CSE 167 Discussion #3

Buffering...

Object Centering

- One shouldn’t expect the vertices in an OBJ file to be centered around the origin
  - The bear is a great example
- How can we center our object?
- Once we find a way to translate our model to the origin, do we make that our toWorld matrix?
  - Why not?

Object Centering

- Once we find a way to translate our model to the origin, do we make that our toWorld matrix?
  - We actually want to change object space itself, not make our toWorld the translation matrix that translates all vertices to center around the origin
  - What would happen if we do make our toWorld matrix the translation matrix?
    - What would then happen if we multiply in a scaling matrix to the right of our toWorld matrix?
      - Scaling would then precede the translation and scale the vertices when they are off-center (and thus not around its own center)

Object Centering

- How can we center our models?
  - First parse and populate data structures to hold vertices
  - Find the minX, minY, minZ, maxX, maxY, maxZ
    - Find the average of the two vectors <minX, minY, minZ> and <maxX, maxY, maxZ> to get the center of the object
  - Loop again through all vertices and subtract the vector representing the center of the object
    - This acts as a translation of each vertex individually which moves our entire model and makes the object's new center the origin

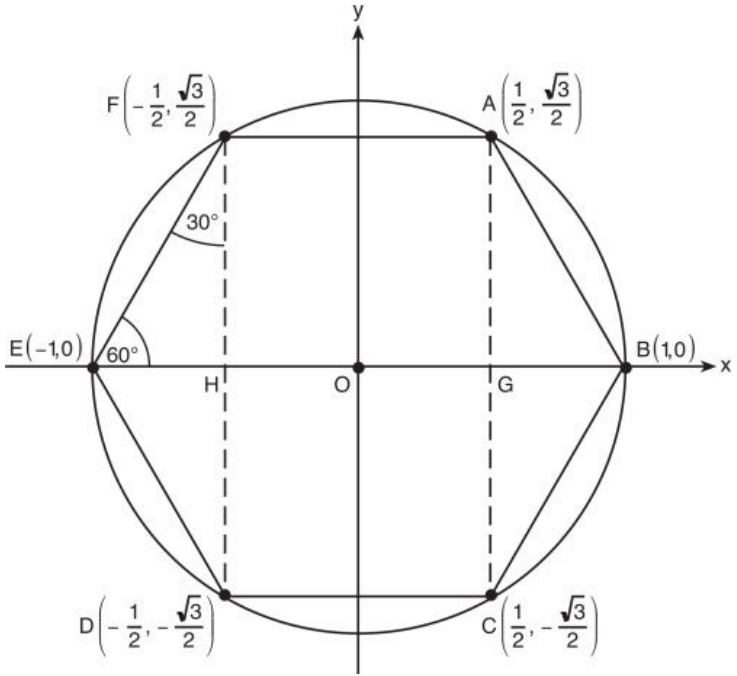
Object Scaling

- For this project we want a given model to fit in a 2x2x2 cube such that all vertex position values are in the range [-1, 1]
  - Loop through vertices
  - Determine the longest dimension of the model
    - The bear’s longest dimension, for example, is the y-dimension since it is taller than it is wide or deep
  - Scale (divide) all vertices using the largest dimension
    - This acts as a uniform scale to squeeze our model to a standardized size

OBJs

- So far we've only parsed
  - Vertices
  - Normals
- What about faces?
  - Why do we need them?
  - Why not just list the vertex?

