Spiele

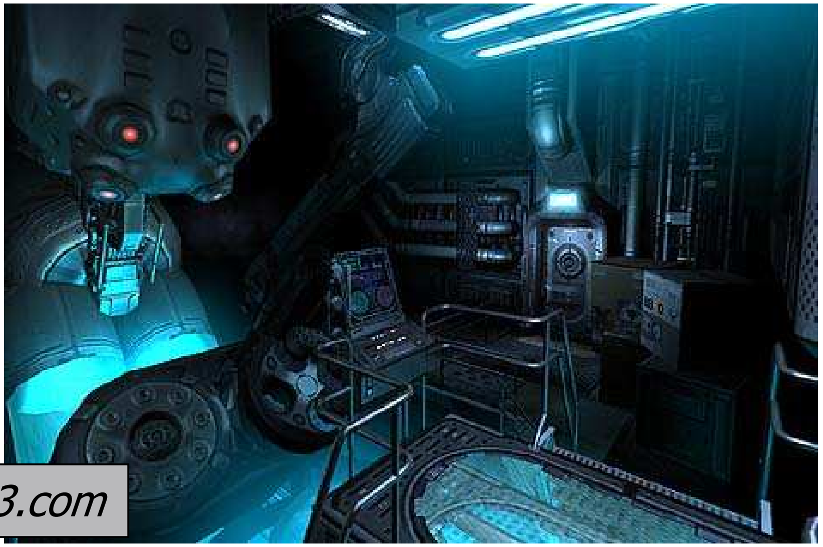
Daniel Scherzer

Hochschule
Ravensburg-Weingarten
Technik | Wirtschaft | Sozialwesen

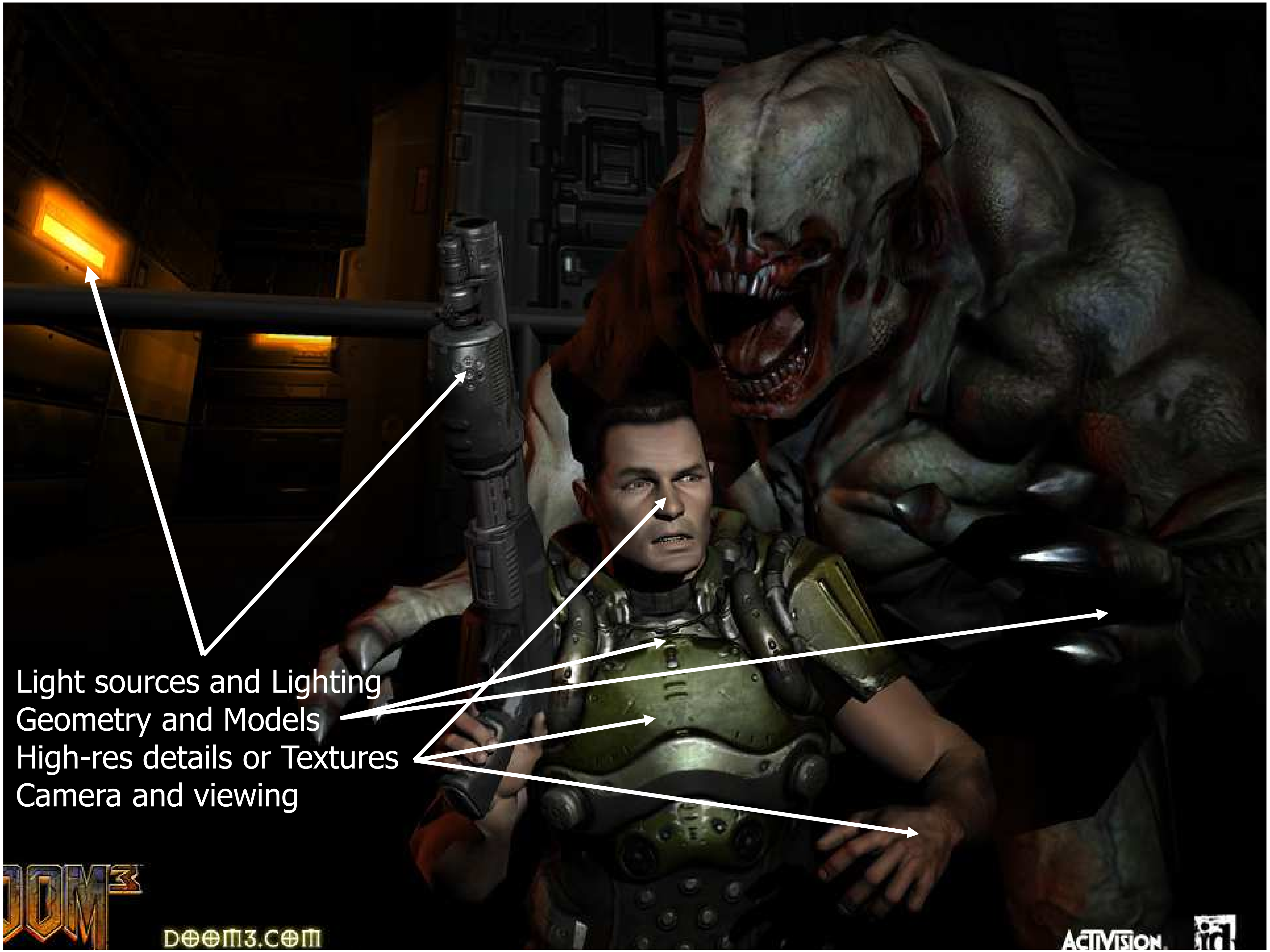
OpenGL

What is OpenGL?

