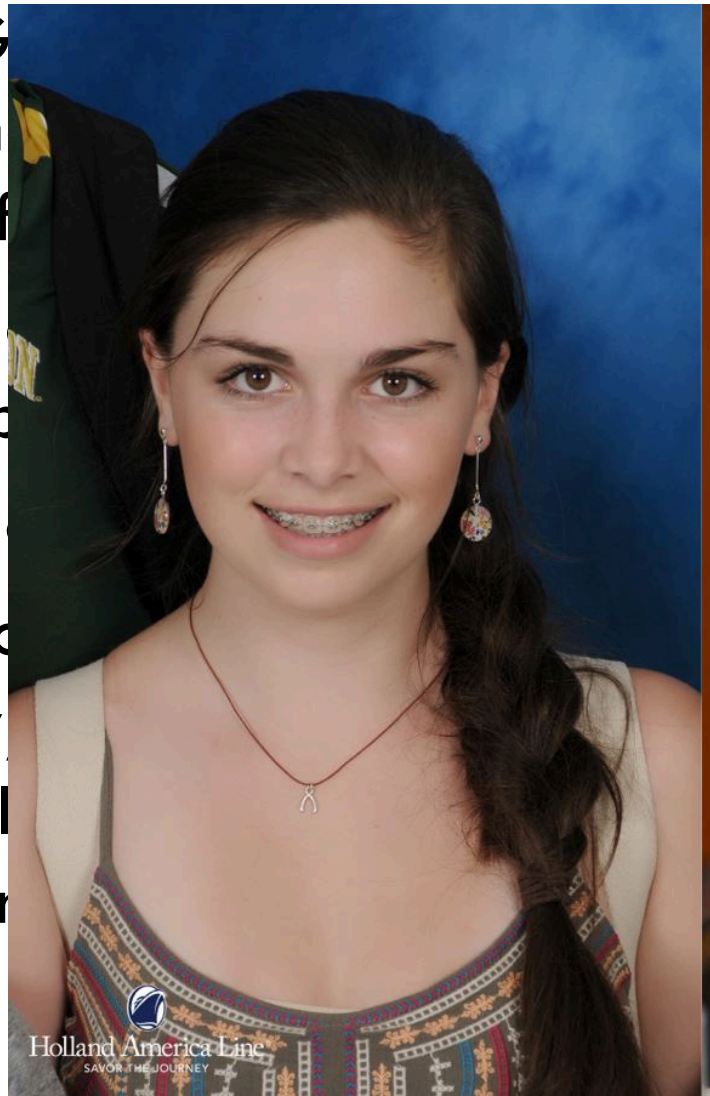
Project #2B (7%), Due Weds Feb 27th



- 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



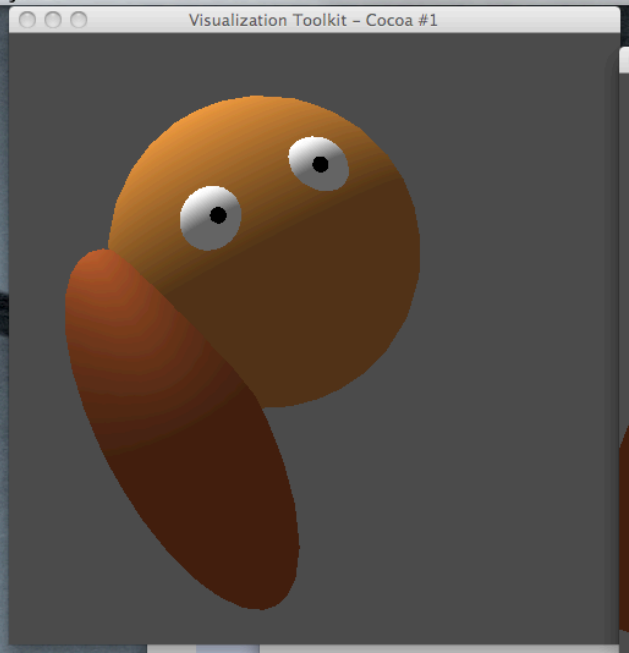
Contents of project2B.cxx

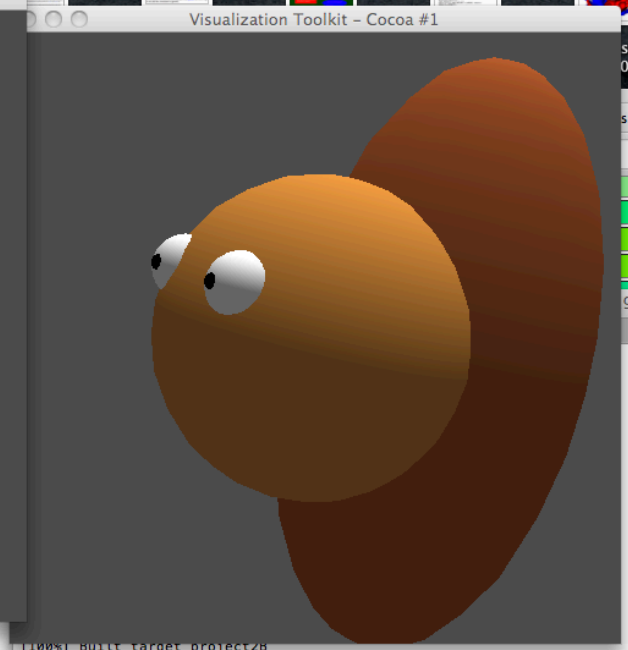
