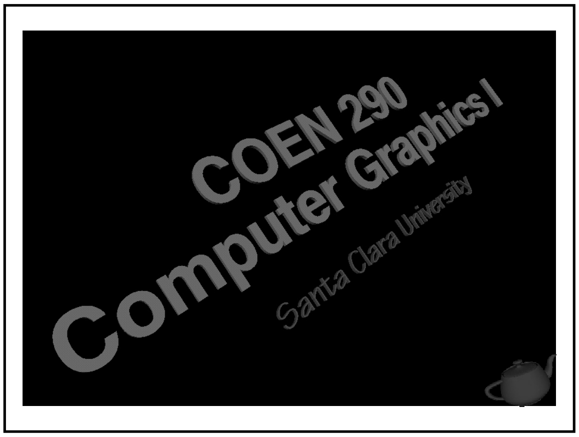
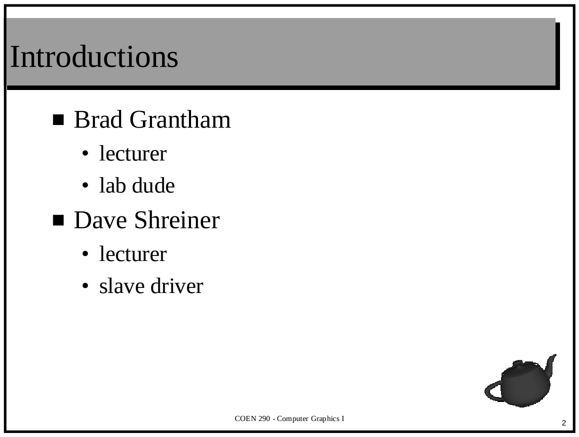


COEN 290

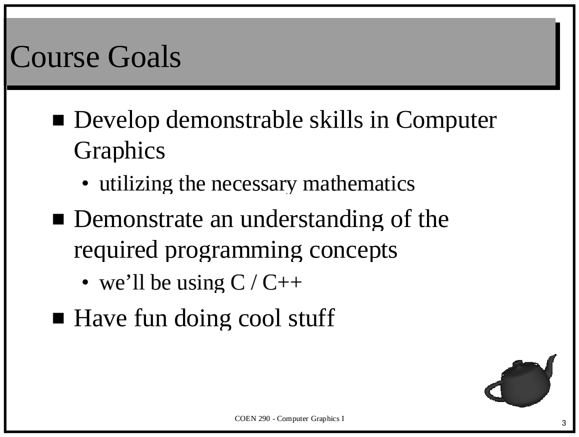
Computer Graphics I

Santa Clara University



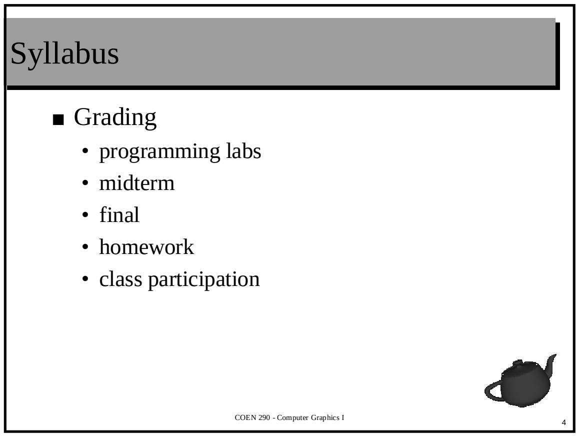
Introductions

- Brad Grantham
  - lecturer
  - lab dude
- Dave Shreiner
  - lecturer
  - slave driver



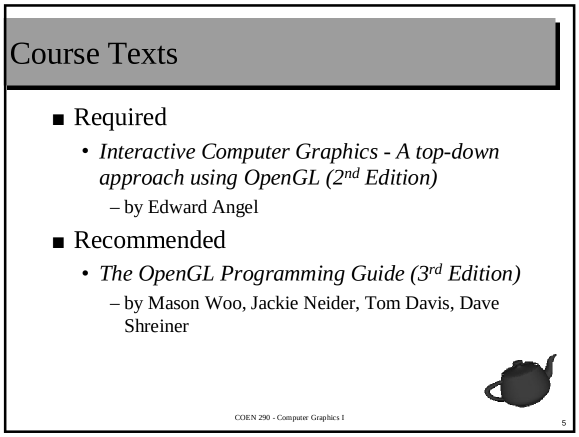
Course Goals

- Develop demonstrable skills in Computer Graphics
  - utilizing the necessary mathematics
- Demonstrate an understanding of the required programming concepts
  - we’ll be using C / C++
- Have fun doing cool stuff



