OpenGL and GLSL

CS GY-6533

GLUT

OpenGL Utility Toolkit

All functions that start with the **glut** prefix are part of the GLUT API

Minimal basic GLUT application.

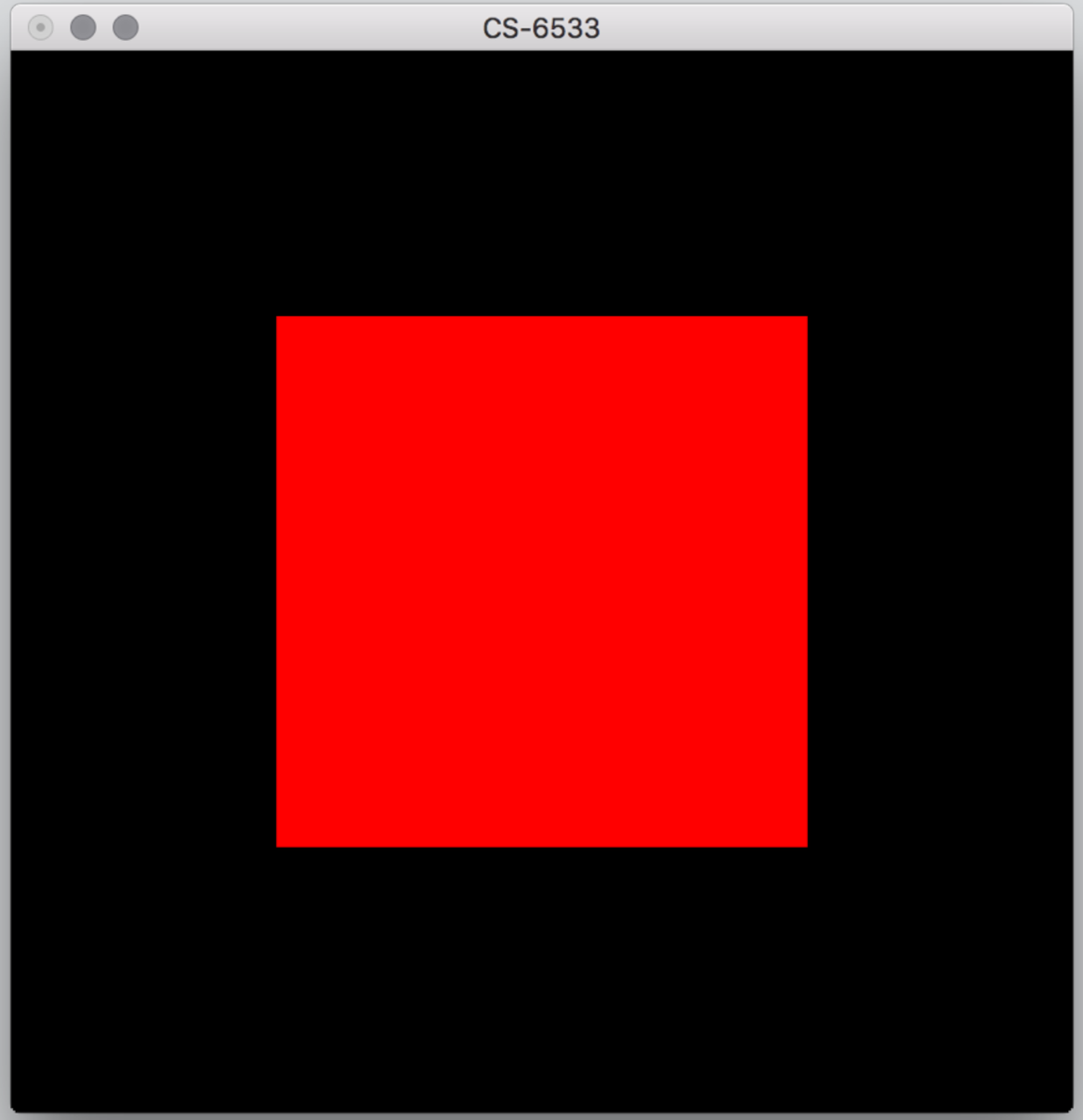
```c
#include <glut.h>

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glutSwapBuffers();
}

void init() {
}

void reshape(int w, int h) {
    glViewport(0, 0, w, h);
}

void idle(void) {
    glutPostRedisplay();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("CS-6533");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(idle);

    init();
    glutMainLoop();
    return 0;
}
```

OpenGL

Open Graphics Library

Part 1

Two triangles.


CS-6533

Our objectives:

1. Write and load a simple **shader program**.
2. Get the **location** of the vertex **“position” attribute** in the loaded program.
3. Create an **array** of **position attributes** as a **vertex buffer object** that describe two triangles.
4. Render the **vertex buffer object** using the **shader program** by **binding it to the location** of the **position attribute**.

A basic shader program.

A shader program always consists of a **vertex shader** and a **fragment shader**. In OpenGL shader programs are written in the GLSL language.

vertex.glsl

```
attribute vec4 position;
void main() {
    gl_Position = position;
}
```