- Routine for generating spheres
- Routine for generating cylinders
- Routine for generating head, eyes, and pupils

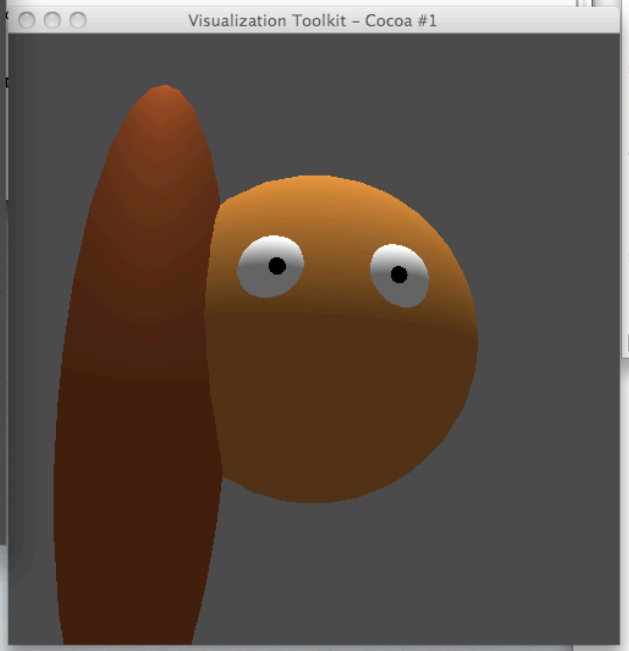
What is the correct answer?

- 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.