- An application programming interface (API)
- A (low-level) Graphics rendering API
- A pipe-line to generate high-quality images composed of geometric and image primitives
- A ***state*** machine



www.doom3.com


Light sources and Lighting
Geometry and Models
High-res details or Textures
Camera and viewing
DOOM3
DOOM3.COM
ACTIVISION
id

A 3D graphics API

- Separates code
  - Opengl32.dll on Windows
  - Vendors package own version of library with graphics card
  - Software-only version of OpenGL(e.x. Mesa) exist

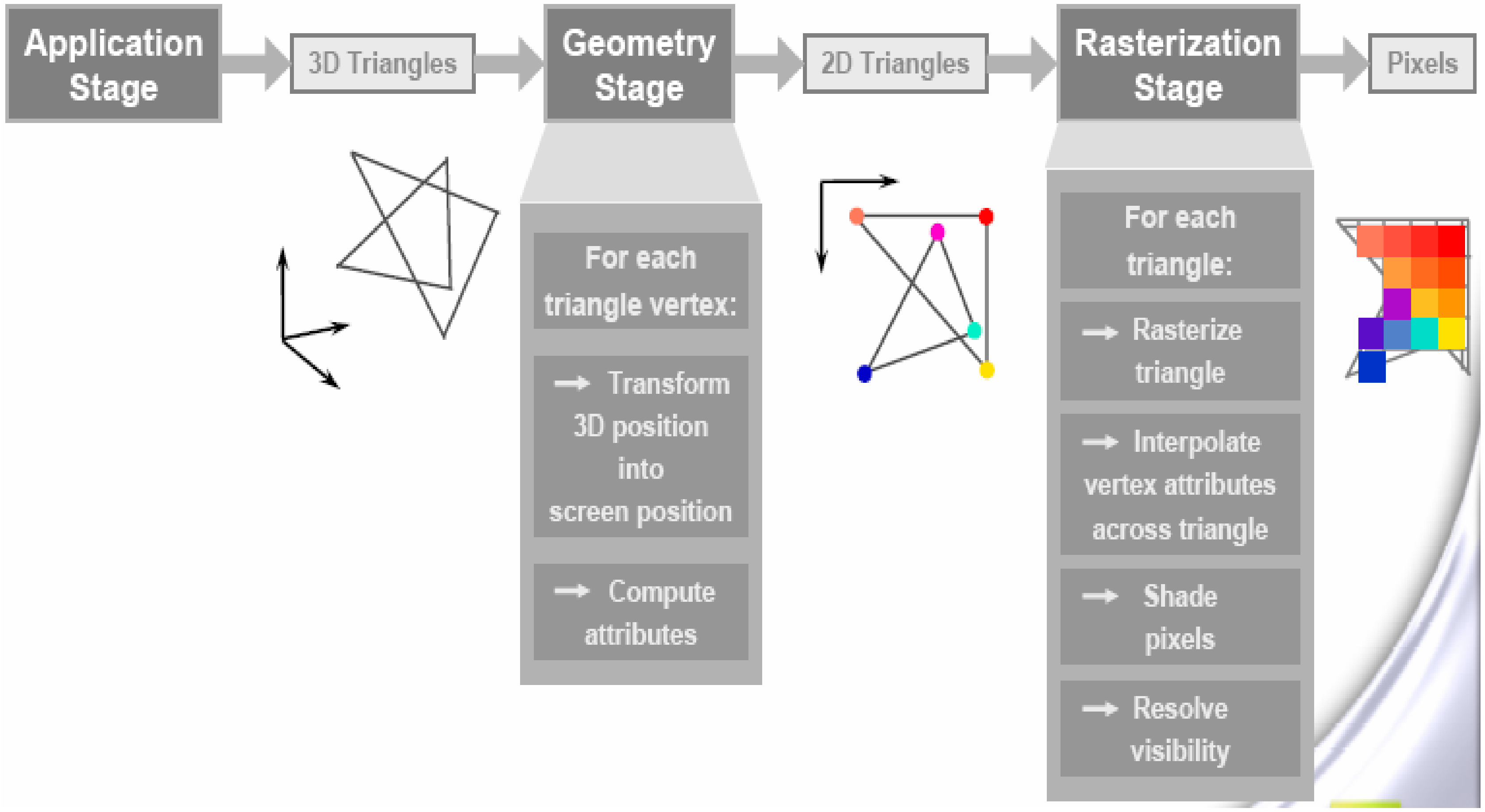
A low-level 3D graphics API

- An interface to hardware
  - Knows how to interface with the card drivers
  - Software fallback for unsupported features

A low-level 3D graphics API

- Primitive-based
  - Objects consist of points, line-segments, and polygons
  - OpenGL is not aware of any connections between primitives

A Pipeline Architecture



A State Machine

- Functions are global and change the state of the OpenGL environment
- State can be pushed onto stacks and popped back off
- OpenGL properties remain as you set them until you set them again

OpenGL Is Not

- A modeling language
- Compiled directly into your code
- Object-oriented

Introducing OpenGL

- Implements the rendering pipeline:
  - Transform geometry (object →world, world → eye)
  - Calculate surface lighting
  - Apply perspective projection (eye → screen)
  - Clip to the view frustum
  - Perform visible-surface processing
- Implementing all this is a lot of work

provides a standard implementation

OpenGL Design Goals

- SGI’s design goals for OpenGL:
  - High-performance (hardware-accelerated) graphics API
  - Hardware independence
  - Built-in extensibility
- OpenGL has become a standard because:
  - It doesn’t try to do too much
    - Only renders the image, doesn’t manage windows, etc.
    - No high-level animation, modeling, sound (!), etc.
  - It does enough
    - Useful rendering effects + high performance
  - It was promoted by SGI (& Microsoft, half-heartedly), is now promoted/supported by NVIDIA, ATI, Apple …

OpenGL Type	Definition in C
GLbyte	signed char
GLshort	short
GLint	int
GLfloat	float
GLdouble	double
GLushort	unsigned short

Example

```
  GLshort shorts[10];
  GLdouble *doubles[10];
```

Command Syntax

`glVertex3fv( … )`

Number of components

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

Data Type

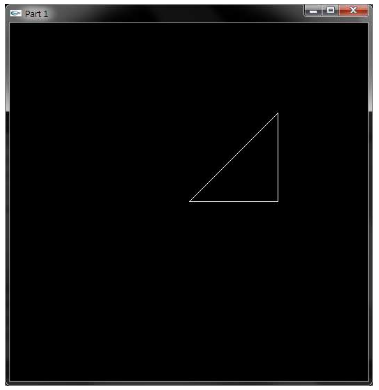
```
b  - byte
ub - unsigned byte
s  - short
us - unsigned short
i  - int
ui - unsigned int
f  - float
d  - double
```