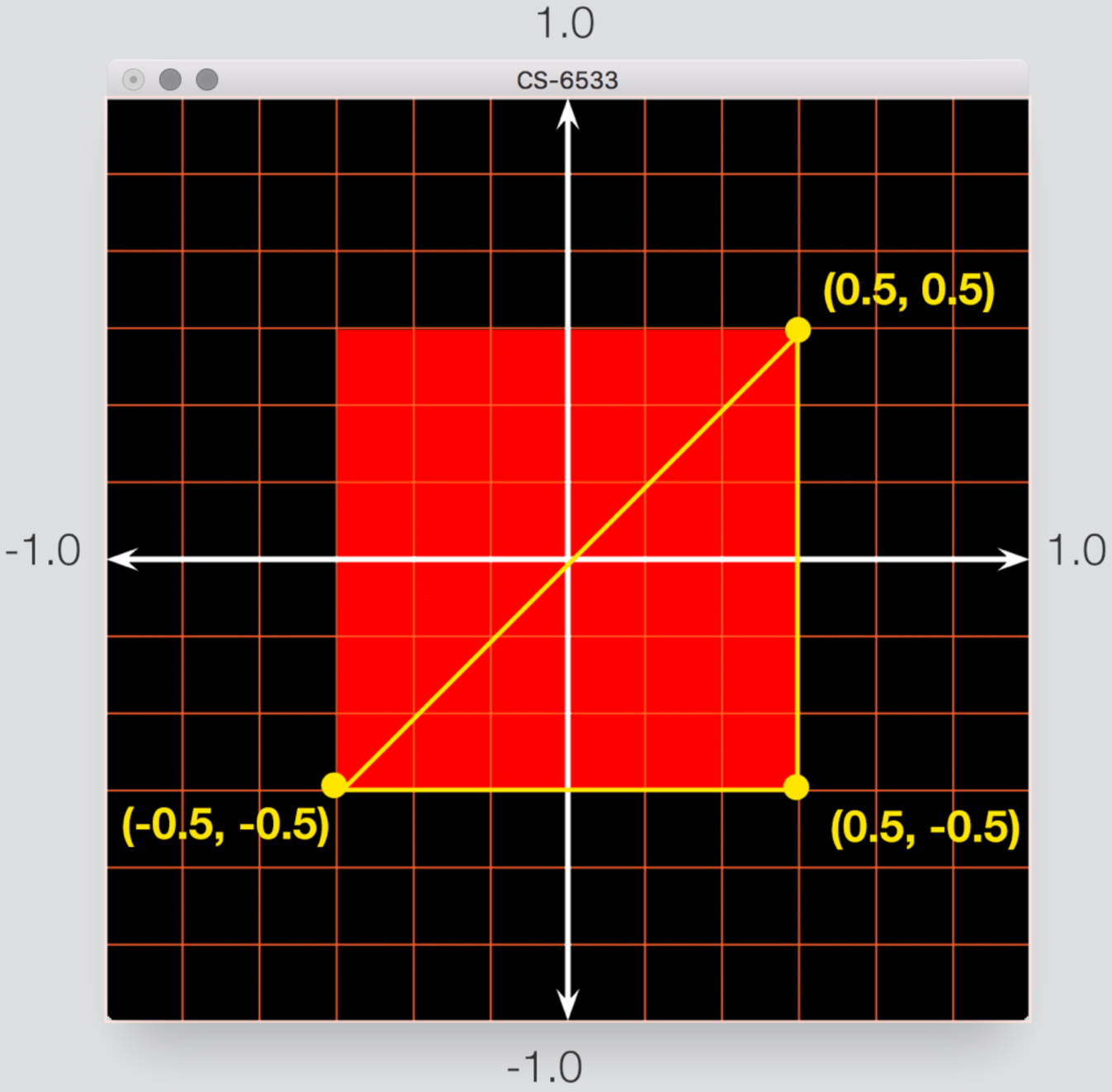
fragment.glsl

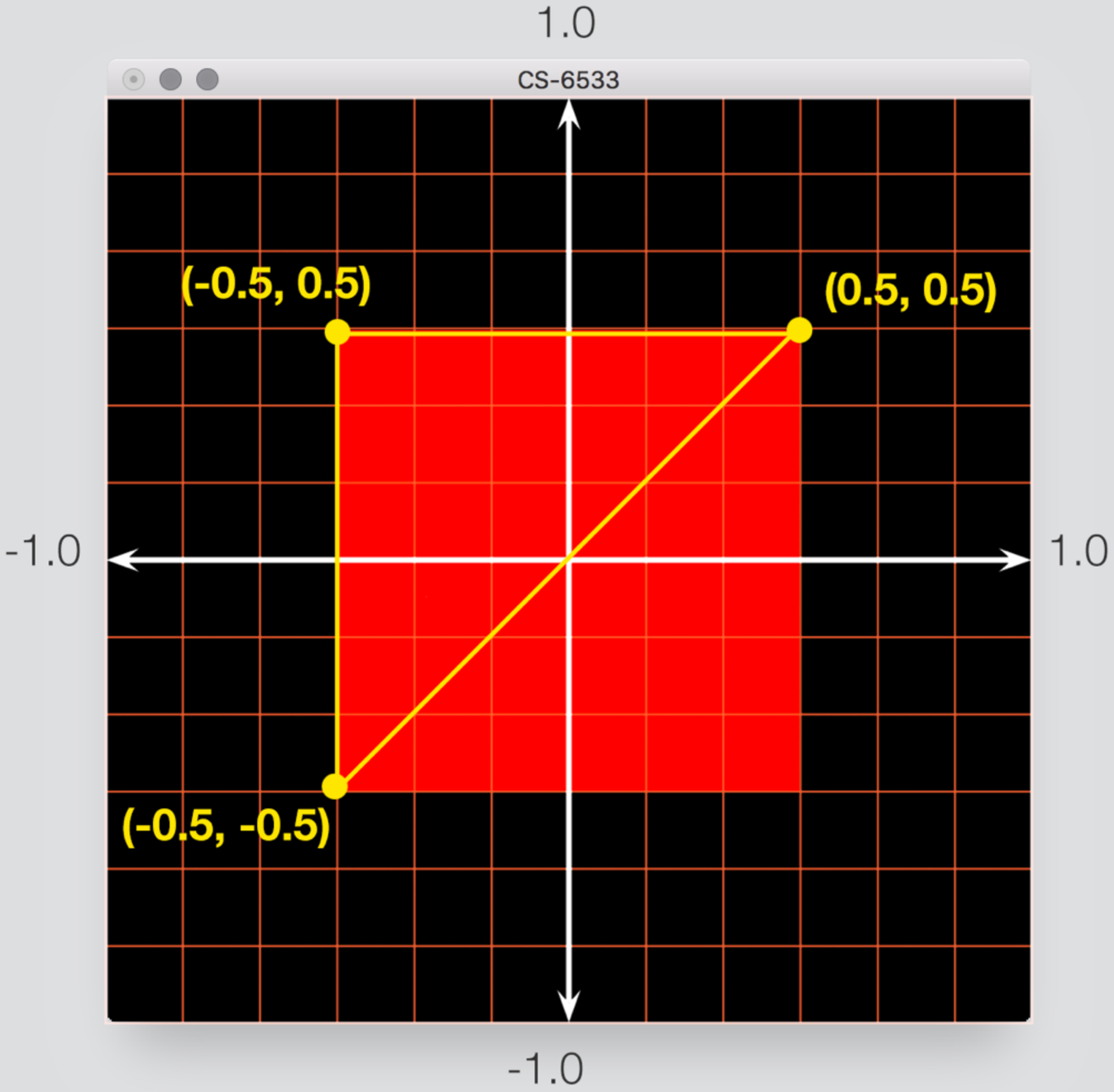
```
void main() {
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Loading a shader program.

