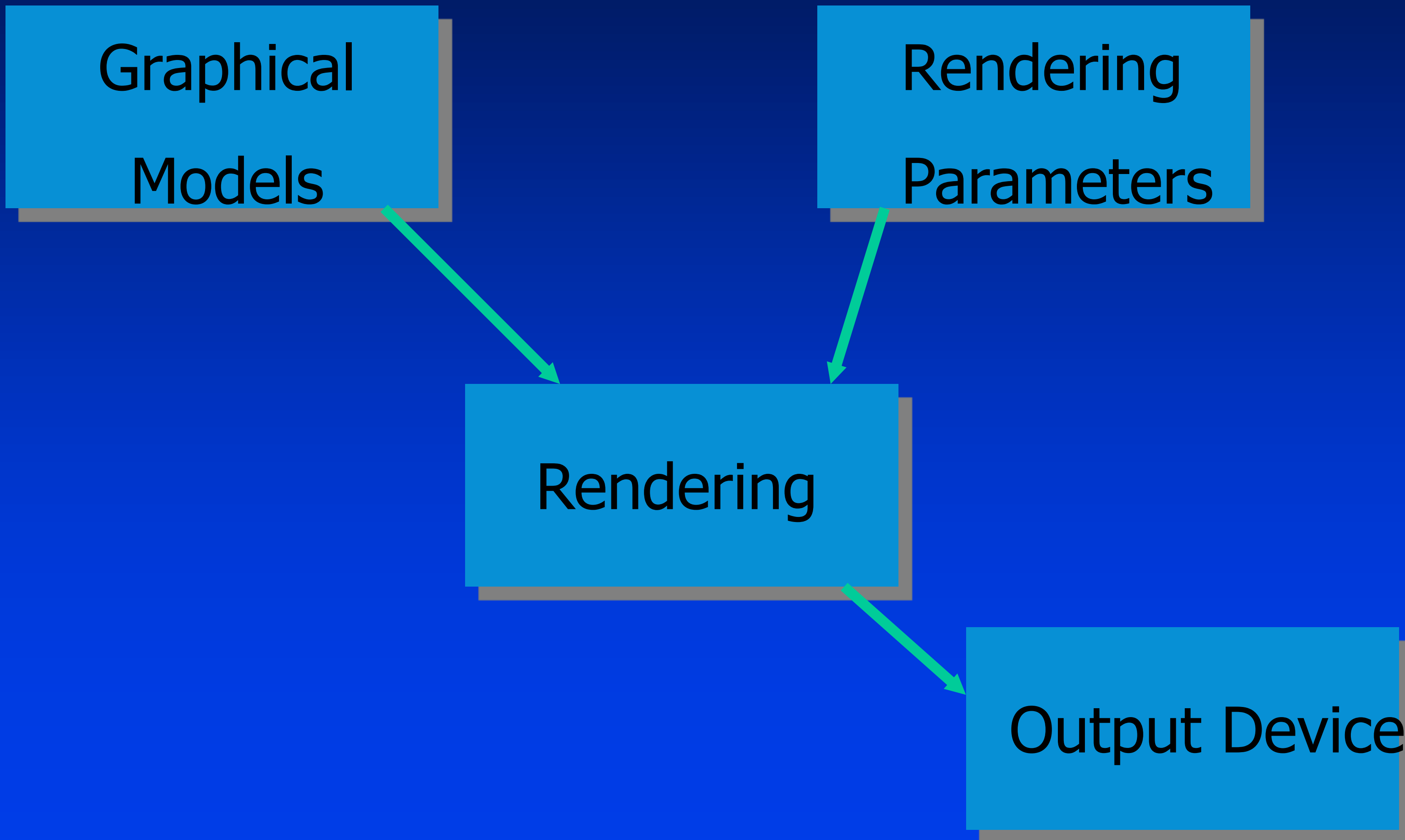
CSE528 Computer Graphics: Theory, Algorithms, and Applications

Hong Qin
Department of Computer Science
Stony Brook University (SUNY at Stony Brook)
Stony Brook, New York 11794-2424
Tel: (631)632-8450; Fax: (631)632-8334
qin@cs.stonybrook.edu
http://www.cs.stonybrook.edu/~qin

Computer Graphics

- (Realistic) pictorial synthesis of real and/or imaginary objects from their computer-based models (datasets)
- It typically includes modeling, rendering (graphics pipeline), and human-computer interaction
- So, we are focusing on computer graphics hardware, software, and mathematical foundations
- Computer Graphics is computation
  - A new method of visual computing
- Why is Computer Graphics useful and important?
- Course challenges: more mathematics oriented, programming requirements, application-driven, inter-disciplinary in nature, etc.

Computer Graphics Systems



Output Devices

- Vector Devices
  - Lasers (for example)

- Raster Devices
  - CRT, LCD, bitmaps, etc.

  - Most output devices are 2D
  - Can you name any 3D output device?

Graphical Models

- 2D and 3D objects
  - Triangles, quadrilaterals, polygons
  - Spheres, cones, boxes
- Surface characteristics
  - Color, reaction to light
  - Texture, material properties
- Composite objects
  - Other objects and their relationships to each other
- Lighting, fog, etc.
- Much, much more…

Rendering

- Conversion of 3D model to 2D image
  - Determine where the surfaces “project” to
  - Determine what every screen pixel might see
  - Determine the color of each surface

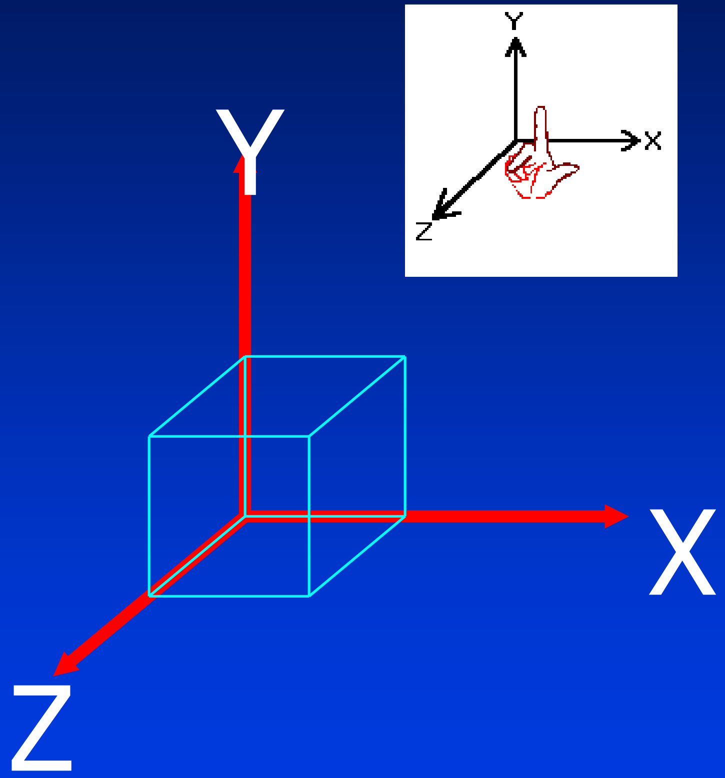
Rendering Parameters

- Camera parameters
  - Location
  - Orientation
  - Focal length

2D Graphics vs. 3D Graphics

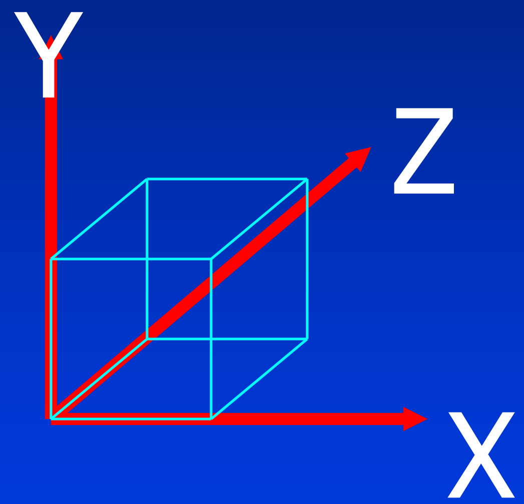
- 2D
  - X, Y - 2 dimensions only
  - We won't spend time on 2D graphics in this course
- 3D
  - X, Y, and Z
  - Space
- Rendering is typically the conversion of 3D to 2D