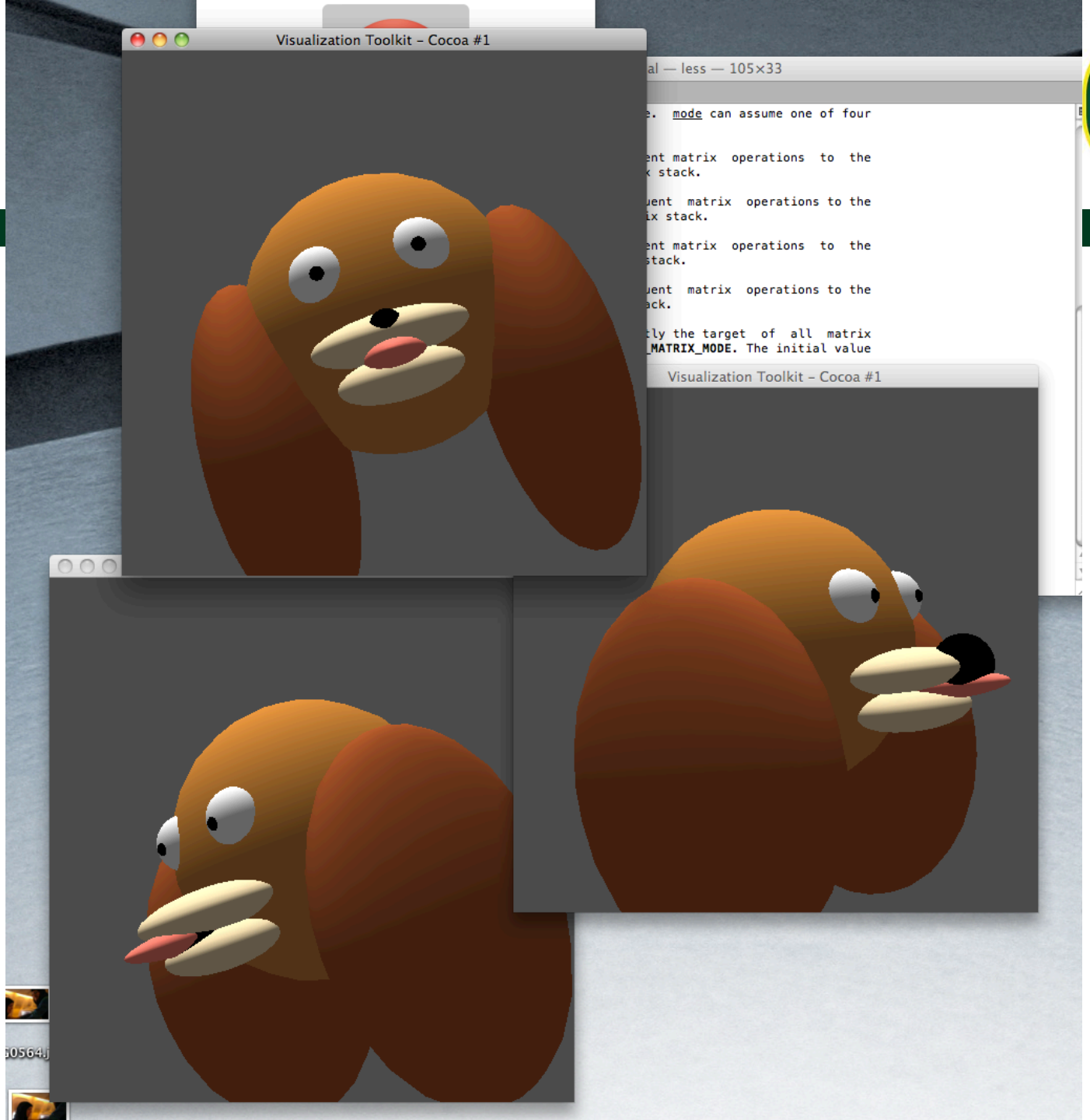
To find out which matrix stack is
ument GL_MATRIX_MODE. The initial value
mode is not an accepted value.