```
#include "glsupport.h"

GLuint program;


void init() {
    program = glCreateProgram();
    readAndCompileShader(program, "vertex.glsl", "fragment.glsl");
}
```

Creating a vertex buffer object (VBO).


1.0
CS-6533
(0.5, 0.5)
-1.0
1.0
(-0.5, -0.5)
(0.5, -0.5)
-1.0


1.0
CS-6533
(-0.5, 0.5)
(0.5, 0.5)
-1.0
1.0
(-0.5, -0.5)
-1.0

```
GLuint program;
GLuint vertPositionVBO;

void init() {
    program = glCreateProgram();
    readAndCompileShader(program, "vertex.glsl", "fragment.glsl");

    glGenBuffers(1, &vertPositionVBO);
    glBindBuffer(GL_ARRAY_BUFFER, vertPositionVBO);
    GLfloat sqVerts[12] = {
        -0.5f, -0.5f,
        0.5f, 0.5f,
        0.5f, -0.5f,

        -0.5f, -0.5f,
        -0.5f, 0.5f,
        0.5f, 0.5f
    };
    glBufferData(GL_ARRAY_BUFFER, 12*sizeof(GLfloat), sqVerts, GL_STATIC_DRAW);
}
```

New code **in bold.**

Getting an attribute location in program.

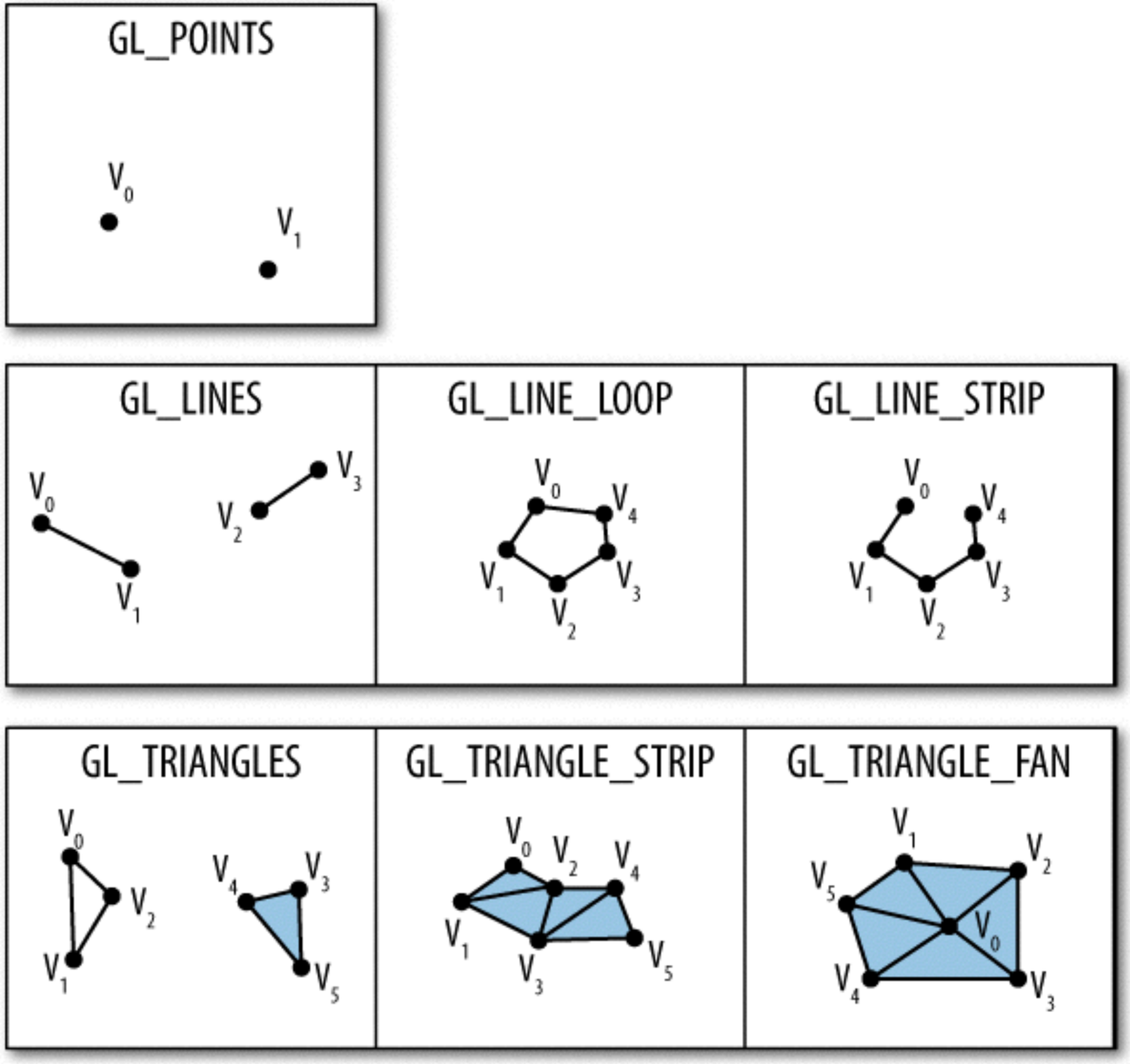
```
GLuint program;
GLuint vertPositionVBO;
GLuint positionAttribute;

void init() {
    program = glCreateProgram();
    readAndCompileShader(program, "vertex.glsl", "fragment.glsl");

    glUseProgram(program);
    positionAttribute = glGetAttribLocation(program, "position");

    glGenBuffers(1, &vertPositionVBO);
    glBindBuffer(GL_ARRAY_BUFFER, vertPositionVBO);
    GLfloat sqVerts[12] = {
        -0.5f, -0.5f,
        0.5f, 0.5f,
        0.5f, -0.5f,