3D Coordinate Systems



Right-Hand Coordinate System

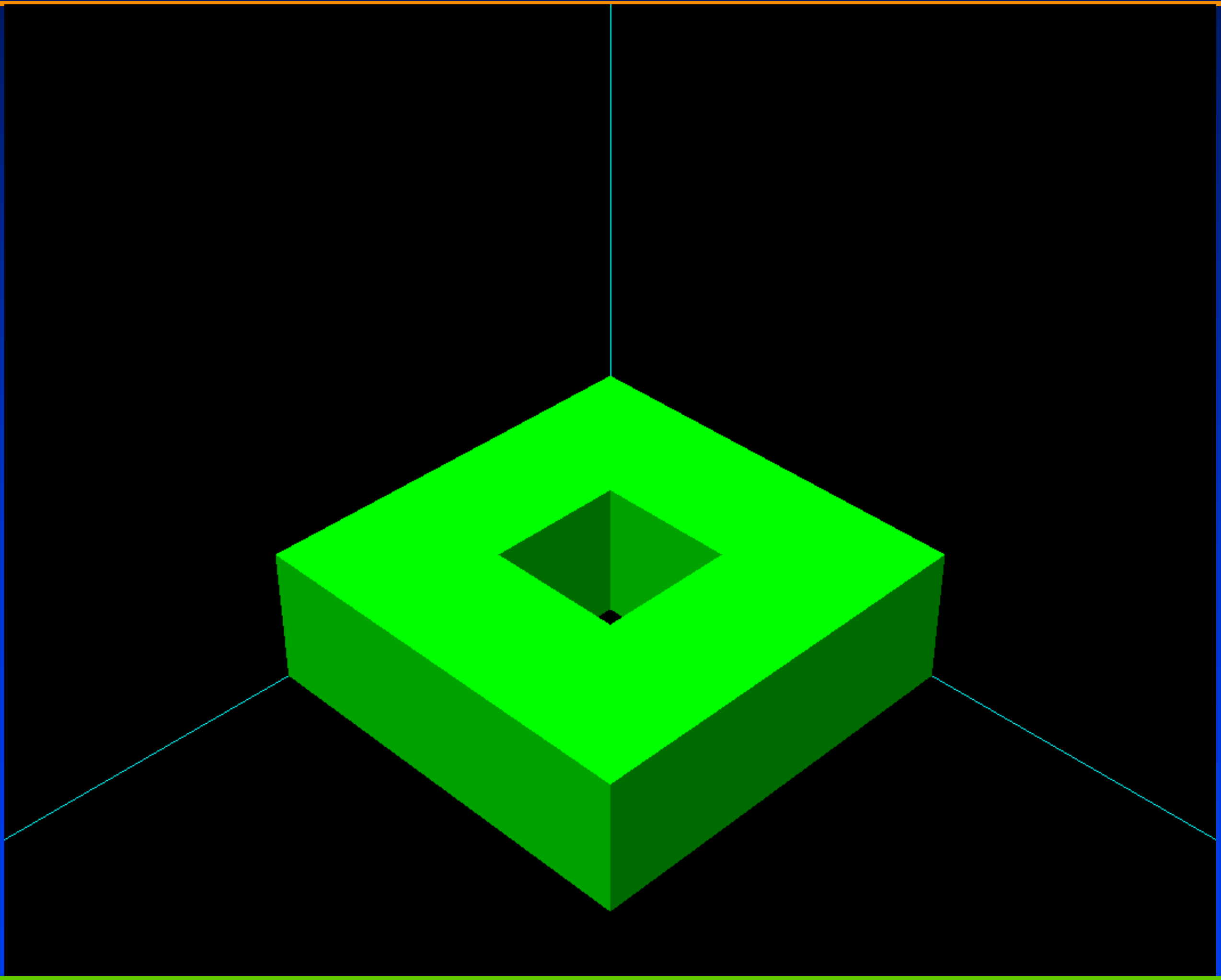
OpenGL uses this!



Left-Hand Coordinate System

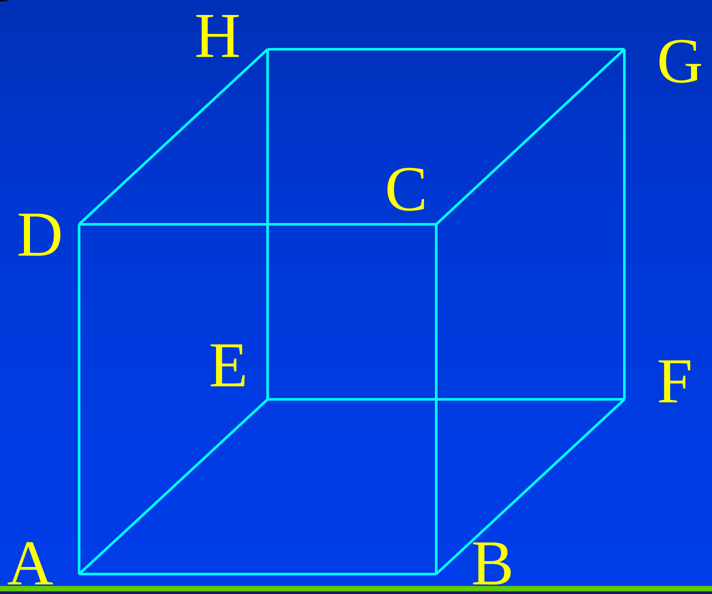
Direct3D uses this!

How to Model/Render This?

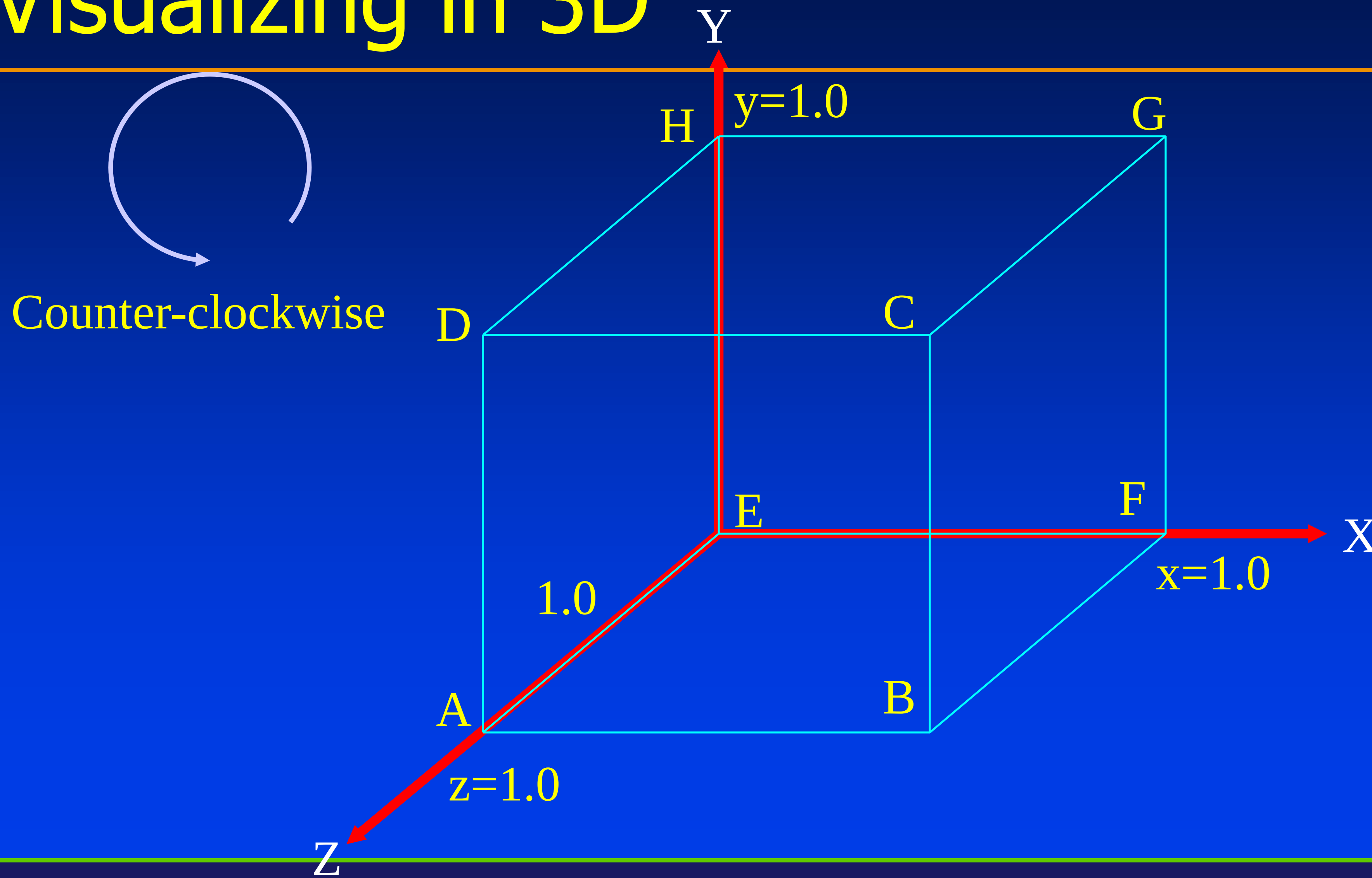


Render/Display a Box in OpenGL

- We render the 6 faces as polygons
  - Polygons are specified as a list of vertices
  - Vertices are specified in counter-clockwise order looking at the surface of the face!