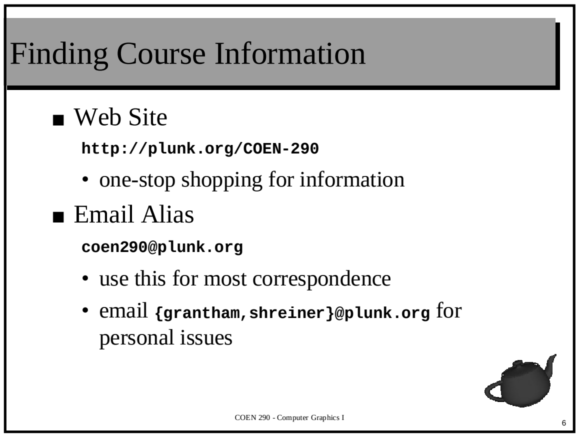
Syllabus

- Grading
  - programming labs
  - midterm
  - final
  - homework
  - class participation



Course Texts

- Required
  - *Interactive Computer Graphics - A top-down approach using OpenGL (2nd Edition)*
    - – by Edward Angel
- Recommended
  - *The OpenGL Programming Guide (3rd Edition)*
    - – by Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner

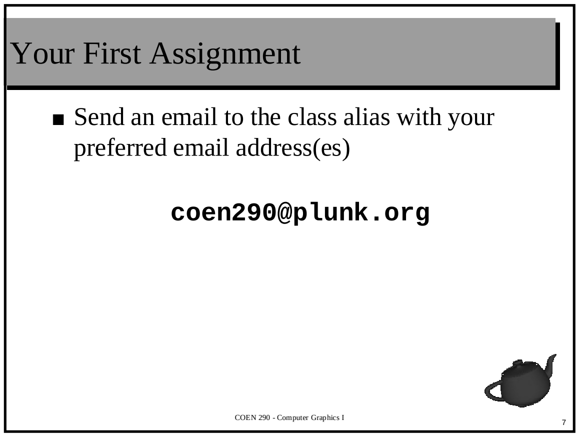


Finding Course Information

- Web Site

  `http://plunk.org/COEN-290`
  - one-stop shopping for information
- Email Alias

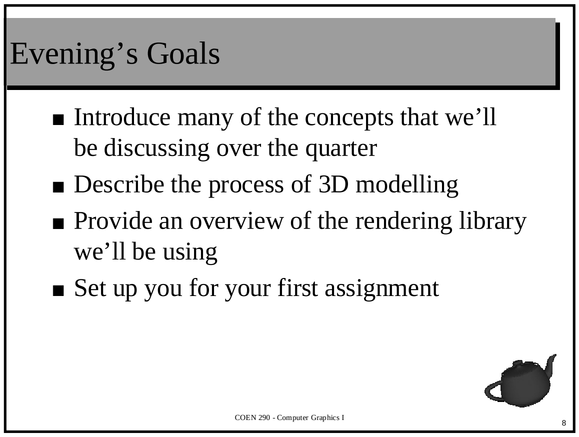
  `coen290@plunk.org`
  - use this for most correspondence
  - email `{grantham,shreiner}@plunk.org` for personal issues



Your First Assignment

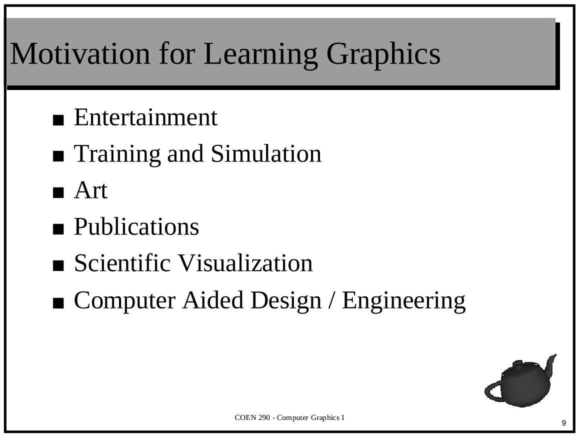
- Send an email to the class alias with your preferred email address(es)

`coen290@plunk.org`



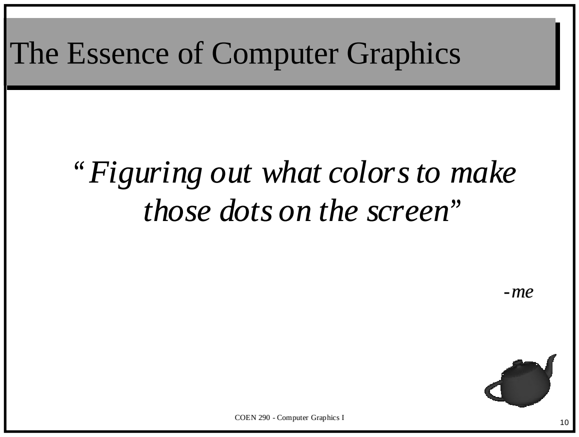
Evening’s Goals

- Introduce many of the concepts that we’ll be discussing over the quarter
- Describe the process of 3D modelling
- Provide an overview of the rendering library we’ll be using
- Set up you for your first assignment