OBJs



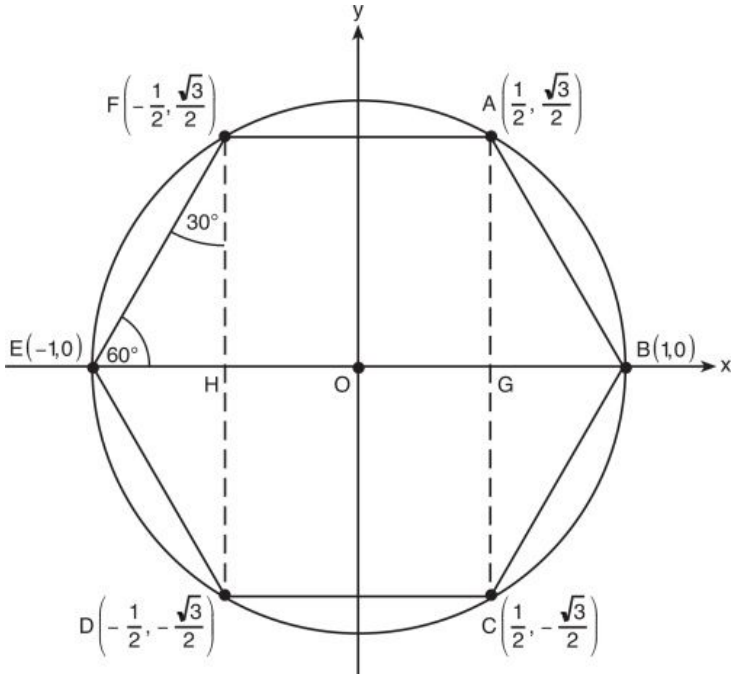
Using faces

```
v   0.0    0.0   0.0
v   1.0    0.0   0.0
v   0.5   0.86   0.0
f  1//  2//  3//
```

Without using faces

```
v   0.0    0.0   0.0
v   1.0    0.0   0.0
v   0.5   0.86   0.0
```

OBJs



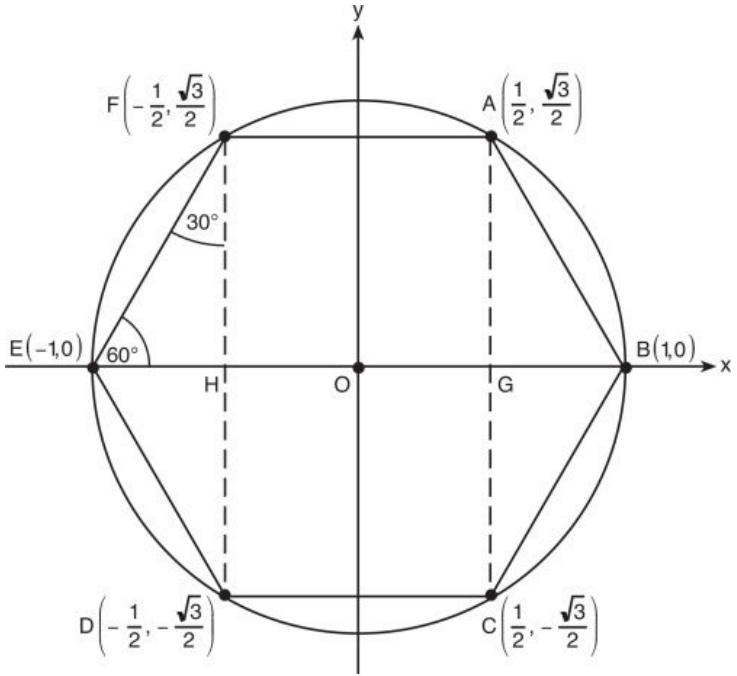
Using faces

```
v   0.0    0.0   0.0
v   1.0    0.0   0.0
v   0.5    0.86  0.0
v  -0.5    0.86  0.0
f  1//  2//  3//
f  1//  3//  4//
```

Without using faces

```
v   0.0    0.0   0.0
v   1.0    0.0   0.0
v   0.5    0.86  0.0
v   0.0    0.0   0.0
v   0.5    0.86  0.0
v  -0.5    0.86  0.0
```