        -0.5f, -0.5f,
        -0.5f, 0.5f,
        0.5f, 0.5f
    };
    glBufferData(GL_ARRAY_BUFFER, 12*sizeof(GLfloat), sqVerts, GL_STATIC_DRAW);
}
```

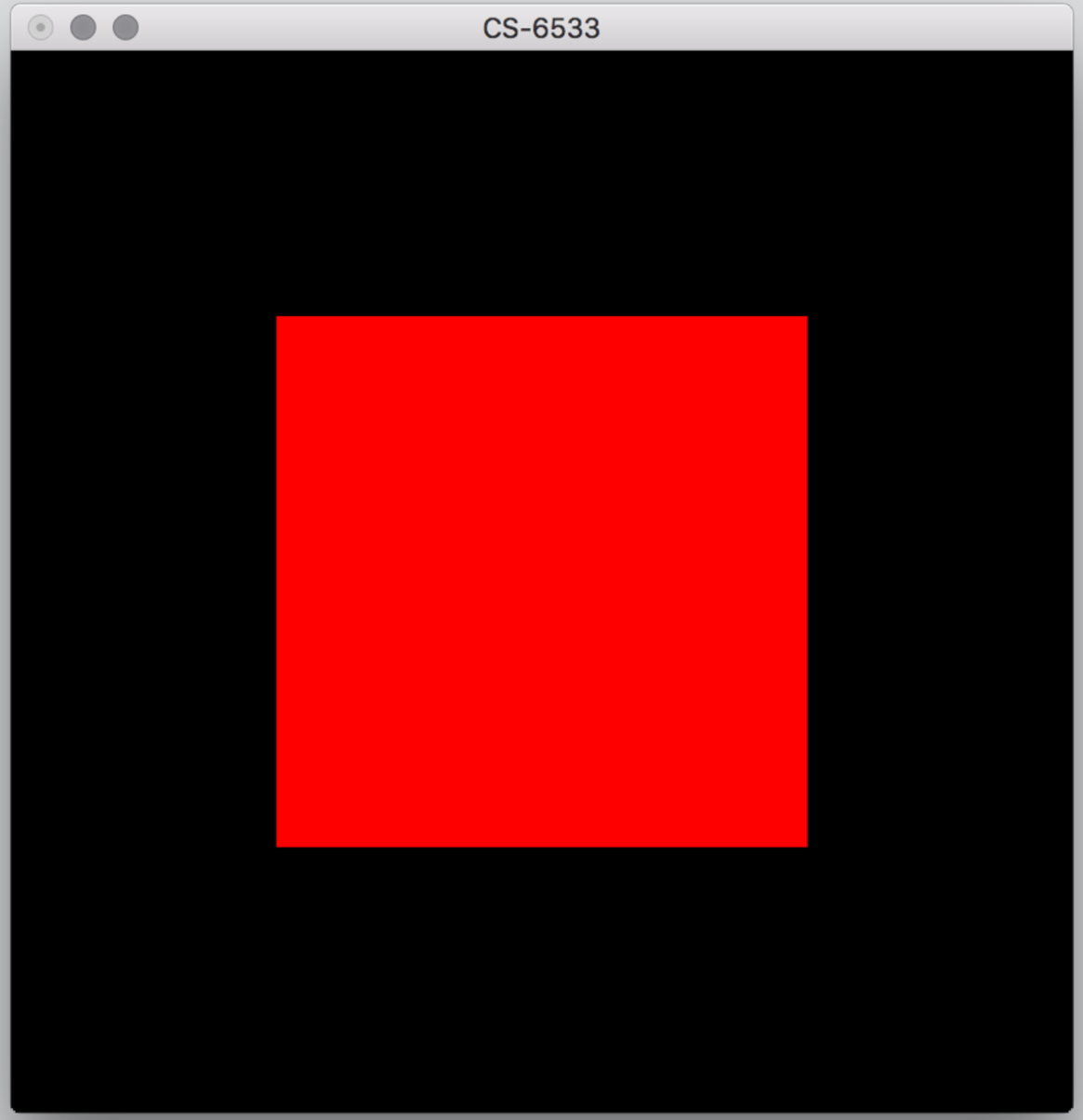
New code **in bold.**

Rendering the vertex buffer.

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

    glUseProgram(program);
    glBindBuffer(GL_ARRAY_BUFFER, vertPositionVBO);

    glVertexAttribPointer(positionAttribute, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(positionAttribute);
    glDrawArrays(GL_TRIANGLES, 0, 6);

    glDisableVertexAttribArray(positionAttribute);

    glutSwapBuffers();
}
```