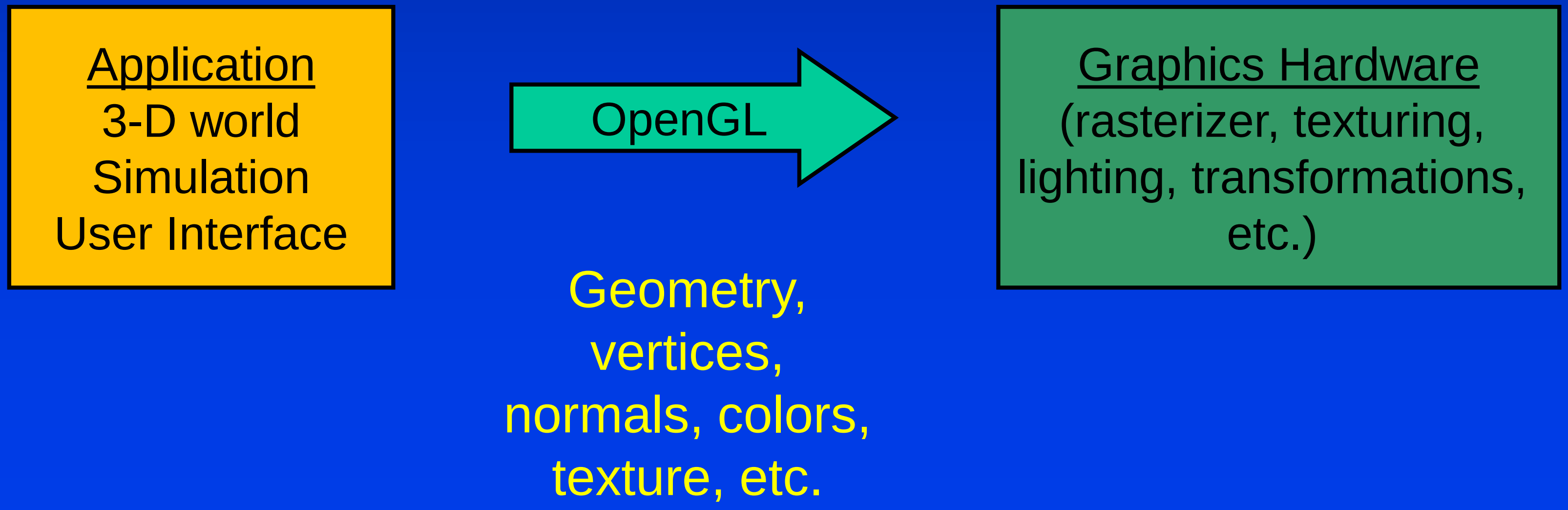


Visualizing in 3D



OpenGL

- OpenGL is a software interface to graphics hardware
- Most widely used 3D graphics application program interface (API).



OpenGL Basics

- Truly open, independent of system platforms.
- Reliable, easy to use and well-documented.
- Default language is C/C++.
- Many online resources are currently available (explore them and use them)!
- OpenGL is a STATE MACHINE: polygons are affected by the current color, transformation, drawing mode, etc.

OpenGL Conventions

- OpenGL is a retained mode graphics system
  - It has a state
  - For example, glBegin(GL_POLYGON) puts us into a polygon rendering state
- C library
  - All function names start with gl

Specifying Vertices for Objects

- Objects are represented by vertices
  - `glVertex3f (2.0, 4.1, 6.0);`
  - `glVertex2i (4, 5);`
  - `glVertex3fv (vector);`
- Current color affects any vertices
  - `glColor3f (0.0, 0.5, 1.0);`
  - `glColor4ub (0, 128, 255, 0);`
  - `glColor3dv (color);`

2D Drawing Primitives

```
glBegin(GL_POLYGON);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 3.0);
    glVertex2f(3.0, 3.0);
    glVertex2f(4.0, 1.5);
    glVertex2f(3.0, 0.0);
glEnd();
```

OpenGL Polygon Rendering

```
GLdouble size = 1.0;

glBegin(GL_POLYGON);        // front face
   glVertex3d(0.0,   0.0,   size);
   glVertex3d(size,  0.0,   size);
   glVertex3d(size,  size, size);
   glVertex3d(0.0,   size,  size);
glEnd();
```

OpenGL Types

- Basic numeric types
  - GLdouble = double
  - GLfloat = float
  - GLint = int
  - GLshort = short
- Mostly, you'll use GLdouble and GLfloat

Defined glVertex3fv

Prefix	Function	# Parms	Type	Suffix
gl	Vertex	1	f (float)	v (vector)
glu	Begin	2	d (double)	
wgl	End	3	i (integer)	
agl	Lighting	4	b (byte)	
	…	…	s (short)	

Only if varying arguments

Function Suffixes

- Many functions have alternatives
  - Alternatives are specified by the suffix
  - **glVertex2d**
    - **2 double parameters**
    - **void glVertex2d(GLdouble x, GLdouble y);**
  - **glVertex3f**
    - **3 float parameters**
    - **void glVertex3f(GLfloat x, GLfloat y, GLfloat z);**
  - **glVertex3fv**
    - **void glVertex3fv(const GLfloat *v);**

All of Them…

- **glVertex2d, glVertex2f, glVertex2i, glVertex2s, glVertex3d, glVertex3f, glVertex3i, glVertex3s, glVertex4d, glVertex4f, glVertex4i, glVertex4s, glVertex2dv, glVertex2fv, glVertex2iv, glVertex2sv, glVertex3dv, glVertex3fv, glVertex3iv, glVertex3sv, glVertex4dv, glVertex4fv, glVertex4iv, glVertex4sv**

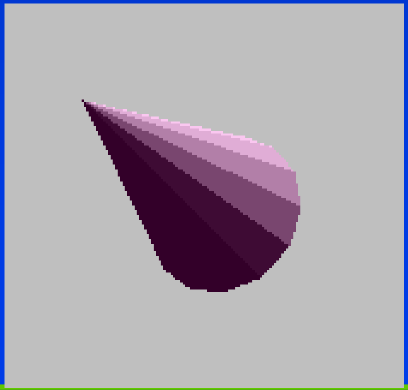
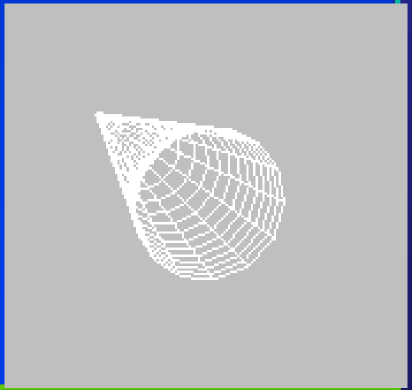
Specifying Objects' Vertices

- Vertices are specified only between `glBegin(mode)` and `glEnd()`, usually in a counter-clockwise order for polygons.

```
glBegin (GL_TRIANGLES);
      glVertex2i (0, 0);
      glVertex2i (2, 0);
      glVertex2i (1, 1);
glEnd();
```

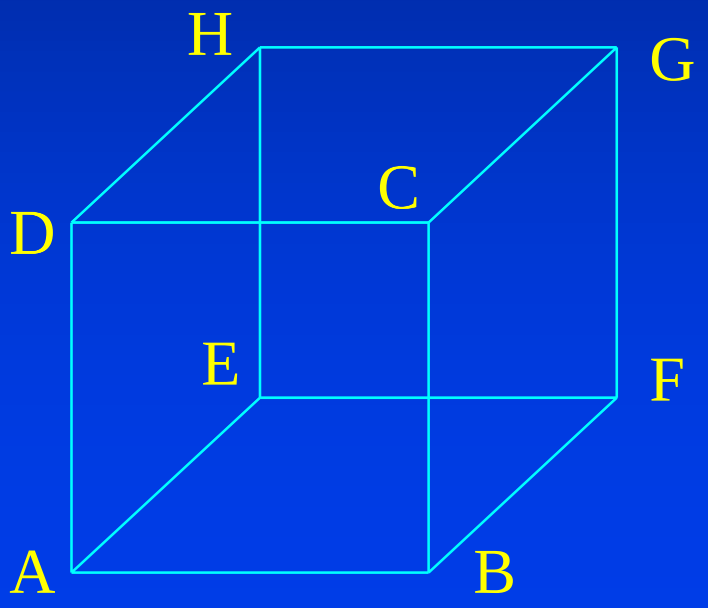
Primitive Types

- Points: GL_POINTS
- Lines: GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP
- Triangles: GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN
- Quads: GL_QUADS, GL_QUAD_STRIP
- Polygons: GL_POLYGON