OBJs



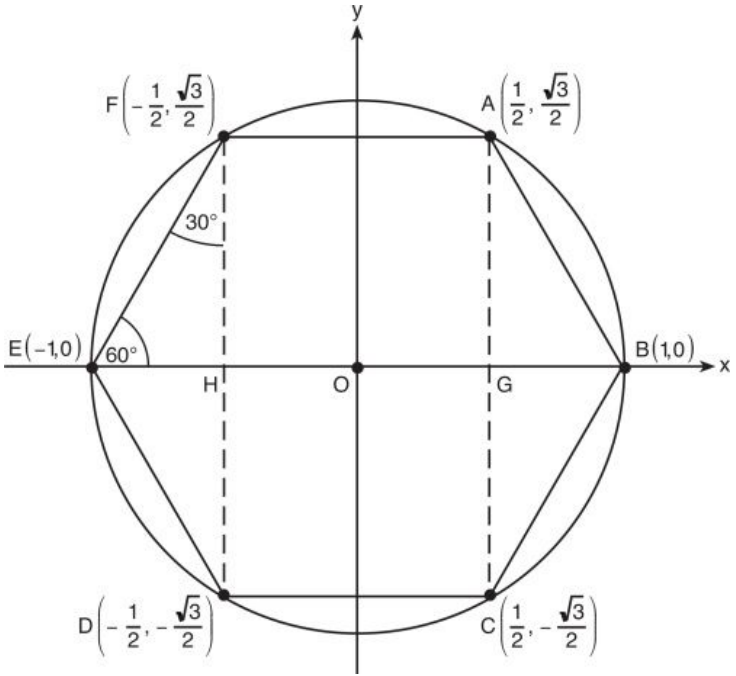
Using faces

```
v   0.0    0.0   0.0
v   1.0    0.0   0.0
v   0.5    0.86  0.0
v  -0.5    0.86  0.0
v  -1.0    0.0   0.0
f  1//  2//  3//
f  1//  3//  4//
f  1//  4//  5//
```

Without using faces

```
v   0.0    0.0   0.0
v   1.0    0.0   0.0
v   0.5    0.86  0.0
v   0.0    0.0   0.0
v   0.5    0.86  0.0
v  -0.5    0.86  0.0
v   0.0    0.0   0.0
v  -0.5    0.86  0.0
v  -1.0    0.0   0.0
```

OBJs



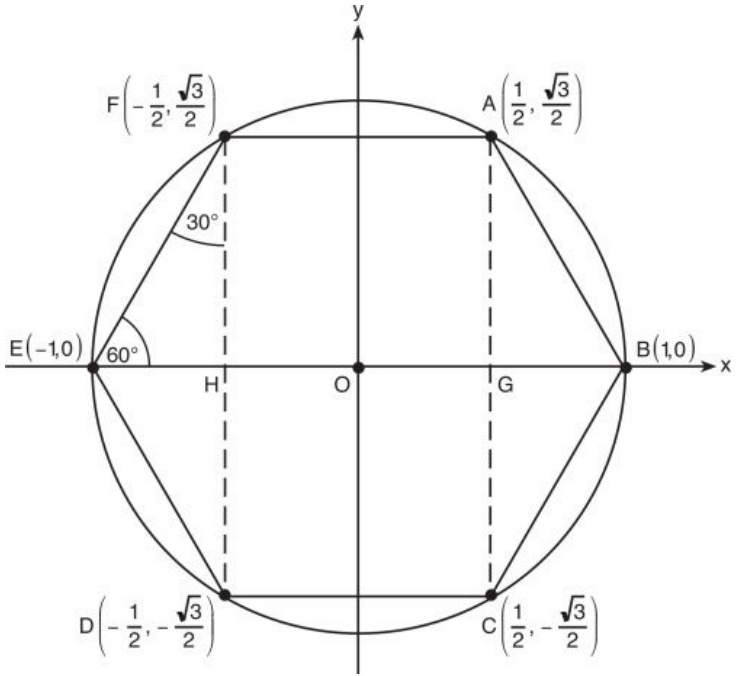
Using faces

```
v   0.0    0.0   0.0
v   1.0    0.0   0.0
v   0.5    0.86  0.0
v  -0.5    0.86  0.0
v  -1.0    0.0   0.0
v  -0.5   -0.86  0.0
f  1//  2//  3//
f  1//  3//  4//
f  1//  4//  5//
f  1//  5//  6//
```