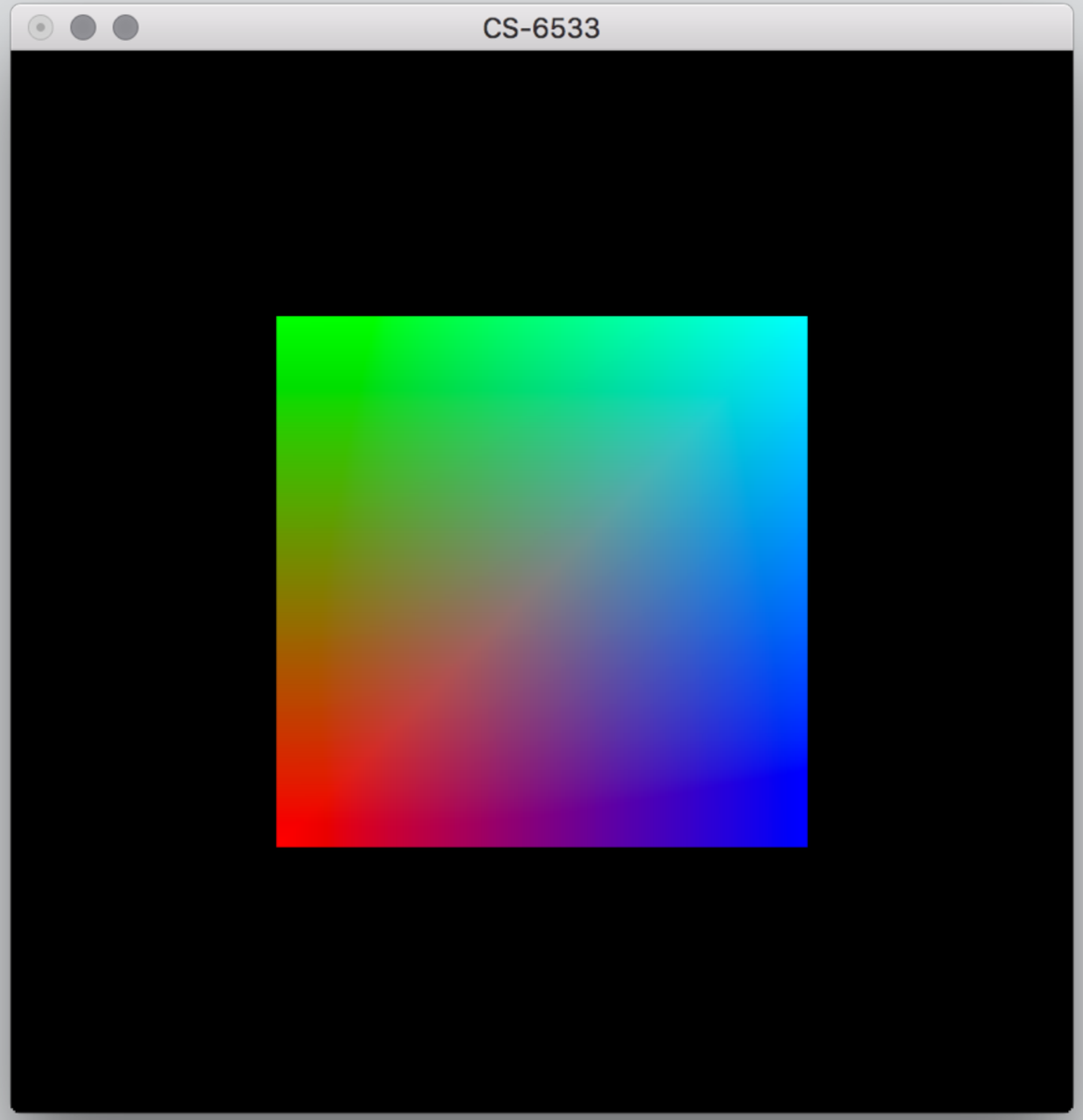
New code **in bold.**


GL_POINTS
V0
V1
GL_LINES
V0
V1
V2
V3
GL_LINE_LOOP
V0
V1
V2
V3
V4
GL_LINE_STRIP
V0
V1
V2
V3
V4
GL_TRIANGLES
V0
V1
V2
V3
V4
V5
GL_TRIANGLE_STRIP
V0
V1
V2
V3
V4
V5
GL_TRIANGLE_FAN
V0
V1
V2
V3
V4
V5


CS-6533

Part 2

Attributes and varying variables.


CS-6533

Our objectives:

1. Add a **“color” attribute** to our **vertex shader program**.
2. Add a color **varying variable** to the vertex shader and set it to the “color” attribute.
3. Add a color **varying variable** to the fragment shader and set our final color to it.
4. Get the **location** of the **“color” attribute** in the loaded program.
5. Create an **array** of **color attributes** as a **buffer object** that describe colors for each vertex.
6. Bind the **color attributes buffer object** to the **color attribute location** when we render.

Our new vertex and fragment shaders.

vertex.glsl

```
attribute vec4 position;
attribute vec4 color;

varying vec4 varyingColor;

void main() {
    varyingColor = color;
    gl_Position = position;
}
```