Motivation for Learning Graphics

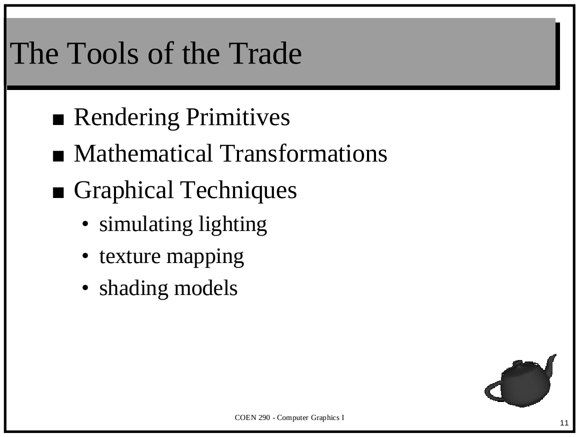
- Entertainment
- Training and Simulation
- Art
- Publications
- Scientific Visualization
- Computer Aided Design / Engineering



The Essence of Computer Graphics

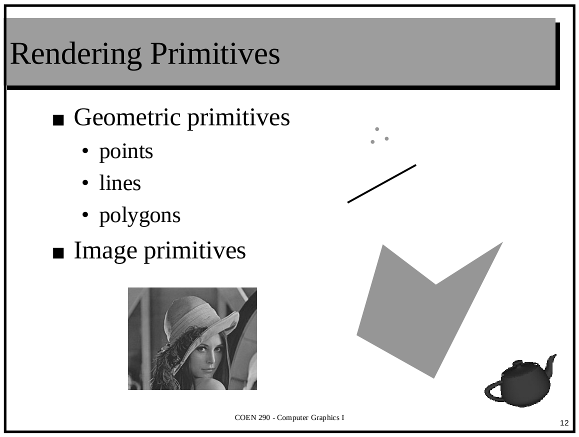
“Figuring out what colors to make those dots on the screen”

-me



The Tools of the Trade

- Rendering Primitives
- Mathematical Transformations
- Graphical Techniques
  - simulating lighting
  - texture mapping
  - shading models



Rendering Primitives

- Geometric primitives
  - points
  - lines
  - polygons
- Image primitives

Mathematical Transformations

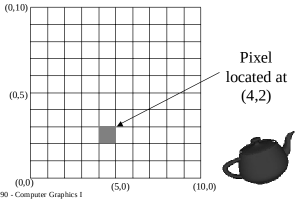
- Use transformations for moving from one coordinate space to another
- The good news
  - only requires multiplication and addition
- The bad news
  - its multiplication and addition of matrices

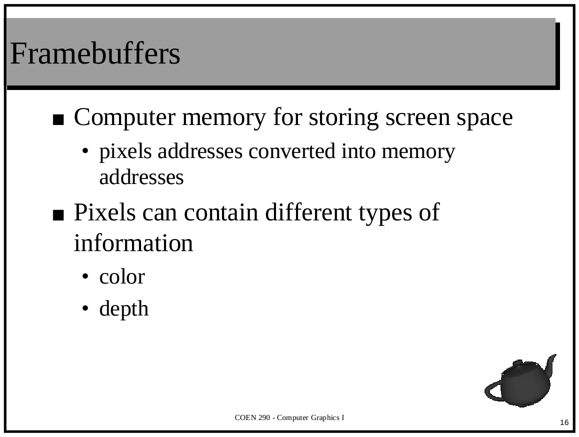
Mathematical Transformations (cont.)

- Coordinate spaces we’ll be using
  - model
  - world
  - eye
  - normalized device (NDC’s)
  - window
  - screen
  - viewport

Screen Space

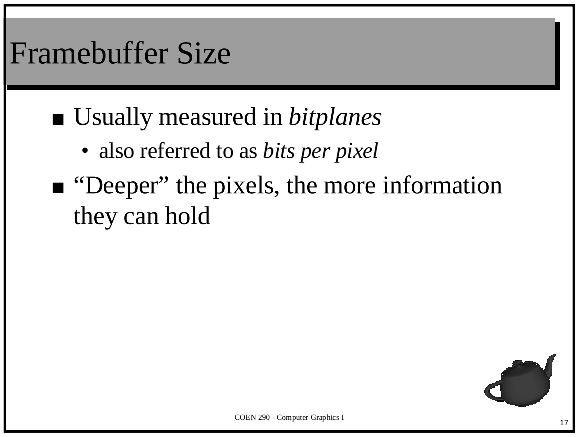
- Addressable space of your display device
  - 2 dimensional space
- Most often measured in *pixels*
  - integer addressing