Vector Parameters

```
GLdouble a[ ] = {0, 0, 1};
GLdouble b[ ] = {1, 0, 1};
GLdouble c[ ] = {1, 1, 1};
GLdouble d[ ] = {0, 1, 1};

glBegin(GL_POLYGON);              // front face
   glVertex3dv(a);
   glVertex3dv(b);
   glVertex3dv(c);
   glVertex3dv(d);
glEnd();
```



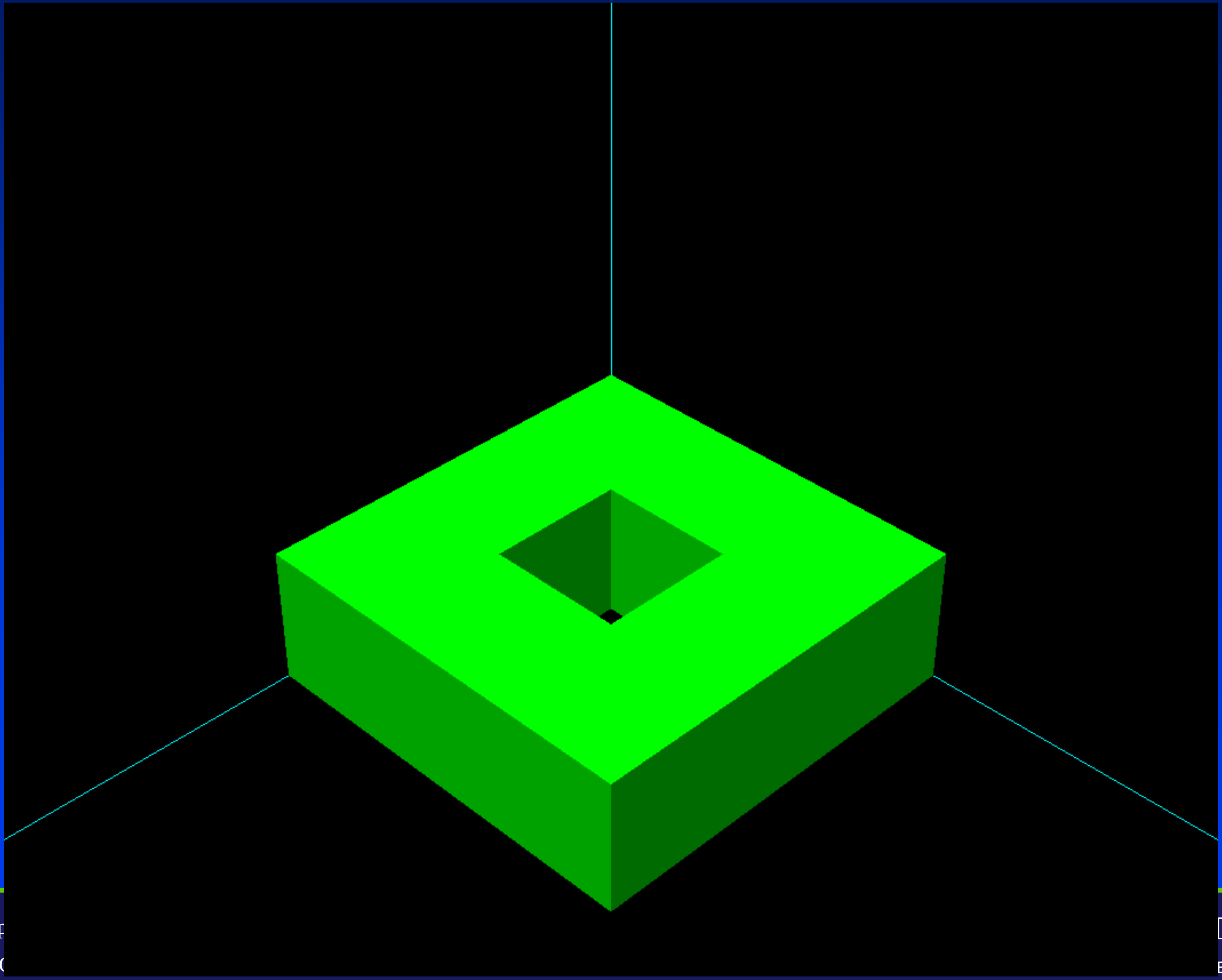
Specify a Color (No Lighting)

- glColor3f(red, green, blue);
- Most of the same suffixes apply…

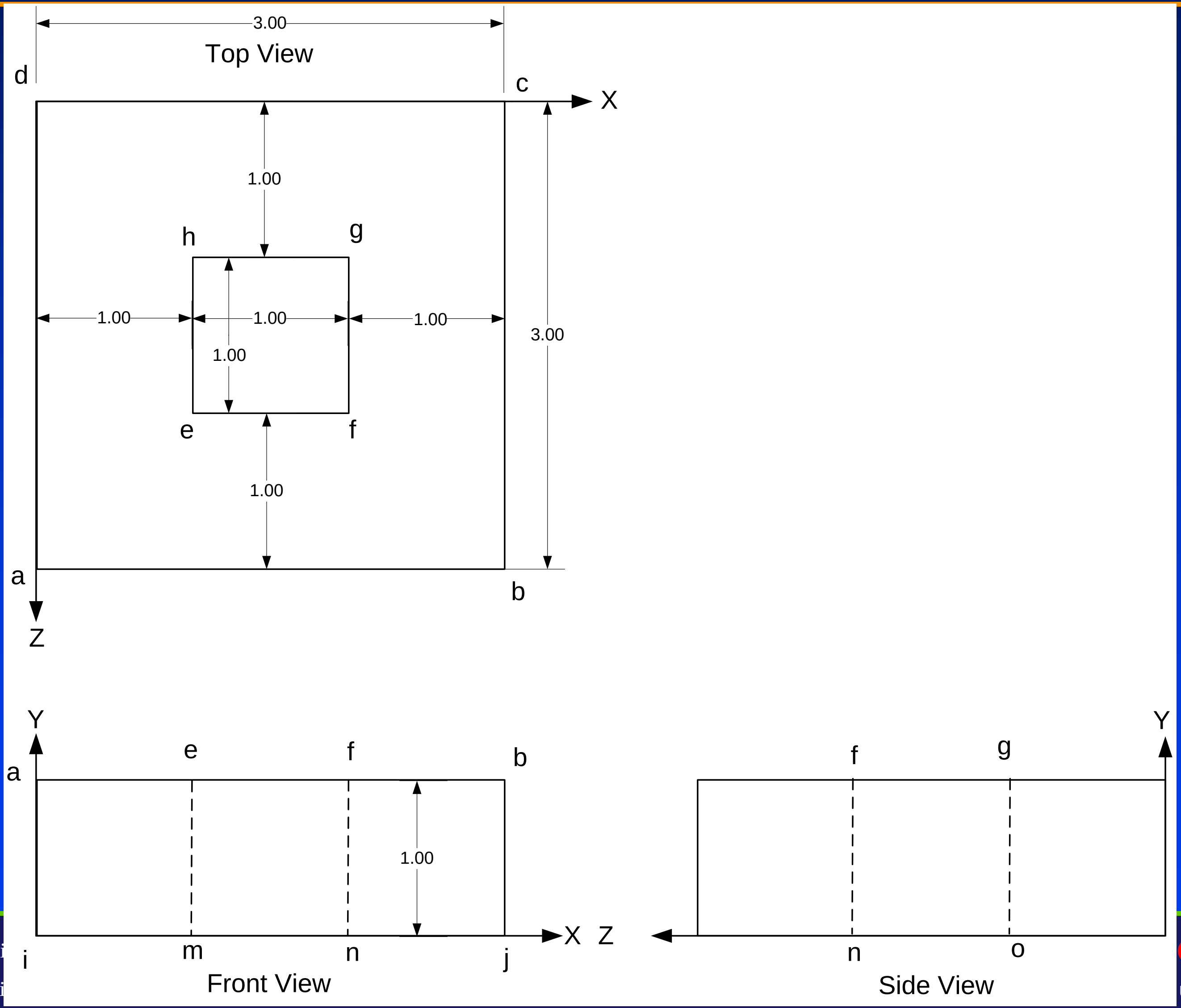
```
GLdouble size = 1.0;

glColor3d(1.0, 0.0, 0.0);               // red
glBegin(GL_POLYGON);  // front face
    glVertex3d(0.0,   0.0,   size);
    glVertex3d(size,  0.0,   size);
    glVertex3d(size,  size, size);
    glVertex3d(size,  0.0,   size);
glEnd();
```