fragment.glsl

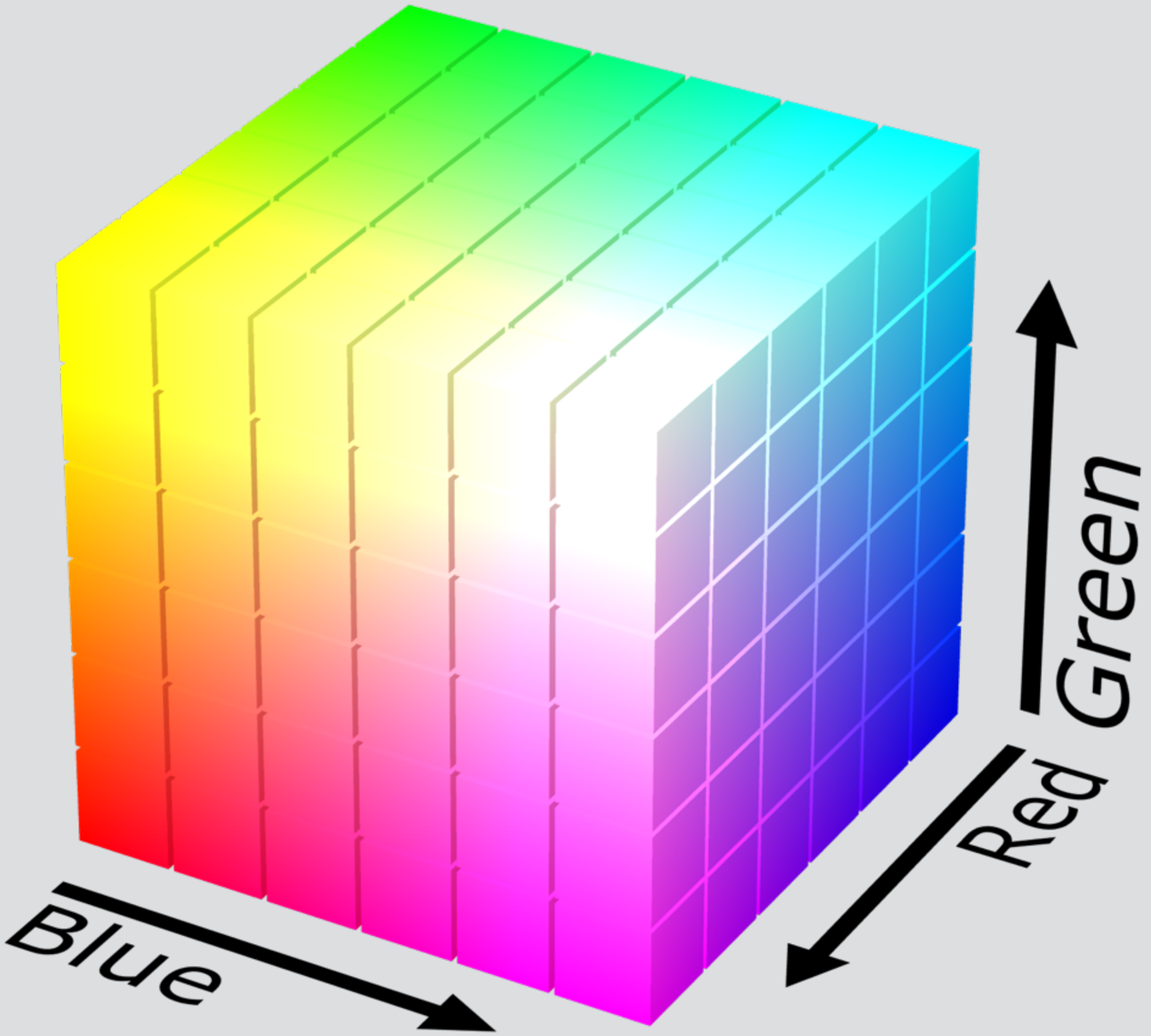
```
varying vec4 varyingColor;

void main() {
    gl_FragColor = varyingColor;
}
```

Getting the location of the new attribute.

```
GLuint positionAttribute;
GLuint colorAttribute;



...


 glUseProgram(program);
 positionAttribute = glGetAttribLocation(program, "position");
 colorAttribute = glGetAttribLocation(program, "color");
```

Creating a buffer of vertex colors.

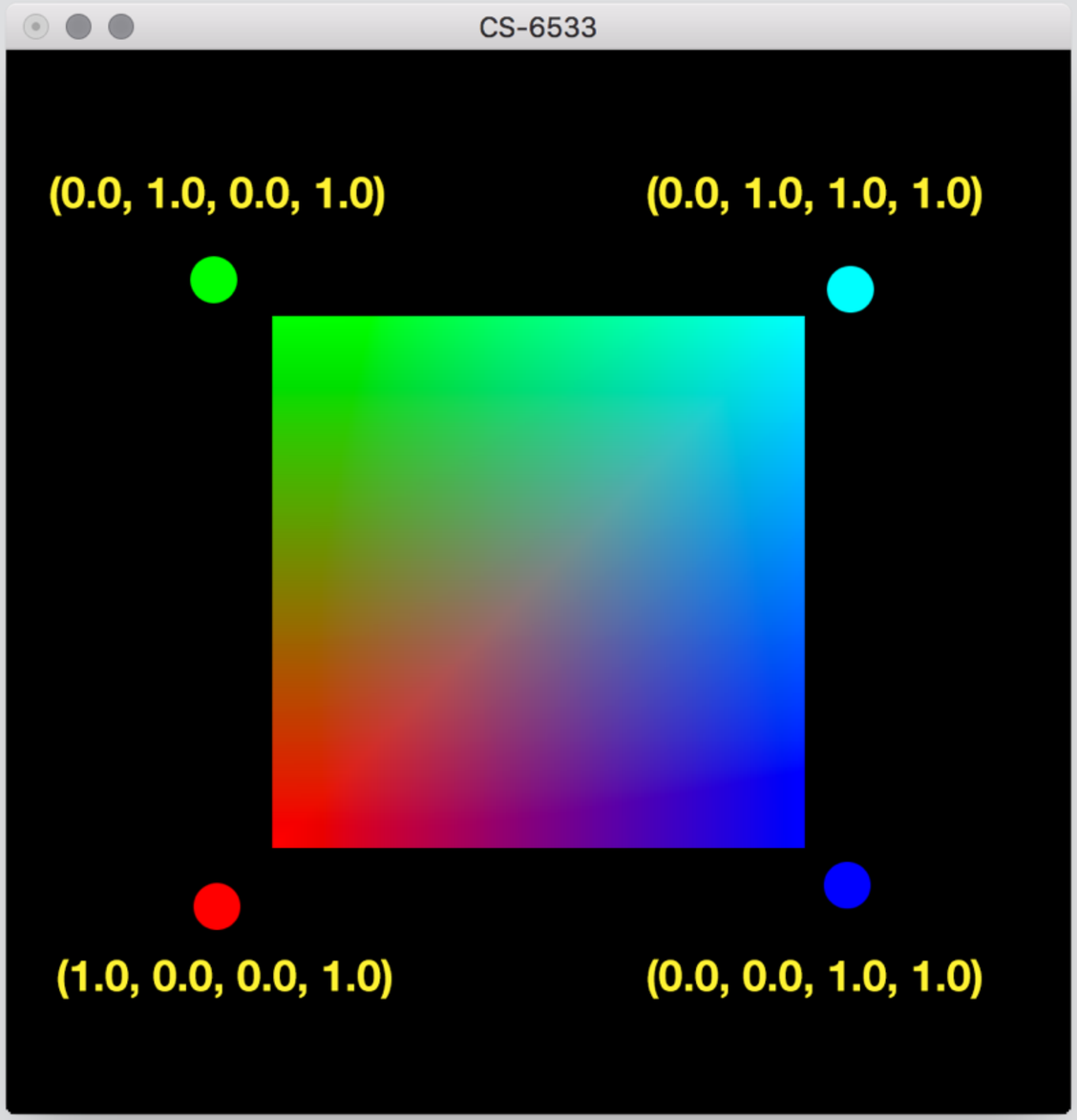
How do we describe a color.