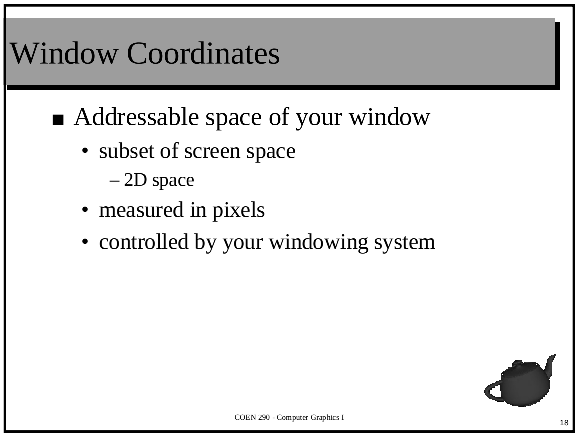
Framebuffers

- Computer memory for storing screen space
  - pixels addresses converted into memory addresses
- Pixels can contain different types of information
  - color
  - depth



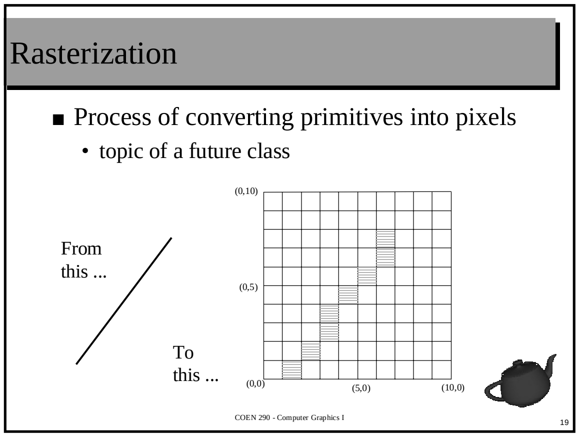
Framebuffer Size

- Usually measured in *bitplanes*
  - also referred to as *bits per pixel*
- “Deeper” the pixels, the more information they can hold



Window Coordinates

- Addressable space of your window
  - subset of screen space
    - 2D space
  - measured in pixels
  - controlled by your windowing system

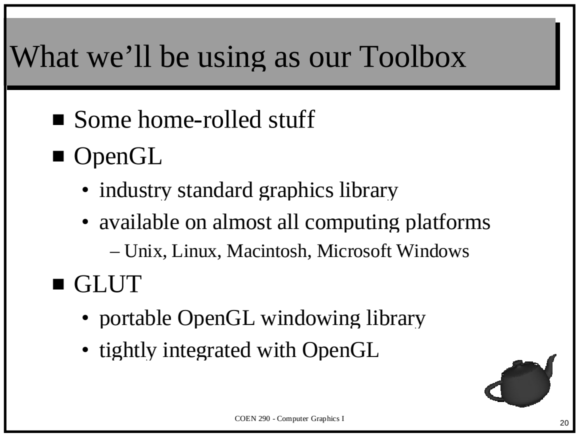


Rasterization

- Process of converting primitives into pixels
  - topic of a future class

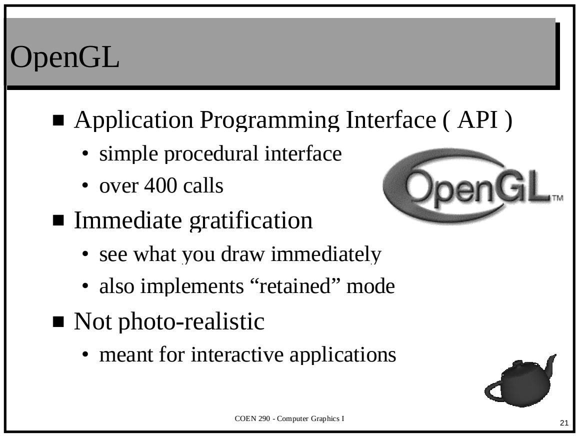
From this ...

To this ...



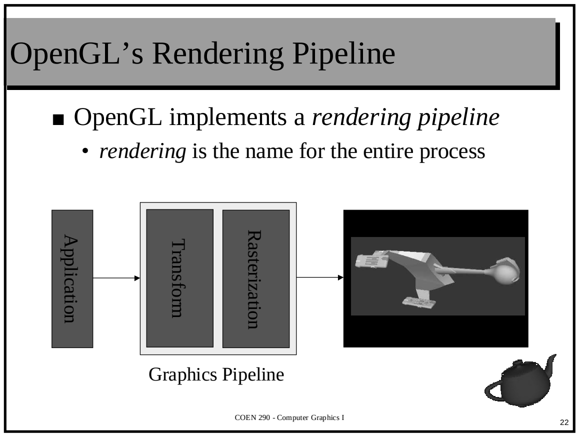
What we’ll be using as our Toolbox

- Some home-rolled stuff
- OpenGL
  - industry standard graphics library
  - available on almost all computing platforms
    - Unix, Linux, Macintosh, Microsoft Windows
- GLUT
  - portable OpenGL windowing library
  - tightly integrated with OpenGL