Without using faces

```
v   0.0    0.0   0.0
v   1.0    0.0   0.0
v   0.5    0.86  0.0
v   0.0    0.0   0.0
v   0.5    0.86  0.0
v  -0.5    0.86  0.0
v   0.0    0.0   0.0
v  -0.5    0.86  0.0
v  -1.0    0.0   0.0
v   0.0    0.0   0.0
v  -1.0    0.0   0.0
v  -0.5   -0.86  0.0
```

OBJs



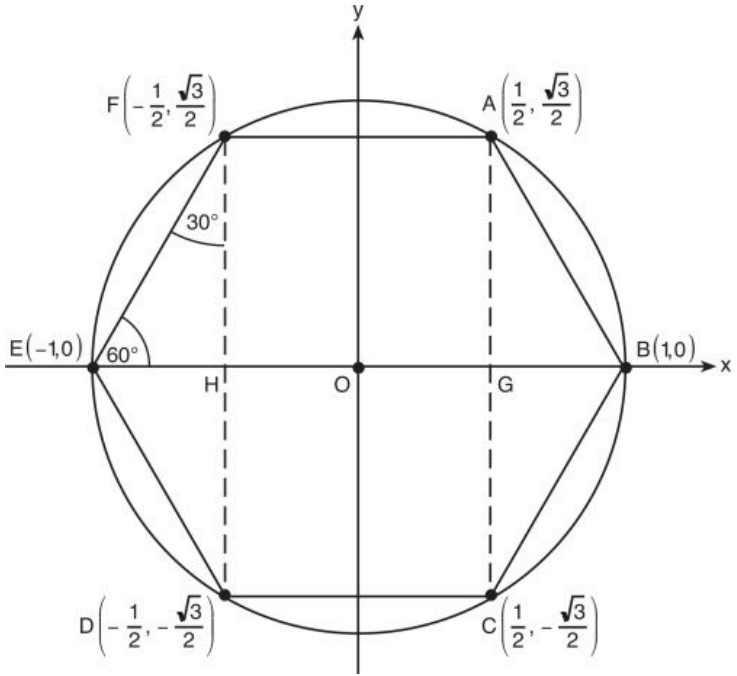
Using faces

```
v   0.0    0.0   0.0
v   1.0    0.0   0.0
v   0.5    0.86  0.0
v  -0.5    0.86  0.0
v  -1.0    0.0   0.0
v  -0.5   -0.86  0.0
v   0.5   -0.86  0.0
f  1//  2//  3//
f  1//  3//  4//
f  1//  4//  5//
f  1//  5//  6//
f  1//  6//  7//
```

Without using faces

```
v   0.0    0.0   0.0
v   1.0    0.0   0.0
v   0.5    0.86  0.0
v   0.0    0.0   0.0
v   0.5    0.86  0.0
v  -0.5    0.86  0.0
v   0.0    0.0   0.0
v  -0.5    0.86  0.0
v  -1.0    0.0   0.0
v   0.0    0.0   0.0
v  -1.0    0.0   0.0
v  -0.5   -0.86  0.0
v   0.0    0.0   0.0
v  -0.5   -0.86  0.0
v   0.5   -0.86  0.0
```

OBJs



Using faces

```
v   0.0    0.0   0.0
v   1.0    0.0   0.0
v   0.5   0.86   0.0
v  -0.5   0.86   0.0
v  -1.0    0.0   0.0
v  -0.5  -0.86   0.0
v   0.5  -0.86   0.0
f  1//  2//  3//
f  1//  3//  4//
f  1//  4//  5//
f  1//  5//  6//
f  1//  6//  7//
f  1//  7//  2//
```

Without using faces

```
v   0.0    0.0   0.0
v   1.0    0.0   0.0
v   0.5   0.86   0.0
v   0.0    0.0   0.0
v   0.5   0.86   0.0
v  -0.5   0.86   0.0
v   0.0    0.0   0.0
v  -0.5   0.86   0.0
v  -1.0    0.0   0.0
v   0.0    0.0   0.0
v  -1.0    0.0   0.0
v  -0.5  -0.86   0.0
v   0.0    0.0   0.0
v  -0.5  -0.86   0.0
v   0.5  -0.86   0.0
v   0.0    0.0   0.0
v   0.5  -0.86   0.0
v   1.0    0.0   0.0
```

The Story So Far...