Green
Red
Blue

- In OpenGL, each color component is a value between **0.0 and 1.0**
- A color is described by **4 floating point values**: **RED (R), GREEN (G), BLUE (B)** and the color's transparency, also known as **ALPHA** (A).

For example, a full red color that is opaque is described as (1.0, 0.0, 0.0, 1.0)

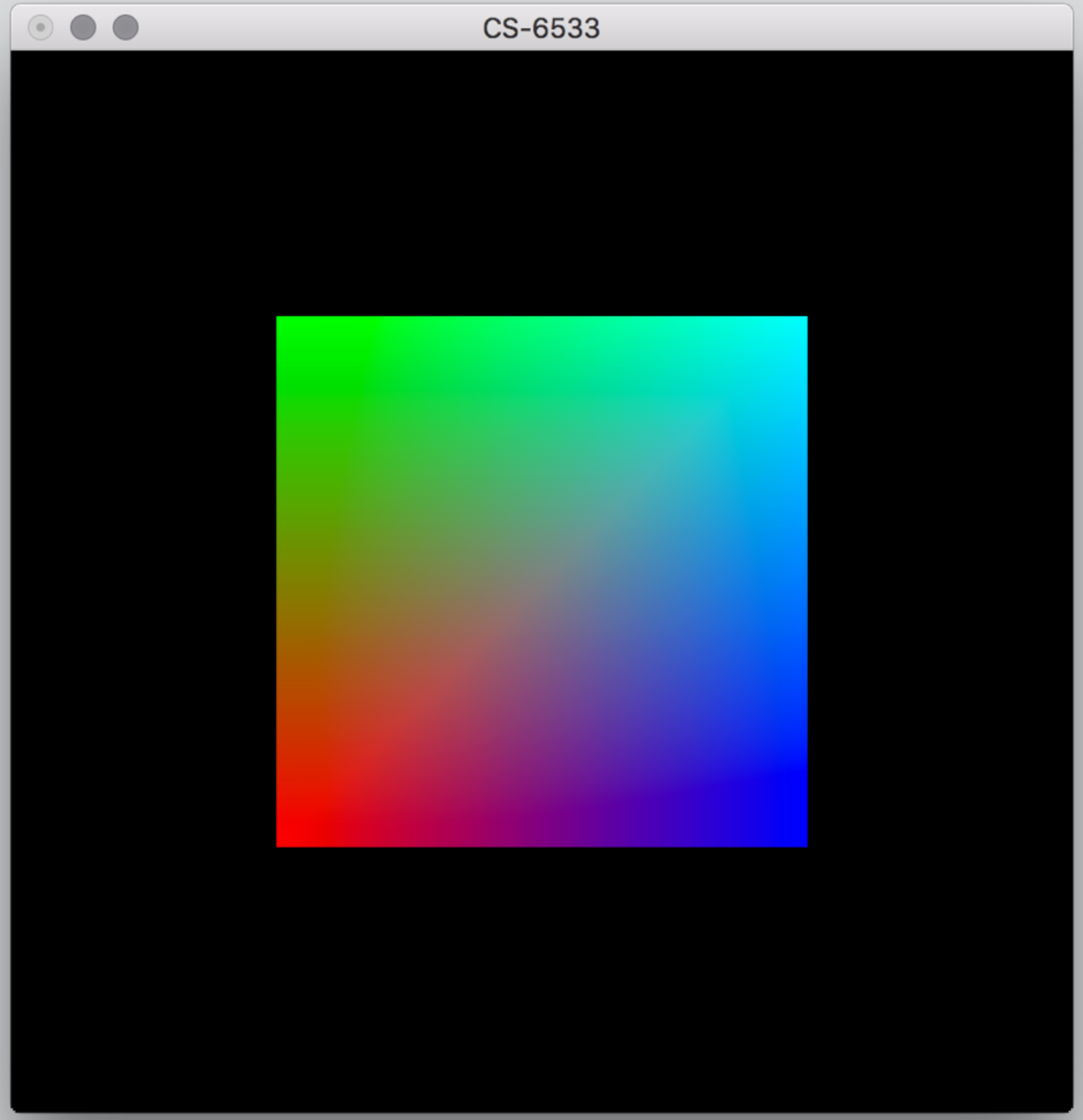
A half-transparent yellow color is (1.0, 1.0, 0.0, 0.5)


CS-6533
(0.0, 1.0, 0.0, 1.0)
(0.0, 1.0, 1.0, 1.0)
(1.0, 0.0, 0.0, 1.0)
(0.0, 0.0, 1.0, 1.0)

```
GLuint vertPositionVBO;
GLuint vertColorVBO;


...


glGenBuffers(1, &vertColorVBO);
glBindBuffer(GL_ARRAY_BUFFER, vertColorVBO);

GLfloat sqColors[24] = {
    1.0f, 0.0f, 0.0f, 1.0f,
    0.0f, 1.0f, 1.0f, 1.0f,
    0.0f, 0.0f, 1.0f, 1.0f,

    1.0f, 0.0f, 0.0f, 1.0f,
    0.0f, 1.0f, 0.0f, 1.0f,
    0.0f, 1.0f, 1.0f, 1.0f
};
glBufferData(GL_ARRAY_BUFFER, 24*sizeof(GLfloat), sqColors, GL_STATIC_DRAW);
```