Vector

```
omit “v” for
scalar form

glVertex2f( x, y )
```

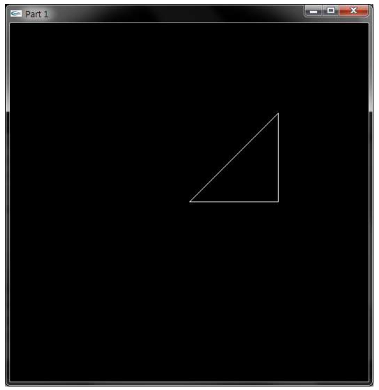
Argument Type

```
#include <GL/glut.h>
void display(void){
   glClearColor(0, 0, 0, 0);
   glClear(GL_COLOR_BUFFER_BIT);
   glDisable(GL_CULL_FACE);
   glPolygonMode(GL_FRONT,GL_LINE);
   glColor3f(1, 1, 1);
   glBegin(GL_TRIANGLES);
        glVertex2f(0, 0);
        glVertex2f(.5, 0);
        glVertex2f(.5, .5);
   glEnd();
  glutSwapBuffers();
}
```



Render polygons of model

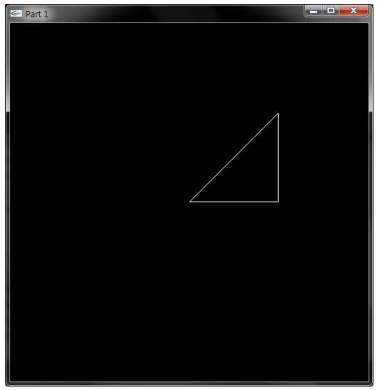
```
#include <GL/glut.h>
void display(void){
   glClearColor(0, 0, 0, 0);
   glClear(GL_COLOR_BUFFER_BIT);
   glDisable(GL_CULL_FACE);
   glPolygonMode(GL_FRONT,GL_LIN
   E);
   glColor3f(1, 1, 1);
   glBegin(GL_TRIANGLES);
        glVertex2f(0, 0);
        glVertex2f(.5, 0);
        glVertex2f(.5, .5);
   glEnd();
  glutSwapBuffers();
}
```



Specifying Geometry

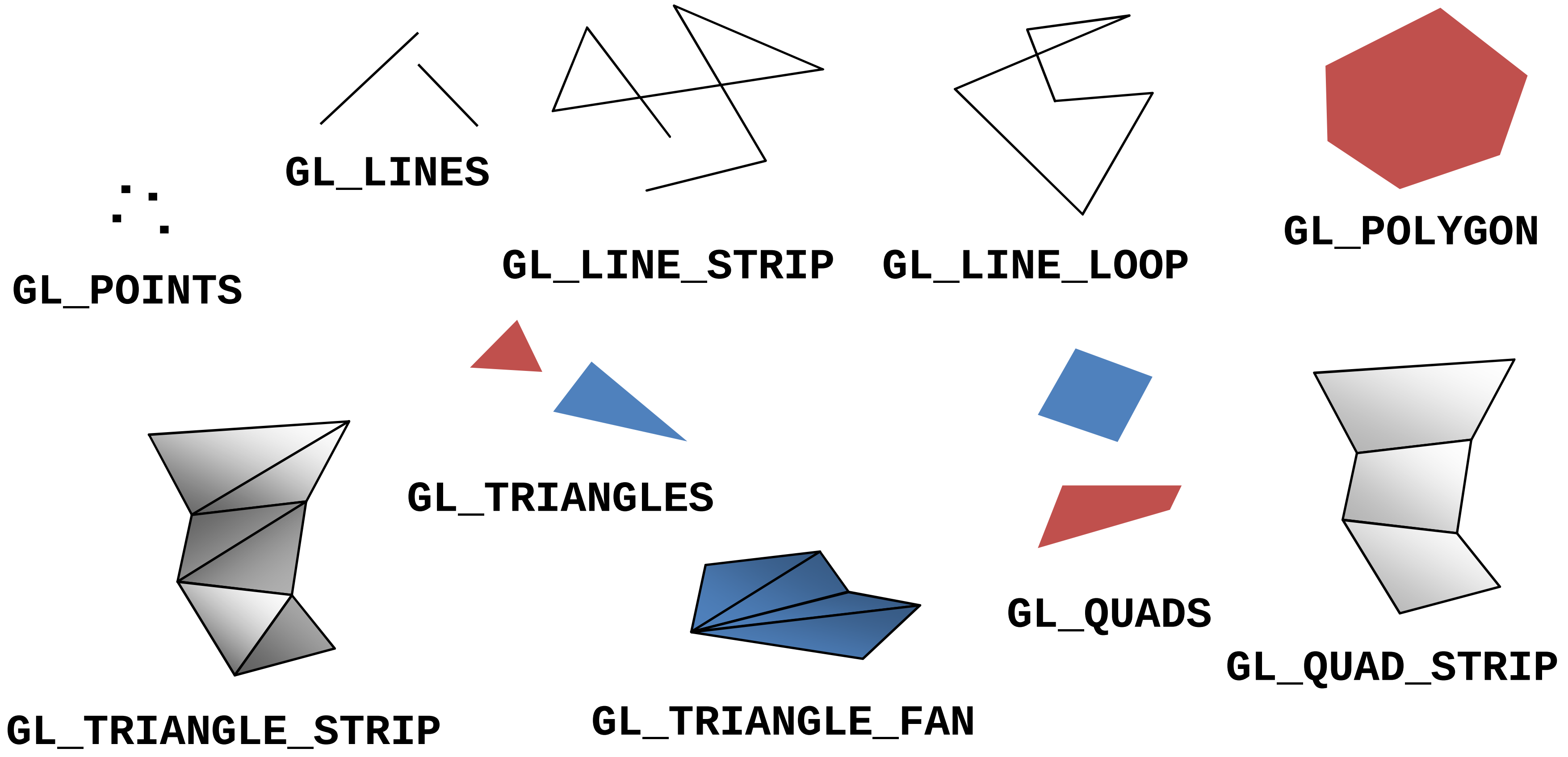
```
#include <GL/glut.h>
void display(void){
   glClearColor(0, 0, 0, 0);
   glClear(GL_COLOR_BUFFER_BIT);
   glDisable(GL_CULL_FACE);
   glPolygonMode(GL_FRONT,GL_LINE);
   glColor3f(1, 1, 1);
   glBegin(GL_TRIANGLES);
       glVertex2f(0, 0);
       glVertex2f(.5, 0);
       glVertex2f(.5, .5);
   glEnd();
  glutSwapBuffers();
}
```