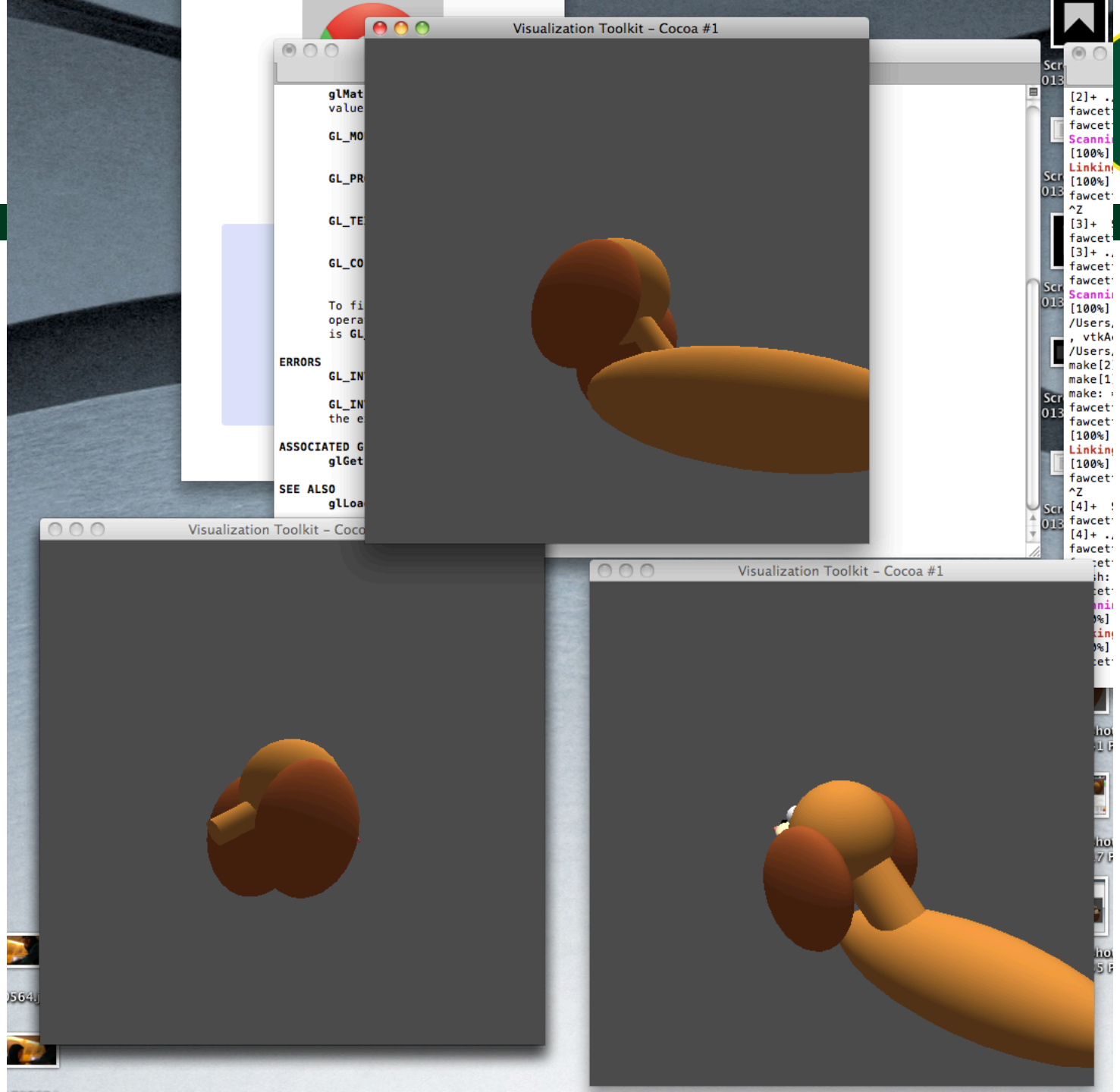
```
[100%] Built target project2B
fawcett:project2B childs$ ./project2B.app/Contents/MacOS/project2B
^Z
[3]+  Stopped                 ./project2B.app/Contents/MacOS/project2B
fawcett:project2B childs$ bg
[3]+ ./project2B.app/Contents/MacOS/project2B &
fawcett:project2B childs$ vi project2B.cxx
fawcett:project2B 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
```