OpenGL

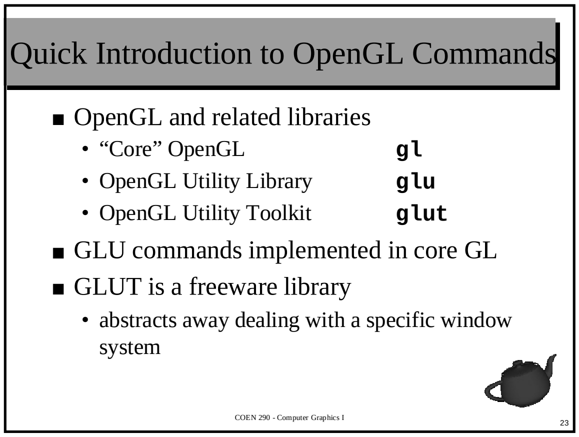
- Application Programming Interface (API)
  - simple procedural interface
  - over 400 calls
- Immediate gratification
  - see what you draw immediately
  - also implements “retained” mode
- Not photo-realistic
  - meant for interactive applications



OpenGL’s Rendering Pipeline

- OpenGL implements a *rendering pipeline*
  - *rendering* is the name for the entire process

Graphics Pipeline

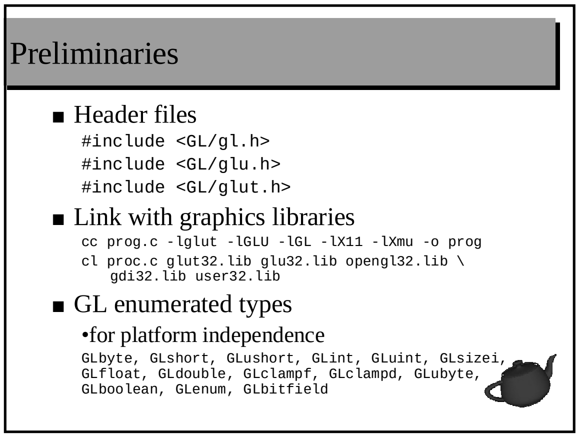


Quick Introduction to OpenGL Commands

- OpenGL and related libraries

• “Core” OpenGL	**`gl`**
• OpenGL Utility Library	**`glu`**
• OpenGL Utility Toolkit	**`glut`**

- GLU commands implemented in core GL
- GLUT is a freeware library
  - abstracts away dealing with a specific window system



Preliminaries

- Header files

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
```

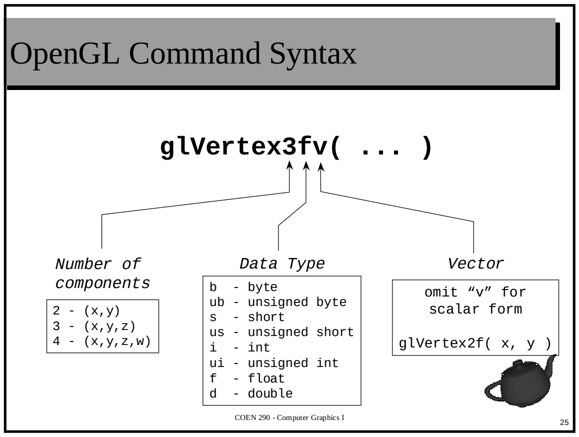
- Link with graphics libraries

```
cc prog.c -lglut -lGLU -lGL -lX11 -lXmu -o prog
cl proc.c glut32.lib glu32.lib opengl32.lib \
   gdi32.lib user32.lib
```

- GL enumerated types

•for platform independence

```
GLbyte, GLshort, GLushort, GLint, GLuint, GLsizei,
GLfloat, GLdouble, GLclampf, GLclampd, GLubyte,
GLboolean, GLenum, GLbitfield
```



OpenGL Command Syntax

`glVertex3fv( ... )`

Number of components