- List of vertices between **`glBegin()`** and **`glEnd()`**
- Usage: **`glBegin(`*geomtype*`)`** where geomtype is:
  - Points, lines, polygons, triangles, quadrilaterals, etc…



Geometric Primitives

- All geometric primitives are specified by vertices



Specifying Geometry

```
glBegin(GL_TRIANGLES);
  glVertex2f(0, 0);
  glVertex2f(.5, 0);
  glVertex2f(.5, .5);
glEnd();
```

- 1 triangle

```
glBegin(GL_POINTS);
  glVertex2f(0, 0);
  glVertex2f(.5, 0);
glEnd();
```

- 2 points

```
glBegin(GL_LINES);
  glVertex2f(0, 0);
  glVertex2f(.5, 0);
glEnd();
```

- 1 line

Specifying Geometry

```
glBegin(GL_QUADS);
   glVertex2f(0, 0);
   glVertex2f(.5, 0);
   glVertex2f(.5, .5);
   glVertex2f(0, .5);
glEnd();
```

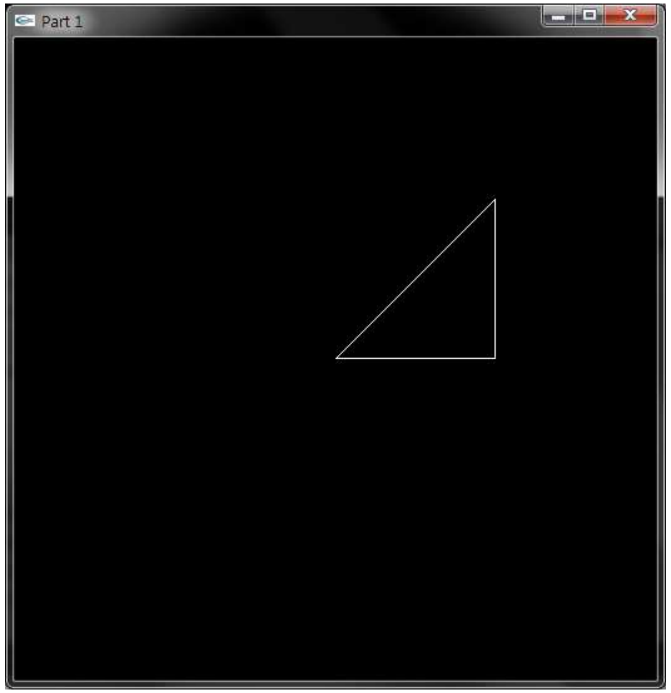
- 1 quadrilateral

```
glBegin(GL_POLYGON);
   glVertex2f(0, 0);
   glVertex2f(.5, 0);
   glVertex2f(.5, .5);
   glVertex2f(.25, .75);
   glVertex2f(0, .5);
glEnd();
```

- 1 polygon

Specifying Triangle - Variations

```
static const float tri[][2] = {{0, 0}, {0.5, 0}, {0.5, 0.5}};


glBegin(GL_TRIANGLES);
   glVertex2f(tri[0][0], tri[0][1]);
   glVertex2f(tri[1][0], tri[1][1]);
   glVertex2f(tri[2][0], tri[2][1]);
glEnd();

glBegin(GL_TRIANGLES);
   glVertex3fv(tri[0]);
   glVertex3fv(tri[1]);
   glVertex3fv(tri[2]);
glEnd();

glBegin(GL_TRIANGLES);
  for (unsigned i = 0; i < 3; ++i) glVertex3fv(tri[i]);
glEnd();
```