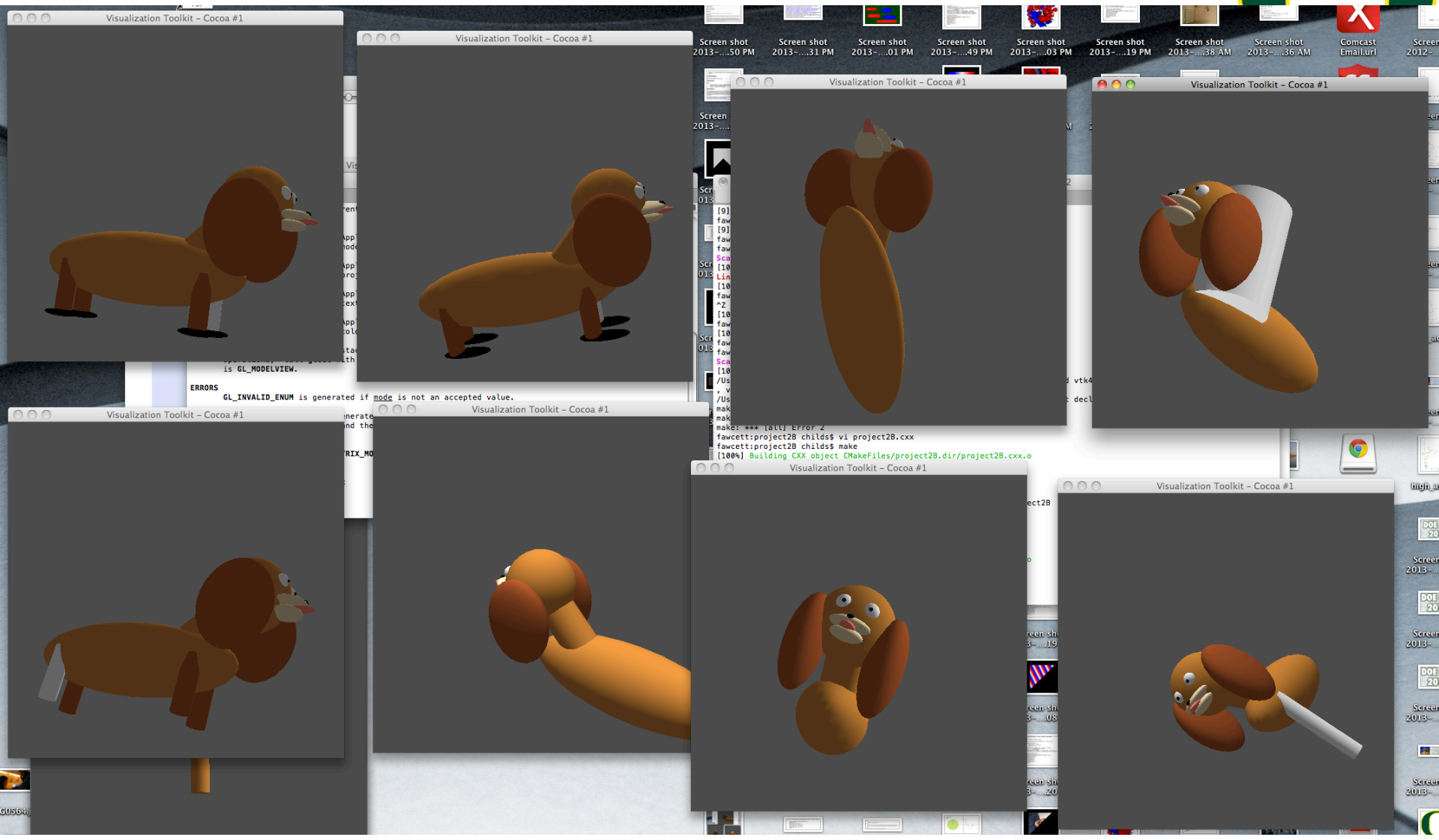
Beige	245-245-220	f5f5dc	
Wheat	245-222-179	f5deb3	
Sandy Brown	244-164-96	f4a460	
Tan	210-180-140	d2b48c	
Chocolate	210-105-30	d2691e	
Firebrick	178-34-34	b22222	
Brown	165-42-42	a52a2a	