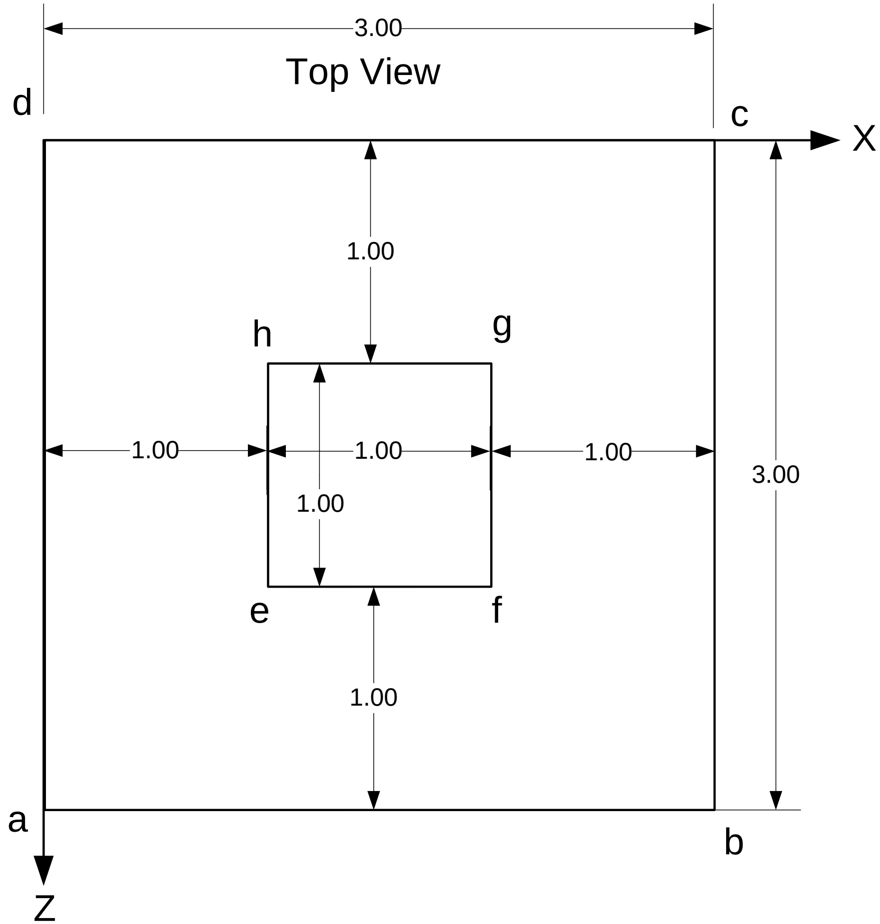
Colors range from 0 to 1

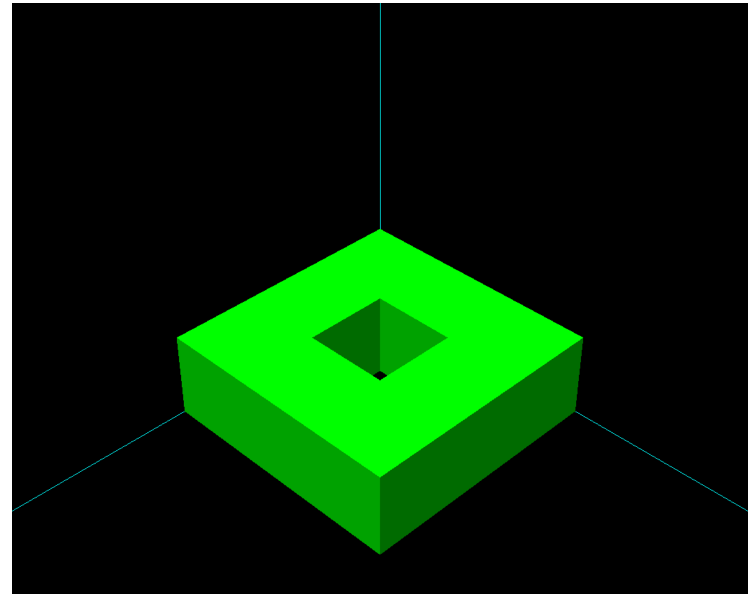
How to Model/Render This?

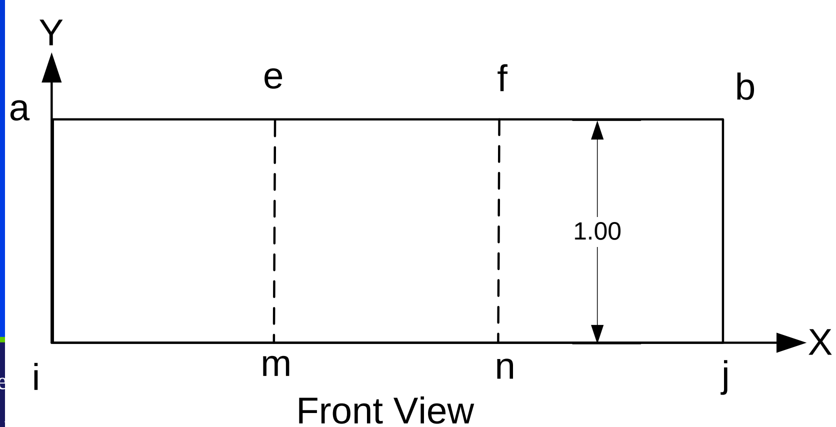


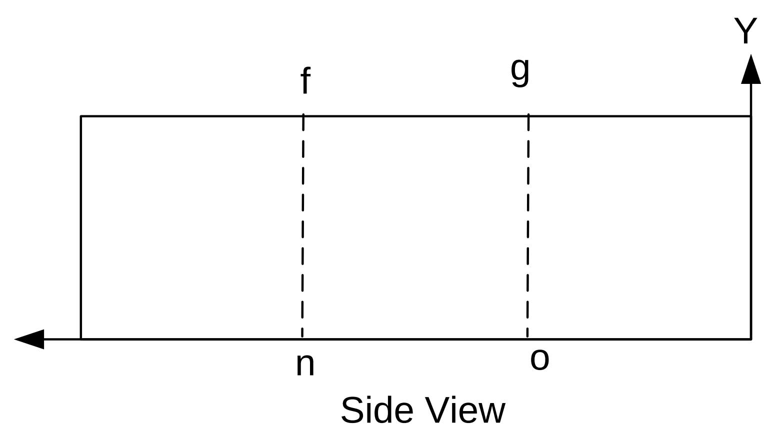
2D Views


3.00
Top View
d
c
X
1.00
h
g
1.00
1.00
1.00
1.00
3.00
e
f
1.00
a
b
Z
Y
a
e
f
b
1.00
X
i
m
n
j
Front View
Z
f
g
Y
n
o
Side View