Specifying Geometry

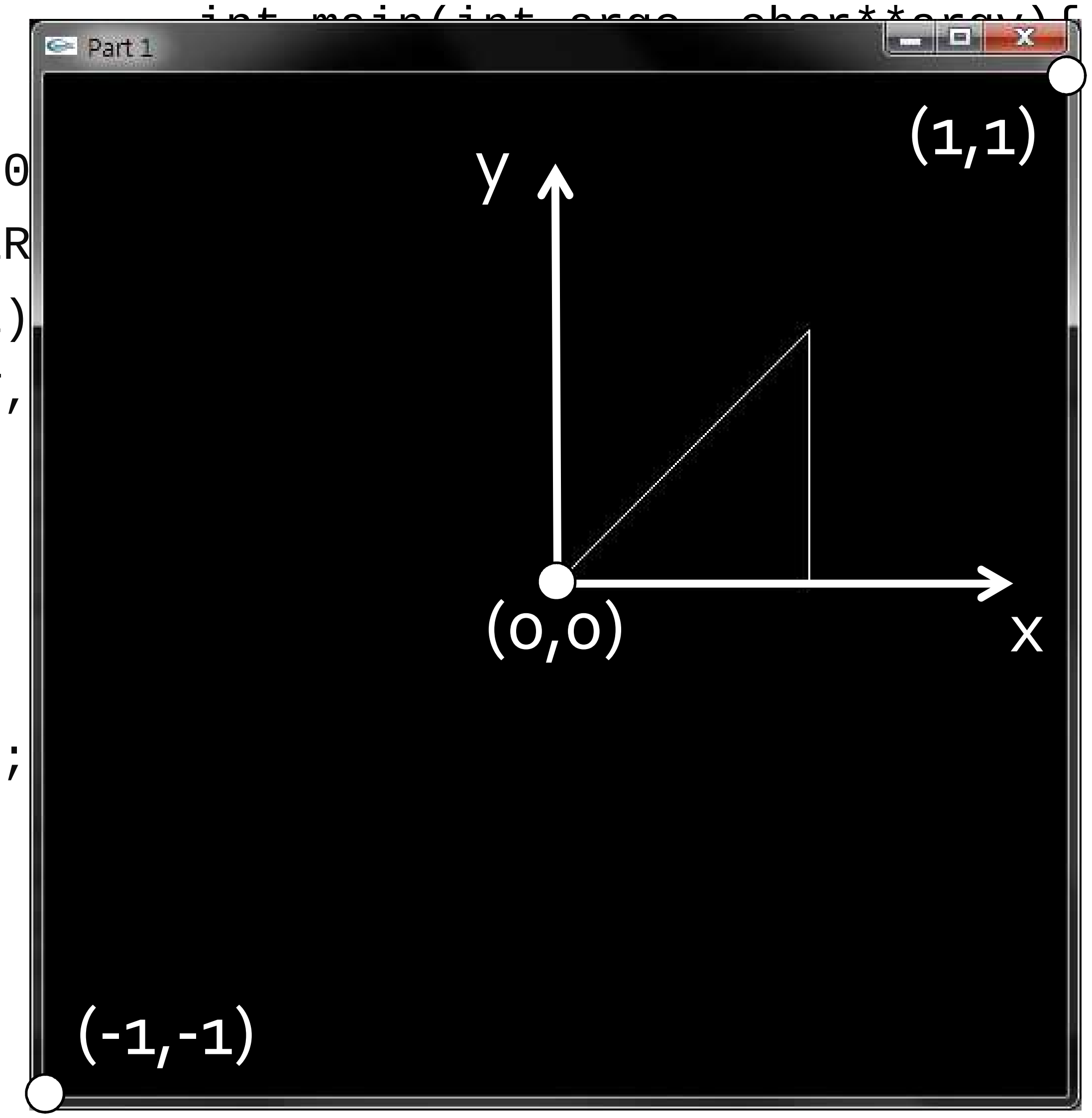
```
#include <GL/glut.h>
void display(void){
   glClearColor(0, 0, 0, 0);
   glClear(GL_COLOR_BUFFER_BIT);
   glDisable(GL_CULL_FACE);
   glPolygonMode(GL_FRONT,GL_LINE);
   glColor3f(1, 1, 1);
   glBegin(GL_TRIANGLES);
        glVertex2f(0, 0);
        glVertex2f(.5, 0);
        glVertex2f(.5, .5);
   glEnd();
  glutSwapBuffers();
}
```

- A **`glVertex`** command sends a vertex to the graphics hardware
- Parameters are the coordinates

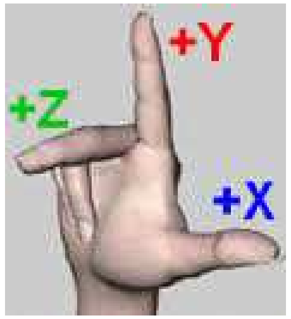
Default Coordinate System

```
#include <GL/glut.h>
void display(void){
   glClearColor(0, 0, 0, 0
   glClear(GL_COLOR_BUFFER
   glDisable(GL_CULL_FACE)
   glPolygonMode(GL_FRONT,
   glColor3f(1, 1, 1);
   glBegin(GL_TRIANGLES);
       glVertex2f(0, 0);
       glVertex2f(.5, 0);
       glVertex2f(.5, .5);
   glEnd();
  glutSwapBuffers();
}
```

```
int main(int argc, char**argv){
```

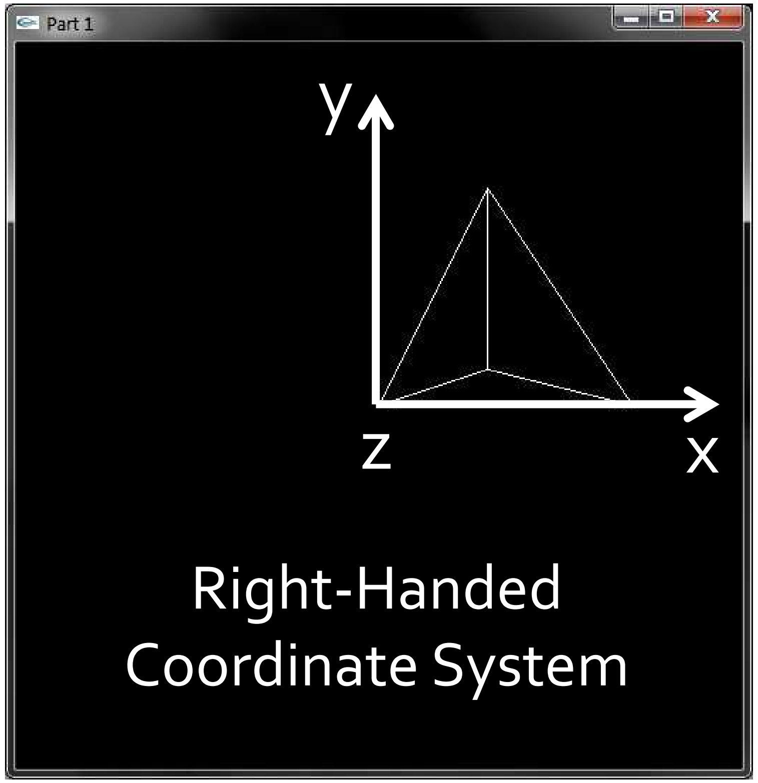


3D Geometry



```
static const float tetrah[][3] = {
   {0, 0, 0}, {0.7, 0, 0}, {0.3, 0.1, 0.7}, {0.3, 0.6, 0}};
glBegin(GL_TRIANGLES);
   glVertex3fv(tetrah[0]);
   glVertex3fv(tetrah[1]);
   glVertex3fv(tetrah[2]);

   glVertex3fv(tetrah[0]);
   glVertex3fv(tetrah[1]);
   glVertex3fv(tetrah[3]);

   glVertex3fv(tetrah[0]);
   glVertex3fv(tetrah[2]);
   glVertex3fv(tetrah[3]);

   glVertex3fv(tetrah[1]);
   glVertex3fv(tetrah[2]);
   glVertex3fv(tetrah[3]);
glEnd();
```