Oranges

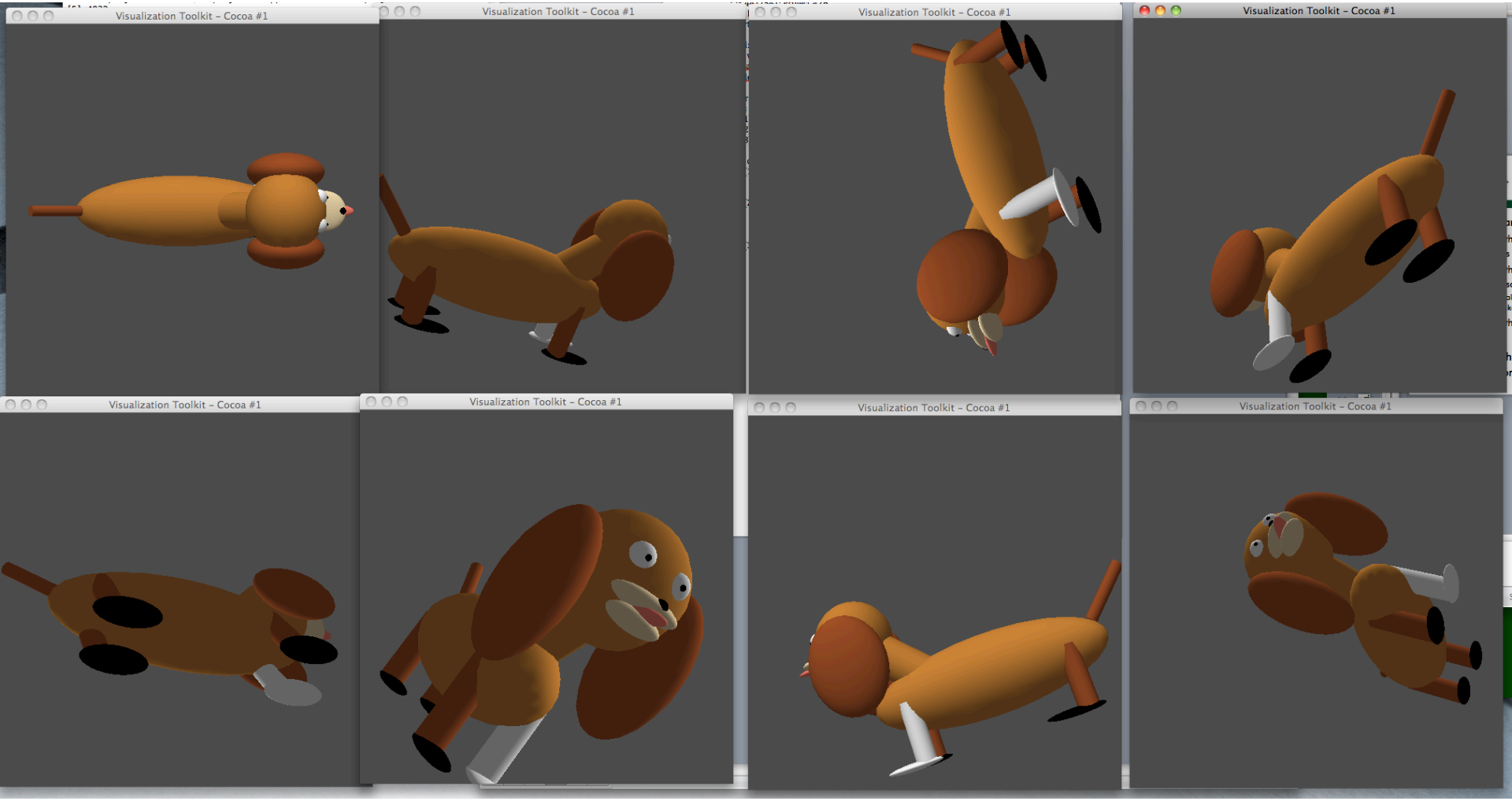
Color Name	RGB CODE	HEX #	Sample


Visualization Toolkit – Cocoa #1
al — less — 105×33
mode can assume one of four
nt matrix operations to the
stack.
ent matrix operations to the
x stack.
nt matrix operations to the
tack.
ent matrix operations to the
ck.
ly the target of all matrix
_MATRIX_MODE. The initial value
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
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
is GL_MODELVIEW.
ERRORS
GL_INVALID_ENUM is generated if mode is not an accepted value.
fawcett:project2B childs$ vi project2B.cxx
fawcett:project2B childs$ make
[100%] Building CXX object CMakeFiles/project2B.dir/project2B.cxx.o
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