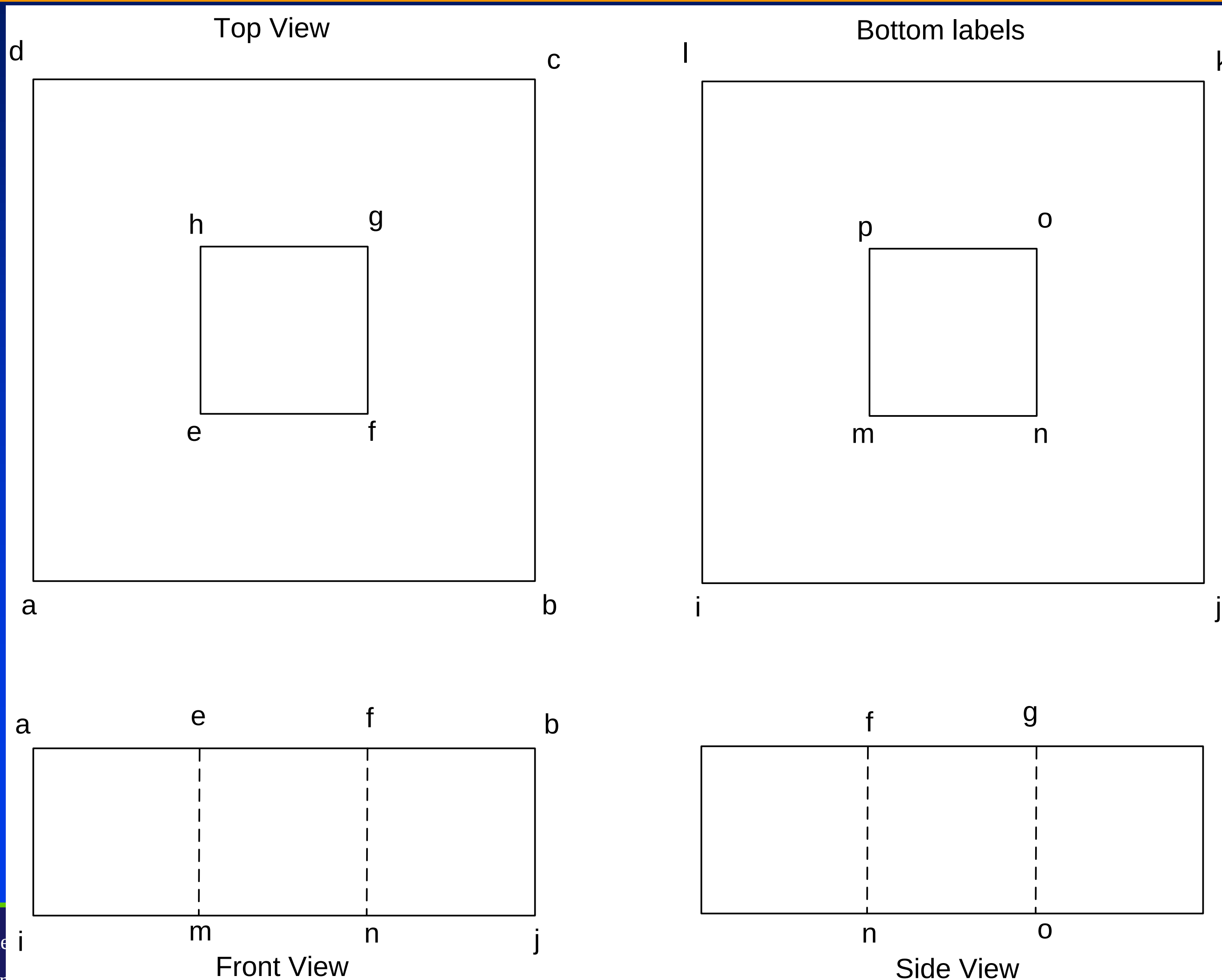

3.00
Top View
d
c
X
1.00
h
g
1.00
1.00
1.00
3.00
1.00
e
f
1.00
a
b
Z




Y
a
e
f
b
1.00
i
m
n
j
X
Front View


Y
f
g
Z
n
o
Side View

Vertices' Labels



The Basic Idea

- Describe an object using surfaces
- Surfaces are polygons
  – Triangles, quadrilaterals, whatever
  – Important thing is that they are flat
  – They must also be convex
- Provide points in counter-clockwise order
  – From the visible side

Transformation and Viewing

OpenGL has 3 different matrix modes:

- **GL_MODELVIEW**
- **GL_PROJECTION**
- **GL_TEXTURE**

- Choose the matrix with:

```
glMatrixMode(…);
```

Transforms Objects within the Scene

- Modelview matrix

Set up Perspective Projection

Projection matrix

- `glFrustrum (...);`
- `gluPerspective (fovy, aspect, near, far);`
- `glOrtho (...);`
- `gluLookAt (...);`