```
2 - (x,y)
3 - (x,y,z)
4 - (x,y,z,w)
```

Data Type

```
b  - byte
ub - unsigned byte
s  - short
us - unsigned short
i  - int
ui - unsigned int
f  - float
d  - double
```

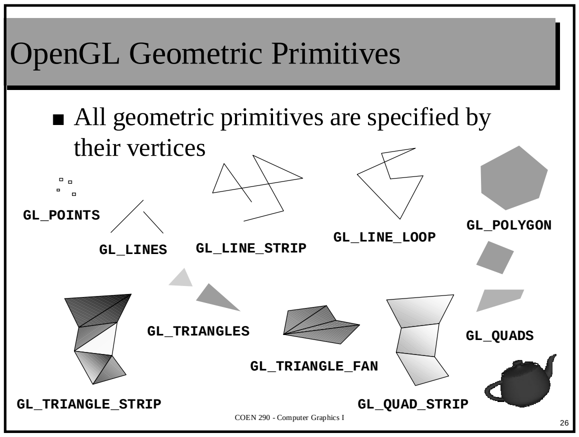
Vector

omit "v" for scalar form

`glVertex2f( x, y )`

COEN 290 - Computer Graphics I

25

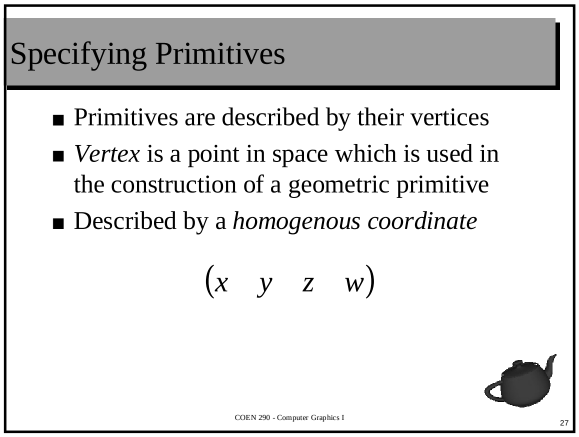


OpenGL Geometric Primitives

- All geometric primitives are specified by their vertices

COEN 290 - Computer Graphics I

26



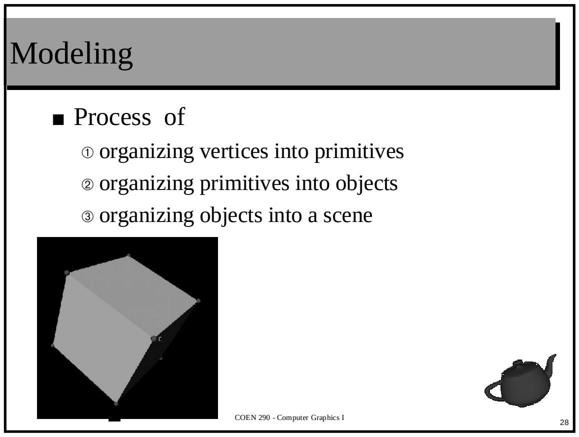
Specifying Primitives

- Primitives are described by their vertices
- *Vertex* is a point in space which is used in the construction of a geometric primitive
- Described by a *homogenous coordinate*

$$(x \quad y \quad z \quad w)$$

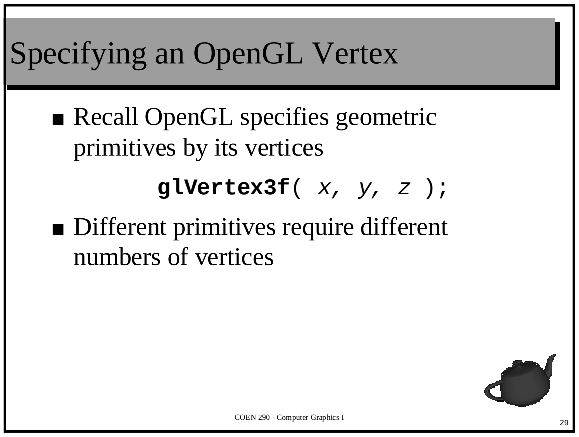
COEN 290 - Computer Graphics I

27



Modeling

- Process of
  - ① organizing vertices into primitives
  - ② organizing primitives into objects
  - ③ organizing objects into a scene

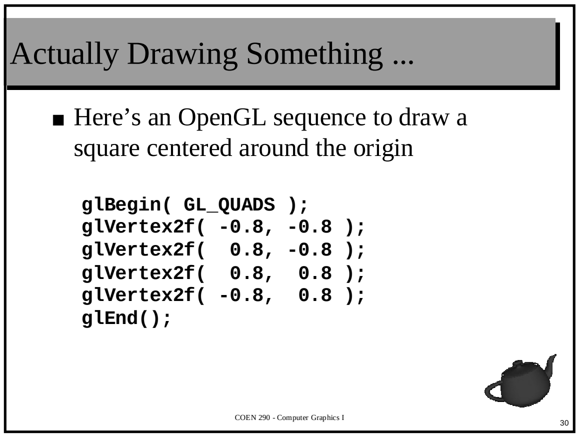


Specifying an OpenGL Vertex

- Recall OpenGL specifies geometric primitives by its vertices