For your reference: my dog



Transparent Geometry

COMPOSITING AND BLENDING

Ed Angel

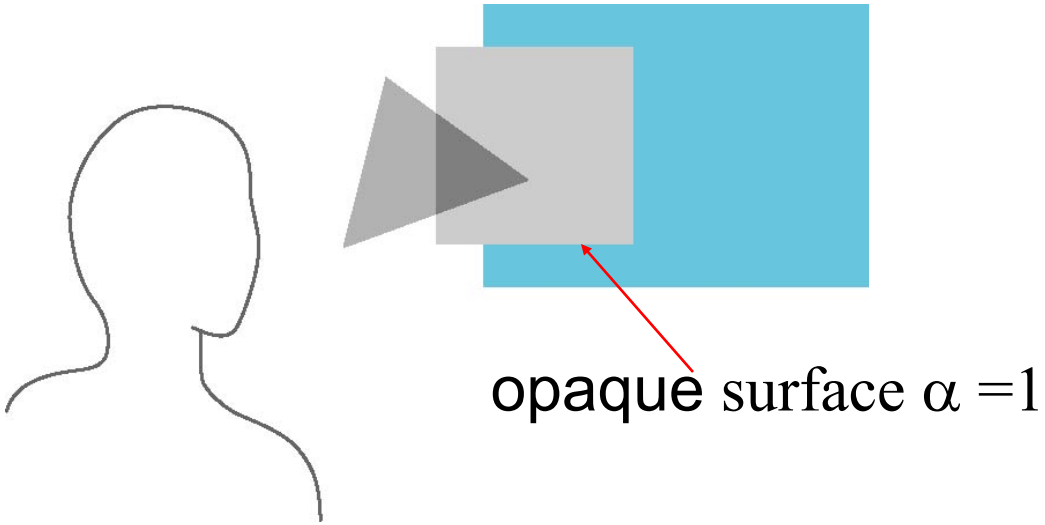
Professor of Computer Science, Electrical and Computer Engineering, and Media Arts

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) ← 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

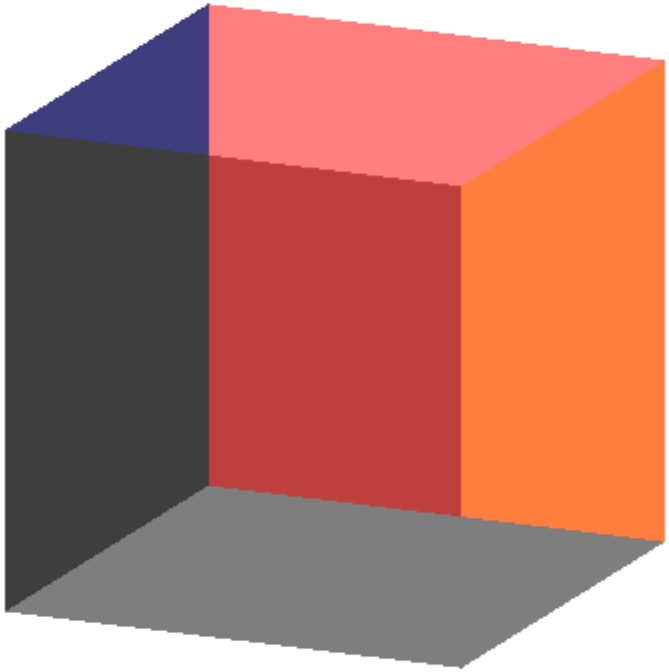
- Imagine pixel (i, j) has:
  - RGB = 255/255/255
  - Alpha=128
  - Depth = -0.25
- And we contribute fragment:
  - RGB=0/0/0
  - Alpha=255
  - Depth = -0.5
- What do we get?
- Answer: (probably) 128/128/128, Z = -0.25
- What's the alpha?