Example

- Projection Matrix

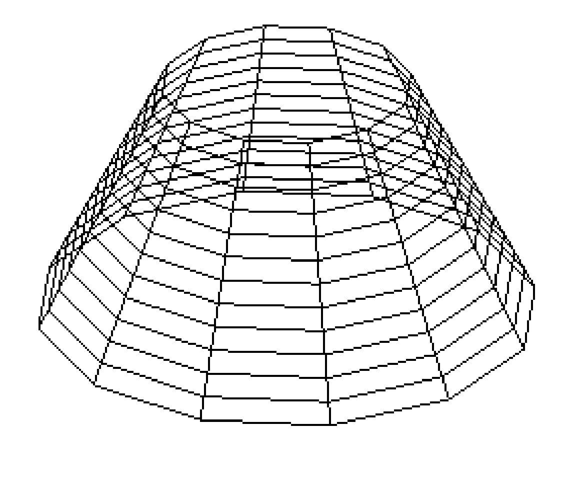
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(64, (float)windowWidth /
                    (float)windowHeight, 4, 4096);
gluLookAt(0.0, 0.0, 2.0, // camera position
          0.0, 0.0, 0.0, // target position
          0.0, 0.0, 2.0); // up vector
```

OpenGL Extensions

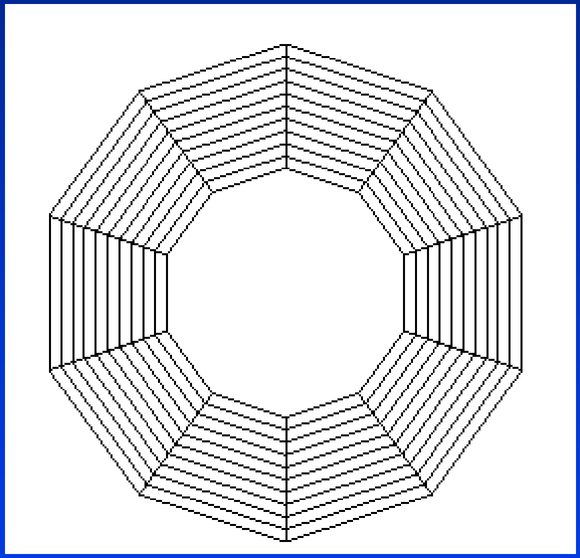
- The **GL** library is the core OpenGL system:
  - modeling, viewing, lighting, clipping
- The **GLU** library (GL Utility) simplifies common tasks:
  - creation of common objects (e.g. spheres, quadrics)
  - specification of standard views (e.g. perspective, orthographic)
- The **GLUT** library (GL Utility Toolkit) provides the interface with the window system.
  - window management, menus, mouse interaction

Defining Cylinder

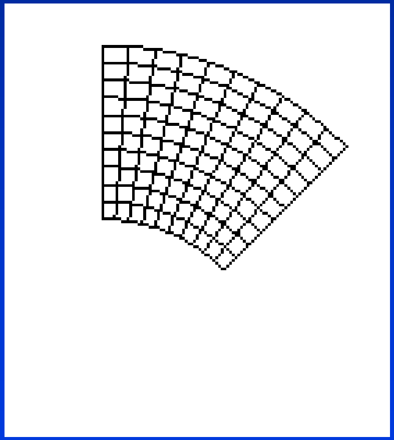
```
GLUquadricOBJ *p;
P = gluNewQuadric(); /*set up object */
gluQuadricDrawStyle(GLU_LINE);/*render
  style*/
gluCylinder(p, BASE_RADIUS, TOP_RADIUS,
           BASE_HEIGHT, sections, slices);
```



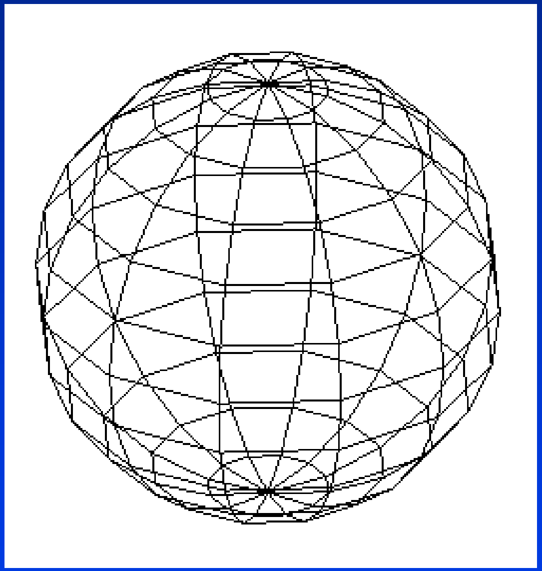
Quadric Objects in GLU



disk



partial disk

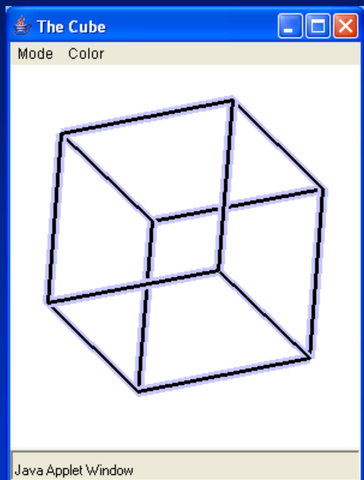


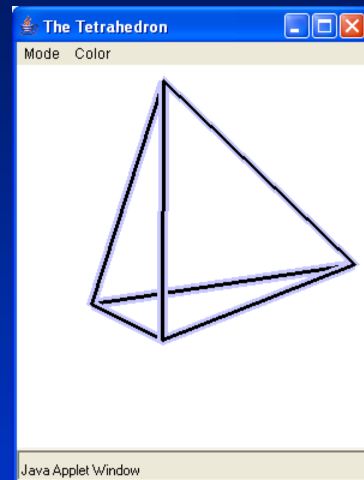
sphere

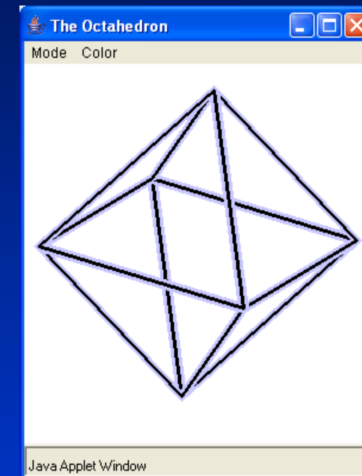
Platonic Solids

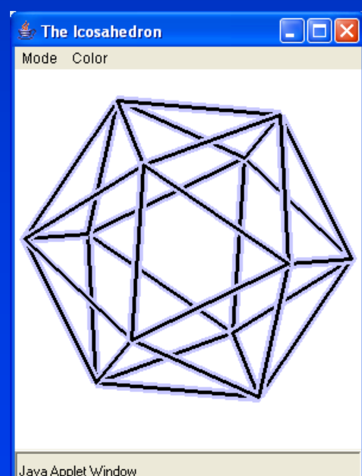
- Also known as the regular solids or regular polyhedra
- Convex polyhedra with equivalent faces composed of congruent regular polygons
- There are five such solids:
  - Cube
  - Dodecahedron
  - Icosahedron
  - Octahedron
  - Tetrahedron

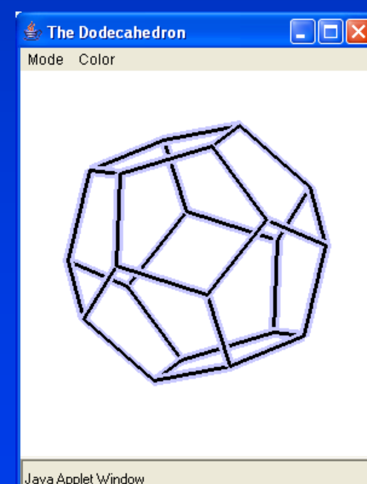
Platonic Solids



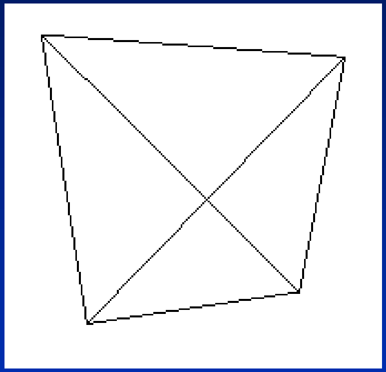




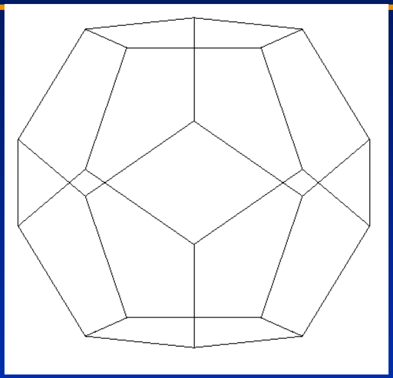




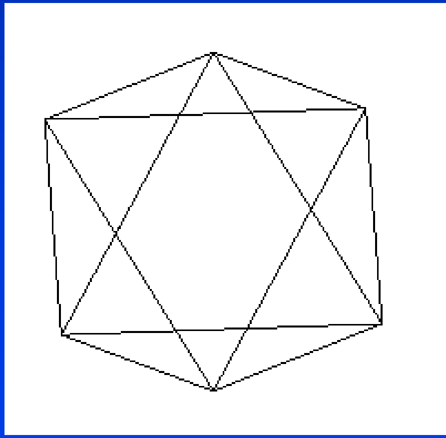
Platonic Solids



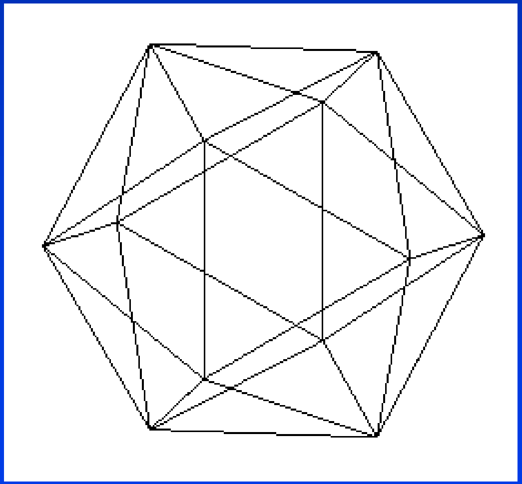
glutWireTetrahedron()



glutWireDodecahedron()



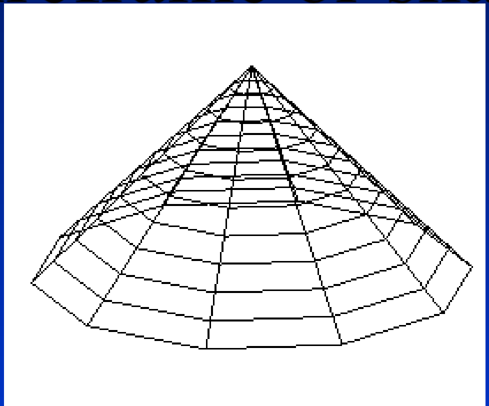
glutWireOctahedron()



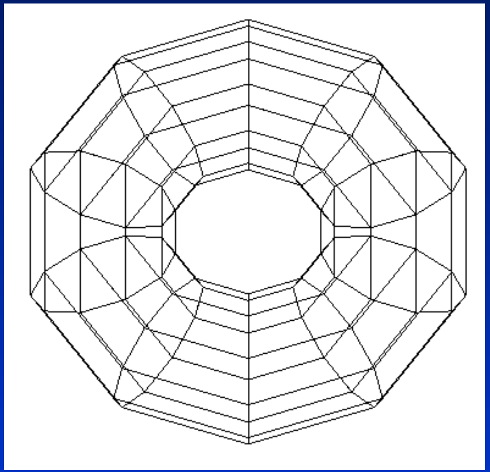
glutWireIcosahedron()

GLUT Objects

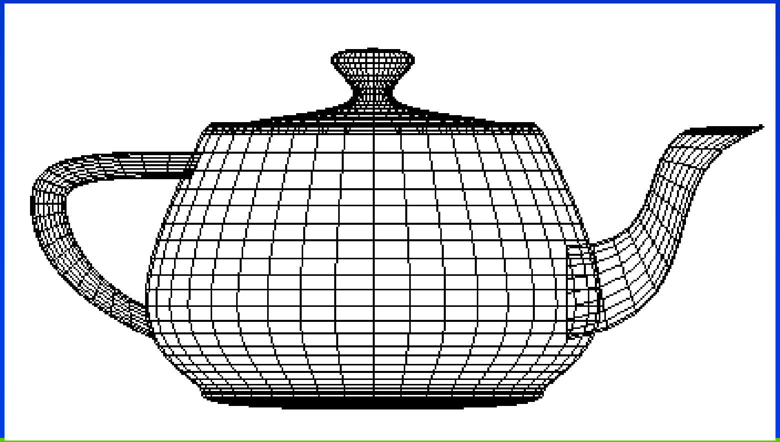
- Wireframe or shaded forms



`glutWireCone()`



`glutWireTorus()`



`glutWireTeapot()`

OpenGL Utility Toolkit (GLUT)

- GLUT is a library that handles system events and windowing across multiple platforms
- Includes some nice utilities
- We *strongly* suggest you use it

GLUT – Starting Point

```
int main (int argc, char *argv[])
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_DEPTH | GLUT_DOUBLE |
                        GLUT_RGBA);
   glutInitWindowSize (windowWidth, windowHeight);
   glutInitWindowPosition (0, 0);
   glutCreateWindow (“248 Video Game!");

   SetStates();          // Initialize rendering states*
   RegisterCallbacks();  // Set event callbacks*

   glutMainLoop();       // Start GLUT
   return 0;
}
```

* Your code here

Rendering States - Setup

- OpenGL is a *state* machine: polygons are affected by the current color, transformation, drawing mode, etc.
- Enable and disable features such as lighting, texturing, and alpha blending.
  - `glEnable (GL_LIGHTING);`
  - `glDisable (GL_FOG);`
- Forgetting to enable something is a common source of bugs!

GLUT Event Callbacks

- Register functions that are called when certain events happen

```
glutDisplayFunc( Display );
glutKeyboardFunc( Keyboard );
glutReshapeFunc( Reshape );
glutMouseFunc( Mouse );
glutPassiveMotionFunc( PassiveFunc );
glutMotionFunc( MouseDraggedFunc );
glutIdleFunc( Idle );
```

Lighting

- Lights have a position, type, color, among other things
- Types of lights include point light, directional light, and spotlight
  - glEnable (GL_LIGHTING)

Normals and Lighting

- OpenGL handles light computations for you!
- You will need to compute normal vector (kept as state) – vertex is assigned to the most recently set normal vector