```
glVertex3f( x, y, z );
```

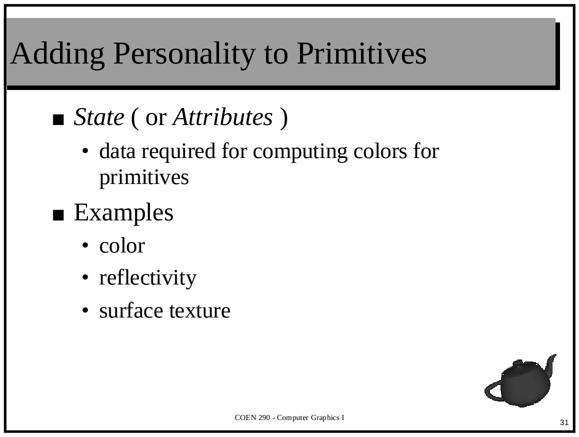
- Different primitives require different numbers of vertices



Actually Drawing Something ...

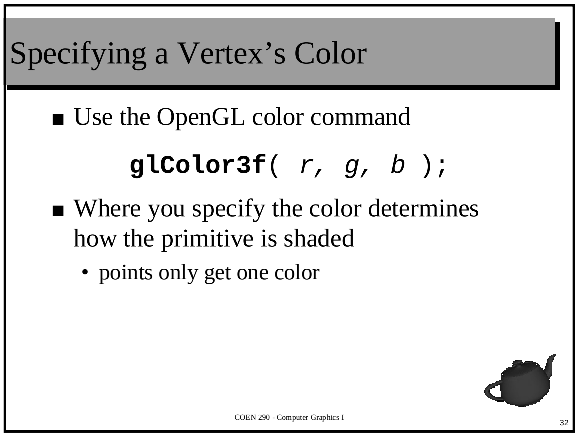
- Here’s an OpenGL sequence to draw a square centered around the origin

```
glBegin( GL_QUADS );
glVertex2f( -0.8, -0.8 );
glVertex2f(  0.8, -0.8 );
glVertex2f(  0.8,  0.8 );
glVertex2f( -0.8,  0.8 );
glEnd();
```



Adding Personality to Primitives

- *State* (or *Attributes*)
  - data required for computing colors for primitives
- Examples
  - color
  - reflectivity
  - surface texture

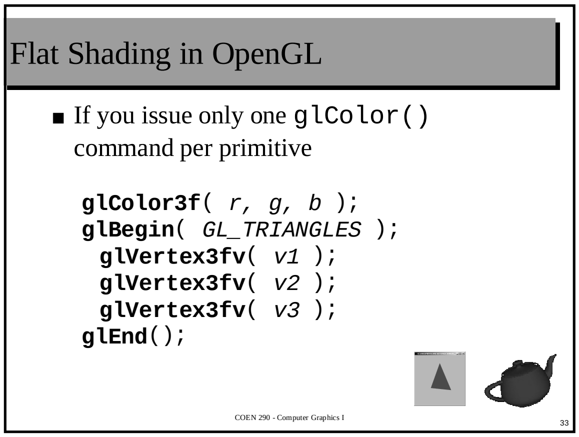


Specifying a Vertex's Color

- Use the OpenGL color command

```
glColor3f( r, g, b );
```

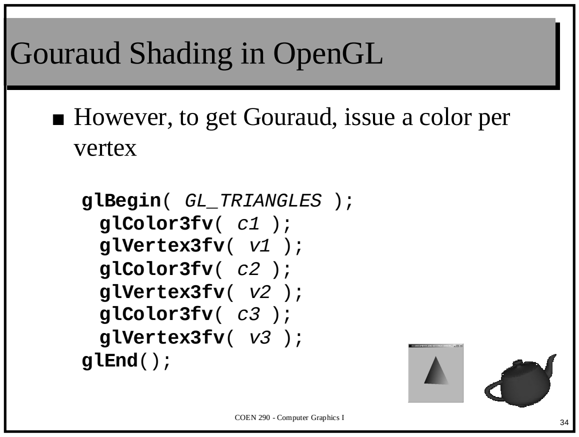
- Where you specify the color determines how the primitive is shaded
  - points only get one color



Flat Shading in OpenGL

- If you issue only one `glColor()` command per primitive

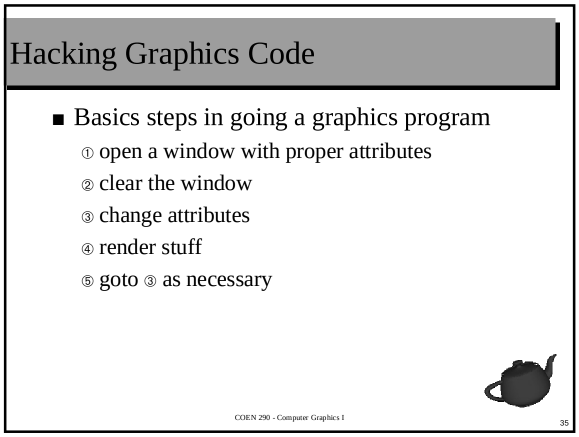
```
glColor3f( r, g, b );
glBegin( GL_TRIANGLES );
  glVertex3fv( v1 );
  glVertex3fv( v2 );
  glVertex3fv( v3 );
glEnd();
```



Gouraud Shading in OpenGL

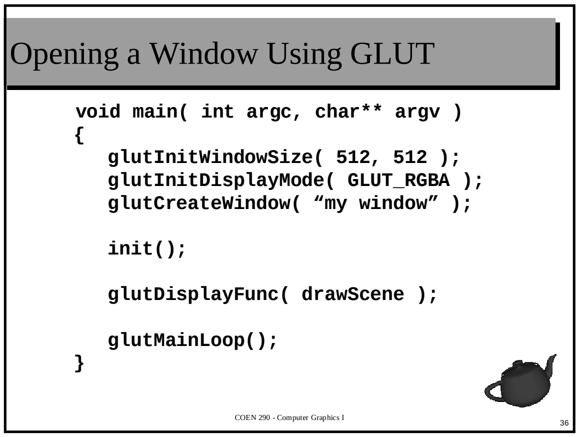
- However, to get Gouraud, issue a color per vertex