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

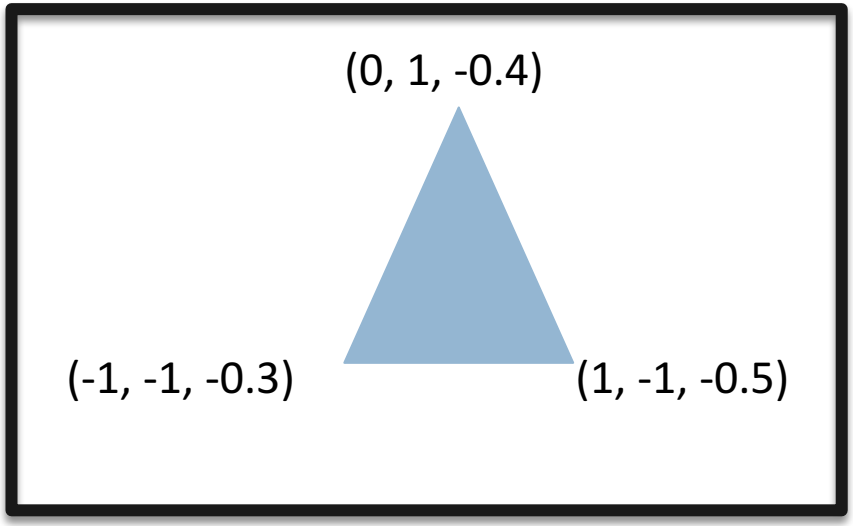


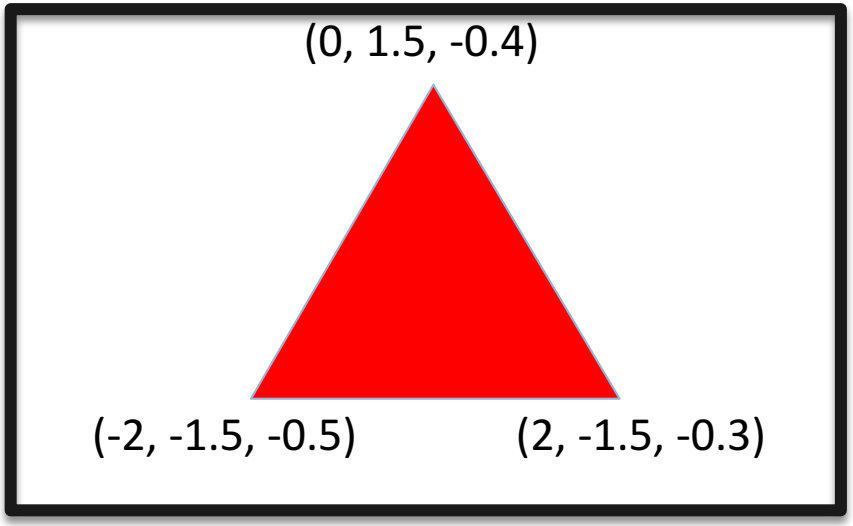
How do you sort?

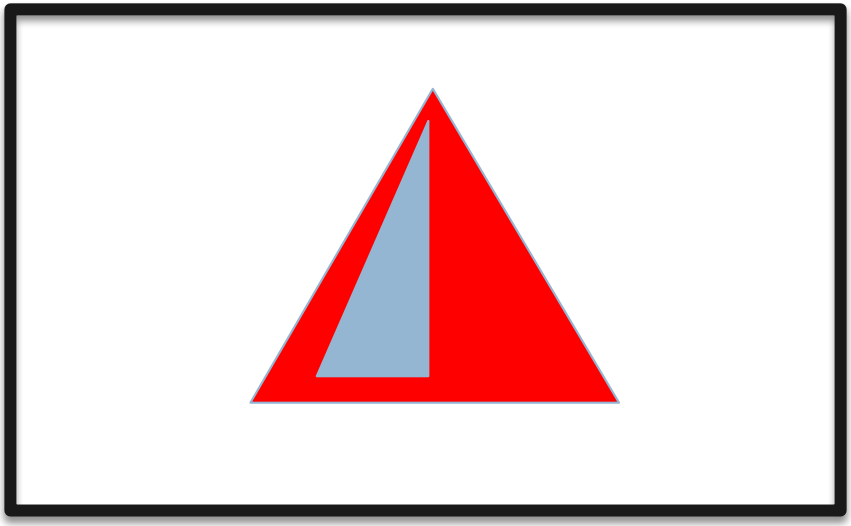
- 1) Calculate depth of each triangle center.
- 2) Sort based on depth
  - Not perfect, but good

- 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…