```
...
glNormal3fv (n0);
glVertex3fv (v0);
glVertex3fv (v1);
glVertex3fv (v2);
...
```

- Note that, normal vectors are of unit length (remember normalization)!

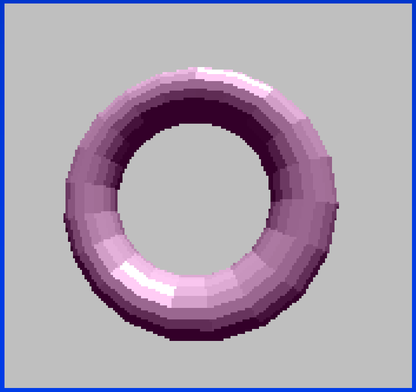
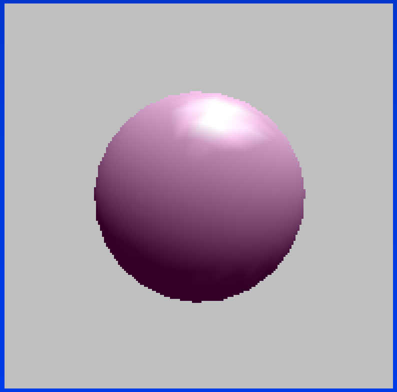
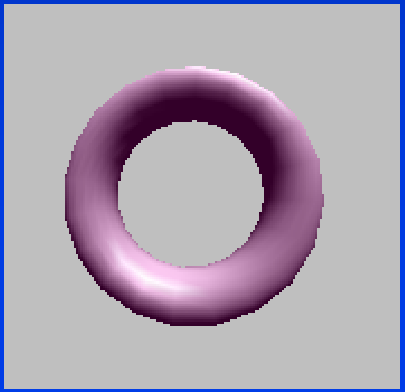
Color Specification

```
glColor3f(0.0, 0.0, 0.0);
draw_object(A);
draw_object(B);
glColor3f(1.0, 0.0, 0.0);
draw_object(C);
```

```
glColor3f(0.0, 0.0, 0.0);        black
glColor3f(1.0, 0.0, 0.0);        red
glColor3f(0.0, 1.0, 0.0);        green
glColor3f(1.0, 1.0, 0.0);        yellow
glColor3f(0.0, 0.0, 1.0);        blue
glColor3f(1.0, 0.0, 1.0);        magenta
glColor3f(0.0, 1.0, 1.0);        cyan
glColor3f(1.0, 1.0, 1.0);        white
```

Shading

- Two basic shading models supported by OpenGL (flat, smooth)
- glShadeModel (GL_FLAT); glShadeModel (GL_SMOOTH);

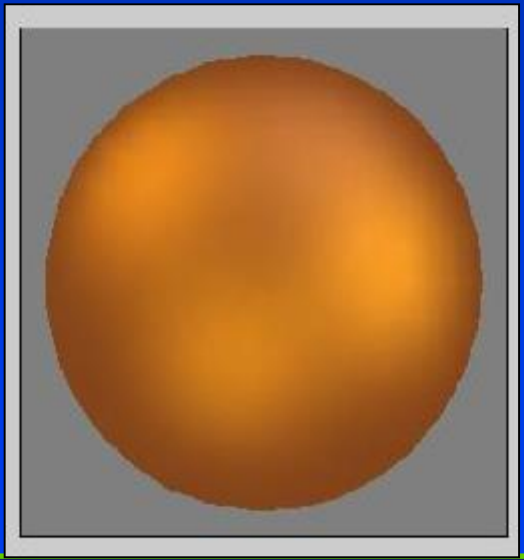
Material Properties

- Some properties (pname)
  - GL_AMBIENT: Ambient color of material
  - GL_DIFFUSE: Diffuse color of material
  - GL_SPECULAR: Specular component (for highlights)
  - GL_SHININESS: Specular exponent (intensity of highlight)
- Material properties are associated with each polygon (corresponding light properties)
  - **`glMaterial*(GLenum face, GLenum pname, TYPE param);`**

Material Selection

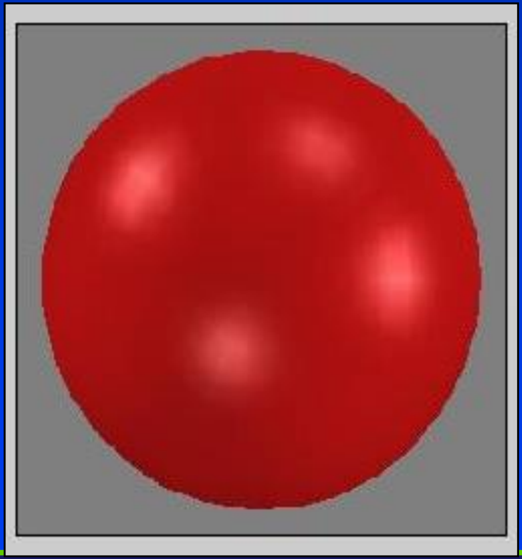
Ambient 0.52
Diffuse 0.00
Specular 0.82
Shininess 0.10

Light intensity 0.31



Ambient 0.39
Diffuse 0.46
Specular 0.82
Shininess 0.75

Light intensity 0.52



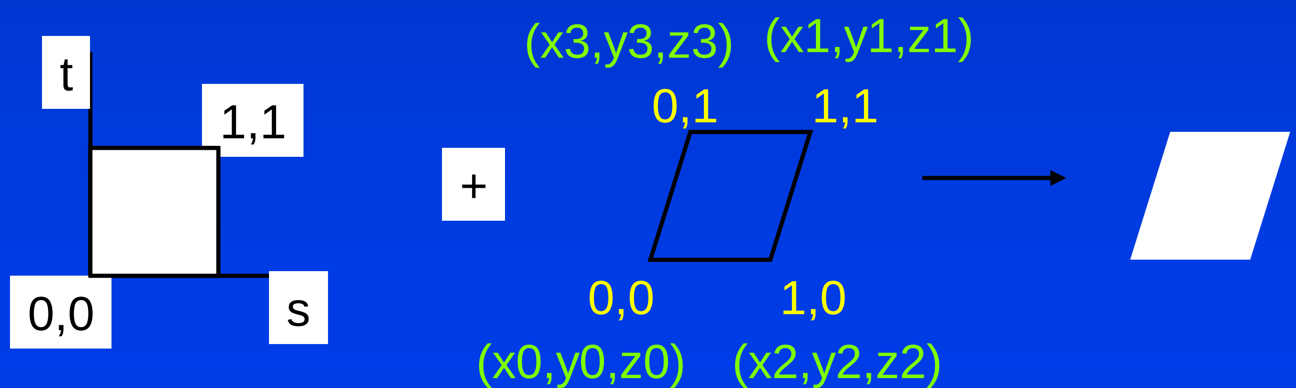
Texturing

Texturing

- Load your data (texture data)
  - This may come from an image: ppm, tiff
  - Or create at run time
  - Final result is always an array
- Setting texture state
  - Creating texture names with “binding”, scaling the image/data, building Mipmaps, setting filters, etc.

Texturing

- Mapping the texture to the polygon
  - specify (s,t) texture coordinates for (x,y,z) polygon vertices
  - texture coordinates (s,t)are from 0,1: **`glTexCoord2f(s,t);`**



Advanced Texturing

- Advanced texturing techniques
  - Mipmapping
  - Multitextures
  - Automatic texture generation
    - Let OpenGL determine texture coordinates for you
  - Environment Mapping
  - Texture matrix stack
  - Fragment Shaders
    - Custom lighting effects

Alpha Blending

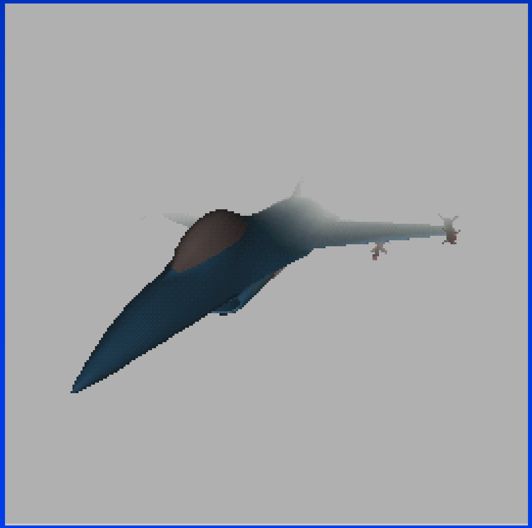
- When enabled, OpenGL uses the alpha channel to blend a new fragment’s color value with a color in the framebuffer

- Useful for overlaying textures or other effects

Fog

Simulate atmospheric effects

- glFog (): Sets fog parameters
- glEnable (GL_FOG);



Other Features

- Display Lists: Speed up your game!
- Quadrics: Pre-made objects
  - Also look at GLUT's objects
- Evaluators: Bezier curves and surfaces
- Selection: Clicking on game objects with a mouse

Buffers

- Multiple types of buffers
  - Color buffers (front/back, left/right)
  - Depth buffer (hidden surface removal)
  - Stencil buffer (allows masking or stenciling)
  - Accumulation buffer (antialiasing, depth of field)
- Clearing buffers:

```
// Clear to this color when screen is cleared.
glClearColor (0.0, 0.0, 0.0, 0.0);

// Clear color and depth buffers.
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Double Buffering

- Double buffering:
  - Draw on *back* buffer while *front* buffer is being displayed.
  - When finished drawing, swap the two, and begin work on the new back buffer.
  - **`glutSwapBuffers();`**
- Primary purpose: eliminate flicker