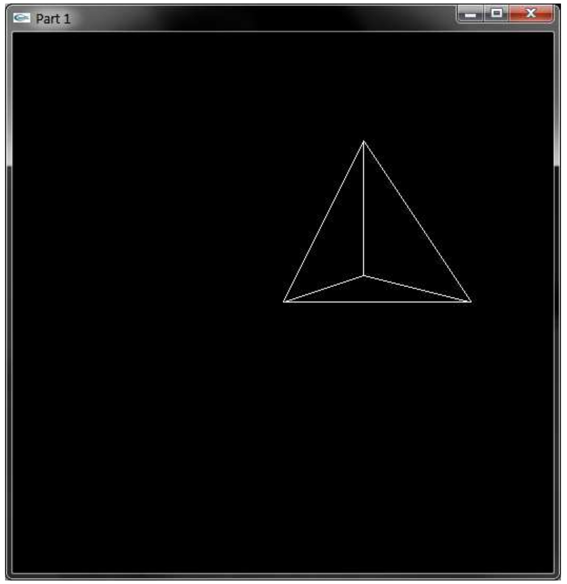
4 triangles



Right-Handed Coordinate System

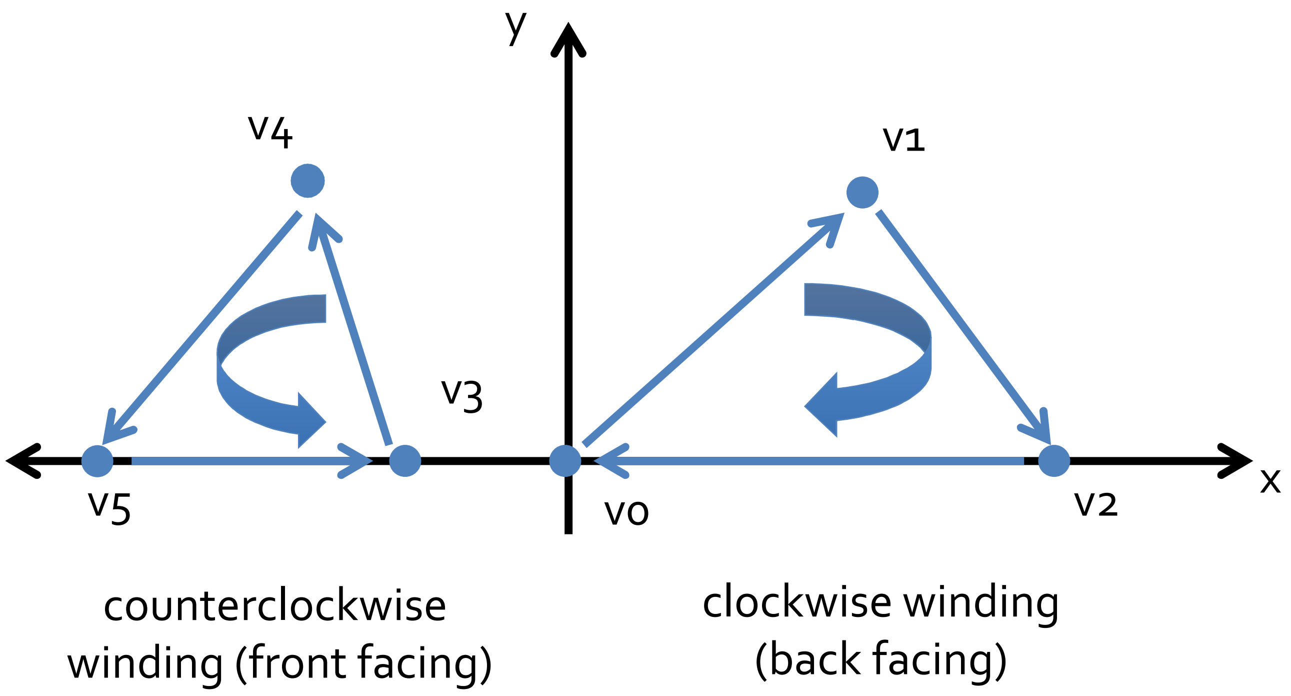
3D Geometry

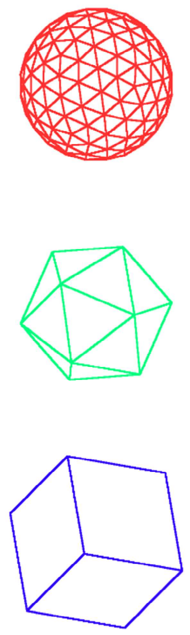
```
void drawTriangle(float *v1, float *v2, float *v3)
{
   glBegin(GL_TRIANGLES);
     glVertex3fv(v1);
     glVertex3fv(v2);
     glVertex3fv(v3);
   glEnd();
}


void drawTetraheder()
{
  static const float tetrah[][3] = {
    {0, 0, 0}, {0.7, 0, 0}, {0.3, 0.1, 0.7}, {0.3, 0.6, 0}};
  drawTriangle(tetrah[0], tetrah[1], tetrah[2]);
  drawTriangle(tetrah[0], tetrah[1], tetrah[3]);
  drawTriangle(tetrah[0], tetrah[2], tetrah[3]);
  drawTriangle(tetrah[1], tetrah[2], tetrah[3]);
}
```



Front/Back Face Rendering

- Each polygon has two sides, front and back
- OpenGL can cull and render them differently
- Ordering of vertices determines side


y
v4
v1
v3
v5
v0
v2
x
counterclockwise winding (front facing)
clockwise winding (back facing)