```
glBegin( GL_TRIANGLES );
  glColor3fv( c1 );
  glVertex3fv( v1 );
  glColor3fv( c2 );
  glVertex3fv( v2 );
  glColor3fv( c3 );
  glVertex3fv( v3 );
glEnd();
```



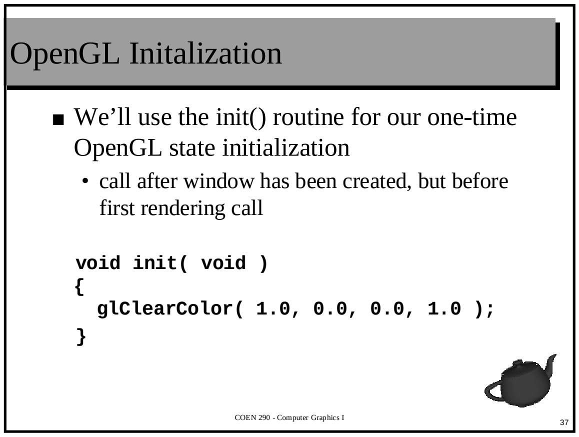
Hacking Graphics Code

- Basics steps in going a graphics program
  - ① open a window with proper attributes
  - ② clear the window
  - ③ change attributes
  - ④ render stuff
  - ⑤ goto ③ as necessary



Opening a Window Using GLUT

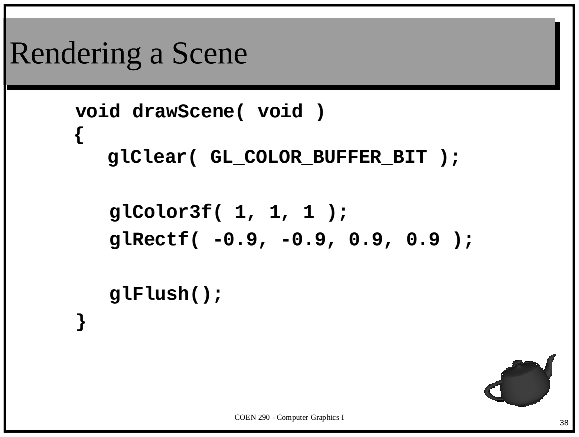
```
void main( int argc, char** argv )
{
   glutInitWindowSize( 512, 512 );
   glutInitDisplayMode( GLUT_RGBA );
   glutCreateWindow( “my window” );

   init();

   glutDisplayFunc( drawScene );

   glutMainLoop();
}
```



OpenGL Initalization

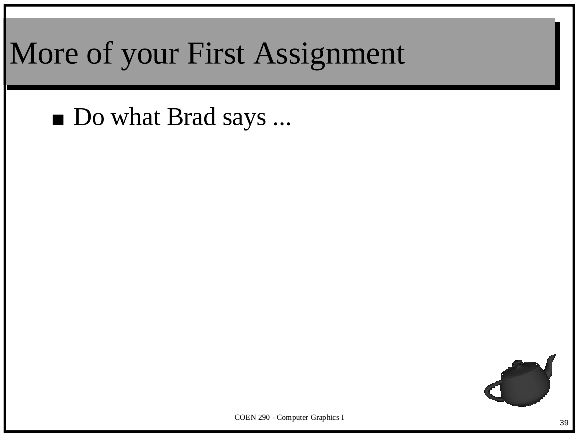
- We’ll use the init() routine for our one-time OpenGL state initialization
  - call after window has been created, but before first rendering call

```
void init( void )
{
  glClearColor( 1.0, 0.0, 0.0, 1.0 );
}
```



Rendering a Scene

```
void drawScene( void )
{
   glClear( GL_COLOR_BUFFER_BIT );

   glColor3f( 1, 1, 1 );
   glRectf( -0.9, -0.9, 0.9, 0.9 );

   glFlush();
}
```



More of your First Assignment

- Do what Brad says ...