Fixed-function Pipeline

Vertices

```
glBegin(GL_POINTS)
  ...
glEnd(GL_POINTS)
```

Geometry

- Model & View Transform
  - `glMatrixMult`
  - `gluLookAt`
- Projection
  - `gluPerspective`
- Vertex operations
- Lighting effects
- ...

Rasterization

- Conversion into pixels
- Texture
- Z-buffer
- Post-processed effects
- ...

In This Episode...

Programmable Pipeline

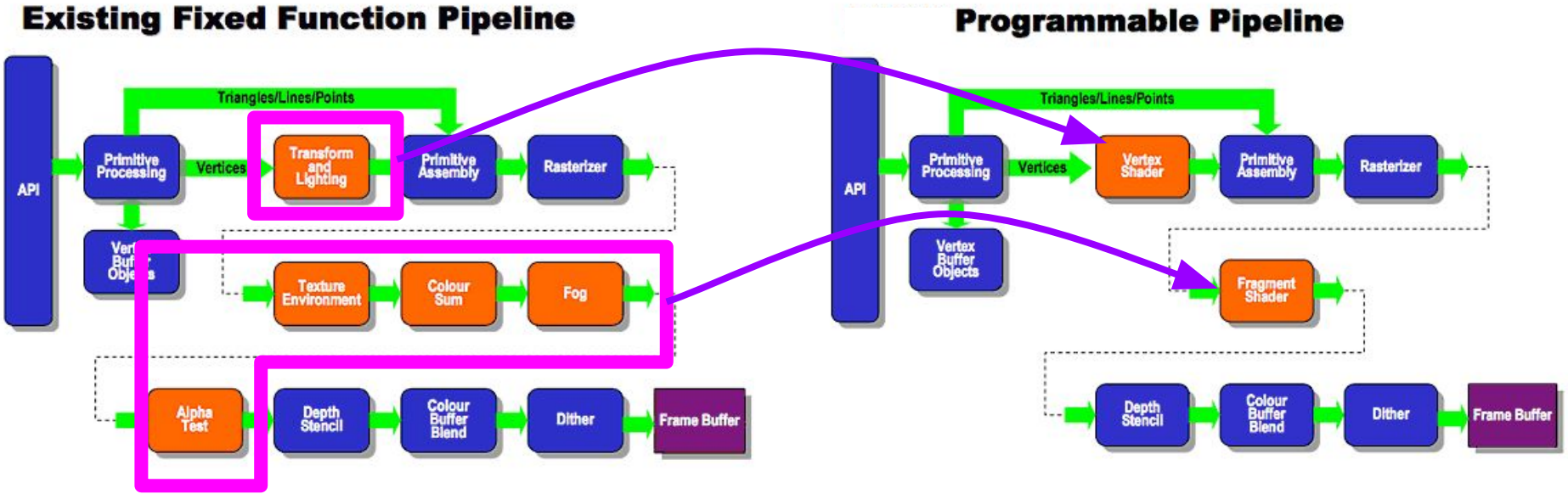
Vertices → Geometry → Rasterization

Vertex Buffer Object (VBO)

- Vertex Shader

- Fragment Shader

So What Changed?



Before We Venture Forth

- We need to make sure all our drivers are up-do-date before we can run modern OpenGL Commands
  - Windows machines make sure to update your graphics card drivers
  - Macs your OS X must be version 10.9 or higher.