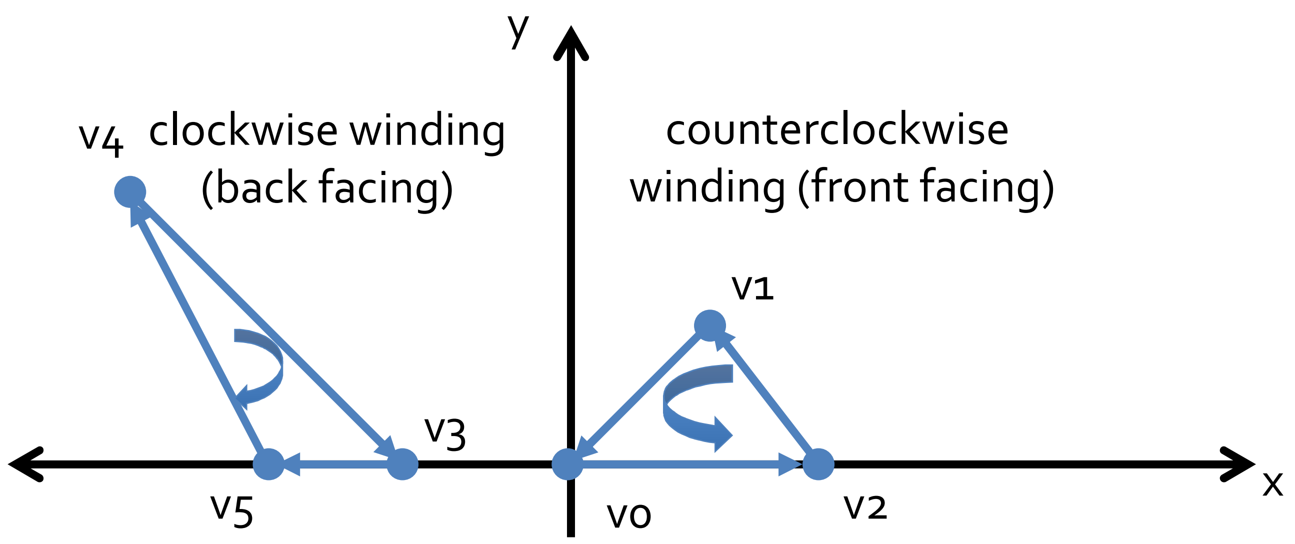


Example

```
glBegin(GL_TRIANGLES);
    glVertex2f(0.0f, 0.0f);       // v0
    glVertex2f(50.0f, 0.0f);      // v2
    glVertex2f(25.0f, 25.0f);     // v1

    glVertex2f(-25.0f, 0.0f);     // v3
    glVertex2f(-50.0f, 0.0f);     // v5
    glVertex2f(-75.0f, 50.0f);    // v4
glEnd();
```

counterclockwise winding (front facing)

clockwise winding (back facing)



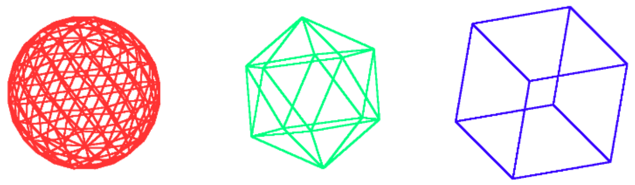
Back Face Culling

```
#include <GL/glut.h>
void display(void){
   glClearColor(0, 0, 0, 0);
   glClear(GL_COLOR_BUFFER_BIT);
   glDisable(GL_CULL_FACE);
   glPolygonMode(GL_FRONT,GL_LINE);
   glColor3f(1, 1, 1);
   glBegin(GL_TRIANGLES);
        glVertex2f(0, 0);
        glVertex2f(.5, 0);
        glVertex2f(.5, .5);
   glEnd();
  glutSwapBuffers();
}
```

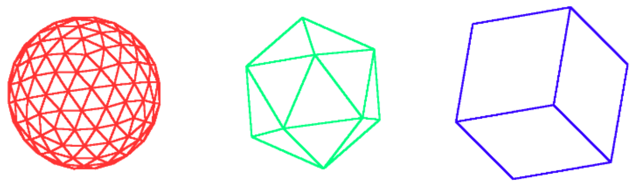
- Cull face means remove back facing polygons
- Here back faces are rendered
- To cull back faces use