VAOs/VBOs/EBOs

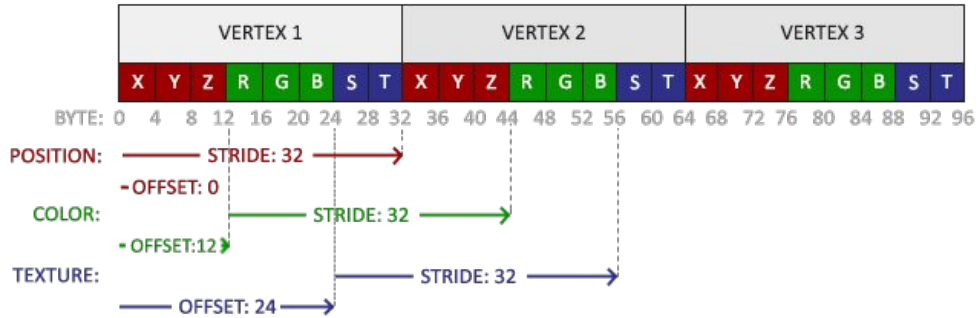
- VAOs (Vertex Array Object) tie together a multitude of buffers
- In the starter code, the Cube has two
  - VBO (Vertex Buffer Object)
    - Vertices
  - EBO (Element Buffer Object)
    - Face indices

VAOs/VBOs/EBOs

- VAOs (Vertex Array Object) tie together a multitude of buffers
- In the starter code, the Cube has two
  - VBO (Vertex Buffer Object)
    - Vertices
  - EBO (Element Buffer Object)
    - Face indices
- What else do we need for OBJs?

VAOs/VBOs/EBOs

- Can we make this scheme better?
  - Let’s make a struct to hold various forms of data unique to each vertex
  - http://learnopengl.com/#!Model-Loading/Mesh



Vertex Shader

- Manipulate geometry primitives vertices
- Transform vertex positions, normals
- Calculate colors, lighting, and camera effects
- Output any information that would be useful in the fragment shader

A Real Vertex Shader

shader.vert

```
#version 330 core

layout (location = 0) in vec3 position;

uniform mat4 MVP;

void main()
{
    gl_Position = MVP * vec4(position.x, position.y, position.z, 1.0);
}
```

We're going to use the vertex positions, given by the VBO

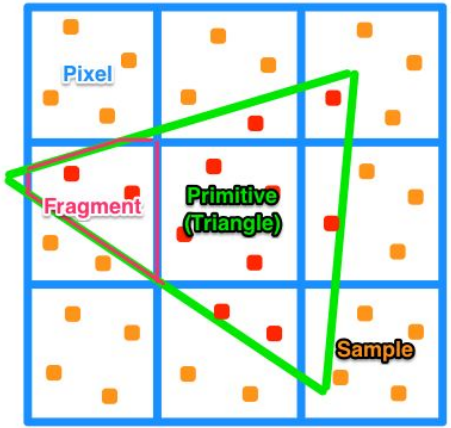
GLSL has some pre-built outputs: `gl_Position` being one

cube.cpp

```
void Cube::draw(GLuint shaderProgram)
{
        ...
        GLuint MatrixID =
                glGetUniformLocation(shaderProgram, "MVP");
        glUniformMatrix4fv(MatrixID, 1,
                        GL_FALSE, &MVP[0][0]);
        ...
}
```

Fragment Shader

- Manipulate a fragment
- Implement screen-space effects
- Output the final color of the fragment



A fragment is a partial pixel that is yet to be grid aligned.

A Real Fragment Shader

shader.frag

```
#version 330 core

out vec4 color;

void main()
{
   color = vec4(1.0f, 0.5f, 0.2f, 1.0f);
}
```

OpenGL will use the variable in position 0 to determine pixel color

Output the same RGBA color for all pixels

Some Code Tips

- Perform all object initialization in the initialize_objects() function in Window.cpp
  - Global pointer declaration is okay
  - Global pointer initialization is not
    - Perform all calls to new in initialize_objects()
  - Calling constructors of objects that perform gl*() function calls before GLFW is initialized will result in a segmentation fault

Thread 1: EXC_BAD_ACCESS (code=1, address=0x0)