  `glEnable(GL_CULL_FACE)`

Back Face Culling

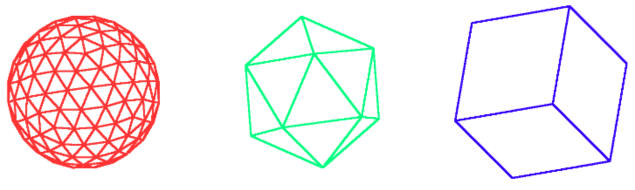


`glDisable(GL_CULL_FACE)`

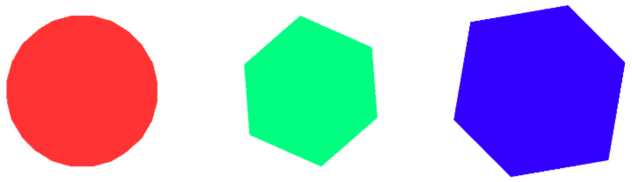


`glEnable(GL_CULL_FACE)`

Wireframe vs Filled Rendering



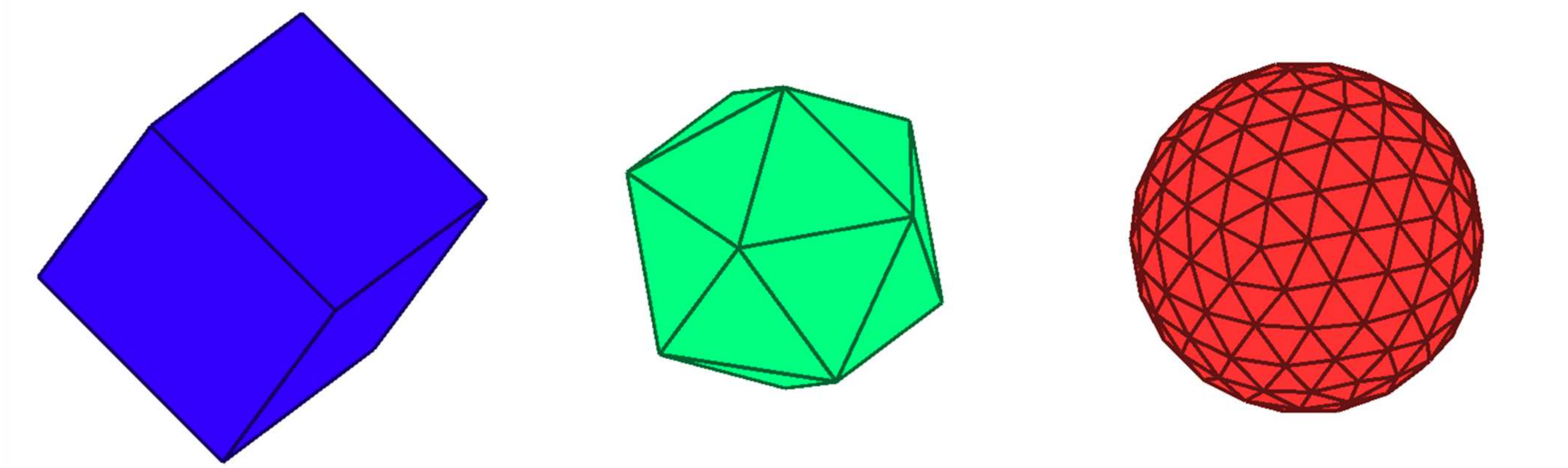
`glPolygonMode(GL_FRONT,GL_LINE)`



`glPolygonMode(GL_FRONT,GL_FILL)`

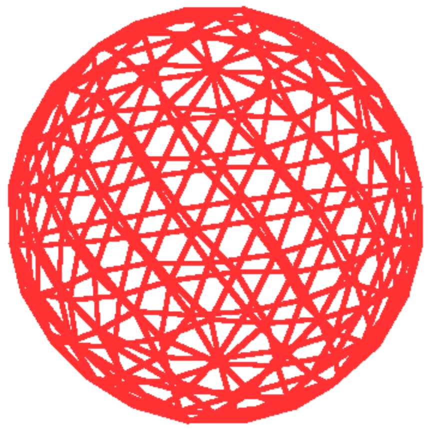
Outlines

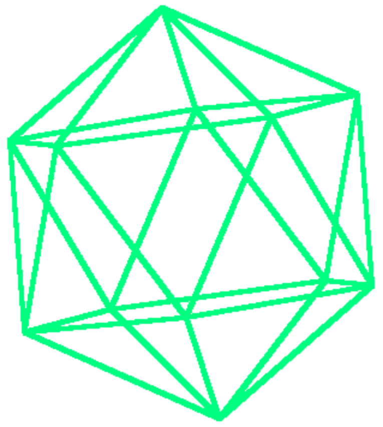
```
glColor3f(red, green, blue);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glDrawObject();
//draw object outlines on top of object
glColor3f(0.4 * red, 0.4 * green, 0.4 * blue);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glLineWidth(3.0);
//make wireframe model bigger so you see lines
glScalef(1.01, 1.01, 1.01);
glDrawObject();
```

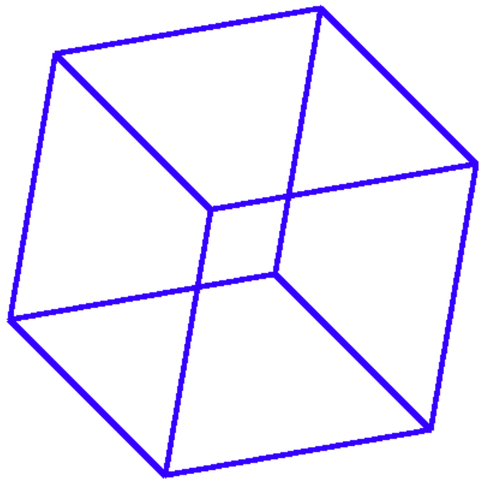


Todo: Draw Simple Geometry

- Cubes
- Icosahedron
- Sphere

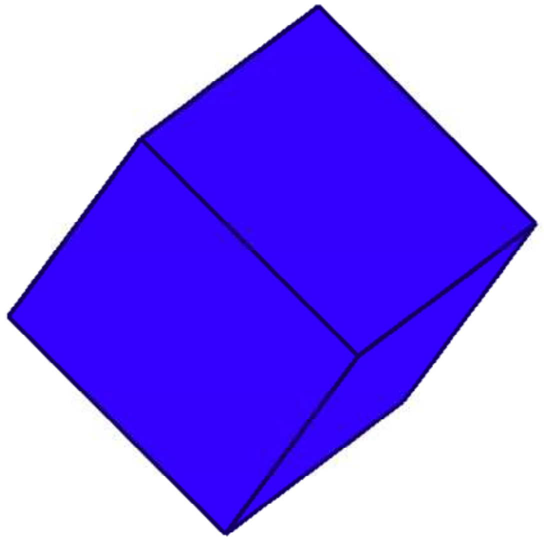
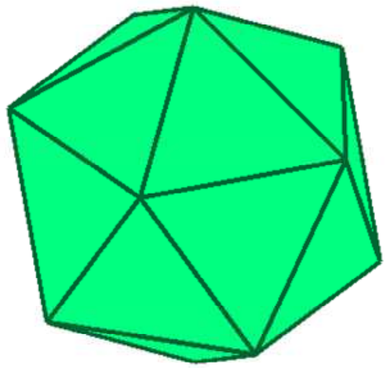
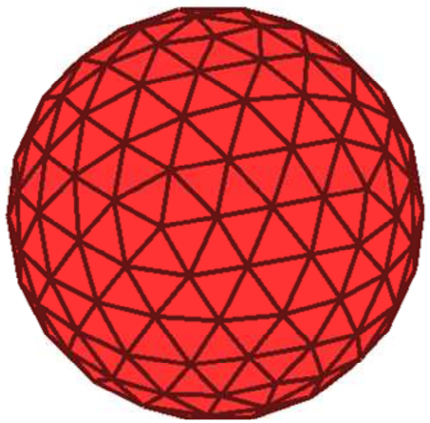






Todo: Draw Simple Geometry

- Cube
- Icosahedron
- Sphere

Todo: Some Rekursive Geometry

- E.x.: Sierpinski Tetrahedra