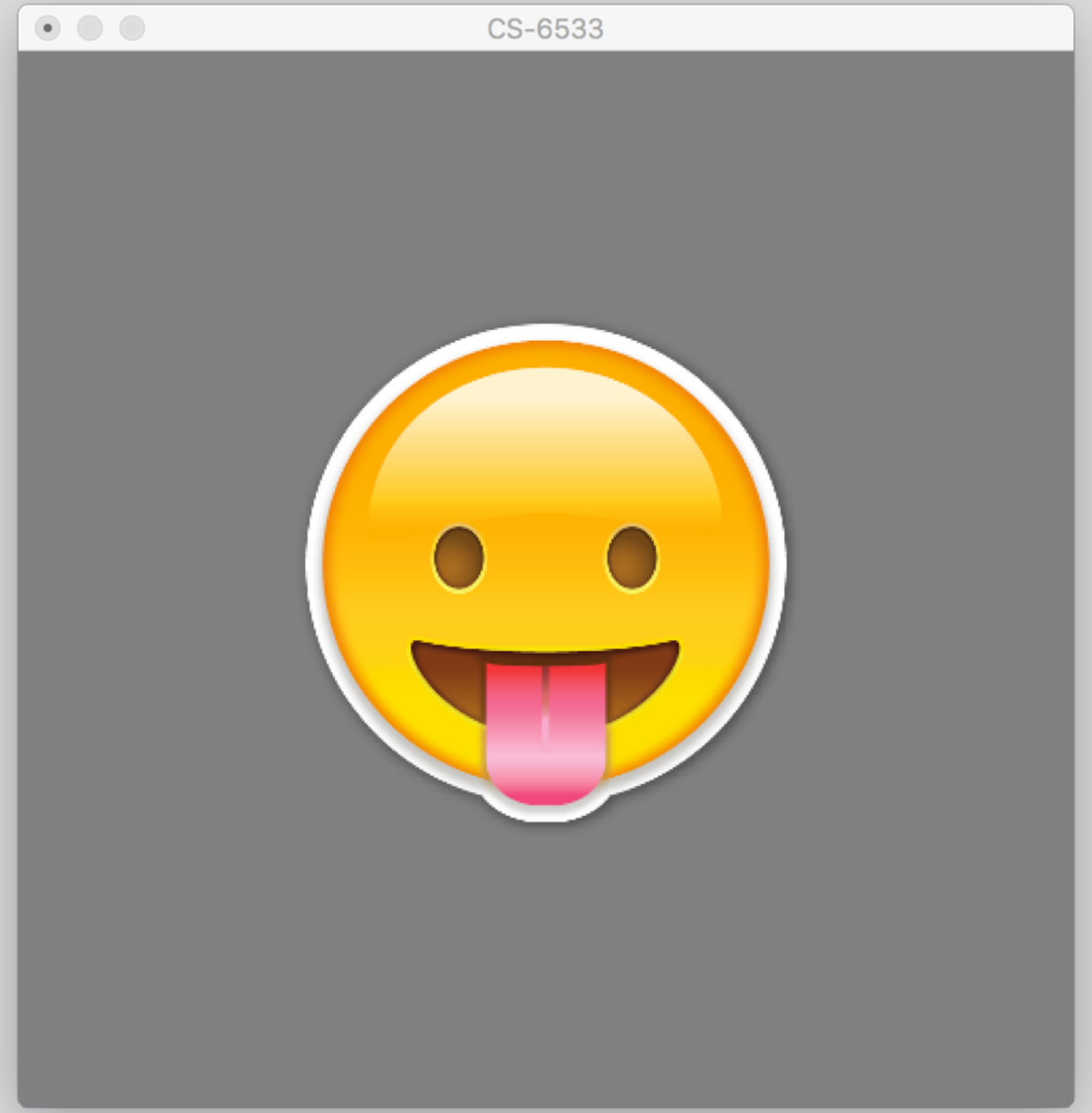
Binding our vertex color buffer to the attribute.

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

    glBindBuffer(GL_ARRAY_BUFFER, vertPositionVBO);
    glVertexAttribPointer(positionAttribute, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(positionAttribute);

    glBindBuffer(GL_ARRAY_BUFFER, vertColorVBO);
    glVertexAttribPointer(colorAttribute, 4, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(colorAttribute);

    glDrawArrays(GL_TRIANGLES, 0, 6);
    glDisableVertexAttribArray(positionAttribute);
    glDisableVertexAttribArray(colorAttribute);

    glutSwapBuffers();
}
```


CS-6533

Part 3

Shader uniforms and textures.


CS-6533

Our objectives:

1. Add a **texture coordinate attribute** to the vertex shader.
2. Add a texture coordinate **varying variable** to the vertex shader and set it to the **texture coordinate attribute**.
3. Add a texture coordinate **varying variable** to the fragment shader.
4. Add a **texture uniform** to the fragment shader and **sample it at the varying texture coordinate**.
5. Get the **location** of the **texture coordinate attribute** in the loaded program.
6. Create an **array** of **texture coordinates** as a **buffer object** that describe how the vertices map to the image.
7. Load an image file into memory and create an OpenGL **texture** from it.
8. Bind the **texture coordinate buffer object** to the **texture coordinate attribute location** when we render.
9. Get the **location of the texture uniform** and **bind our texture location** to it when we render.

Using textures in our shaders.

vertex.glsl

```
attribute vec4 position;
attribute vec2 texCoord;

varying vec2 varyingTexCoord;

void main() {
    varyingTexCoord = texCoord;
    gl_Position = position;
}
```

fragment.glsl

```
varying vec2 varyingTexCoord;
uniform sampler2D texture;

void main() {
    gl_FragColor = texture2D(texture, varyingTexCoord);
}
```

Getting locations of the new attribute on the C++ side.

```
GLuint positionAttribute;
GLuint texCoordAttribute;
```

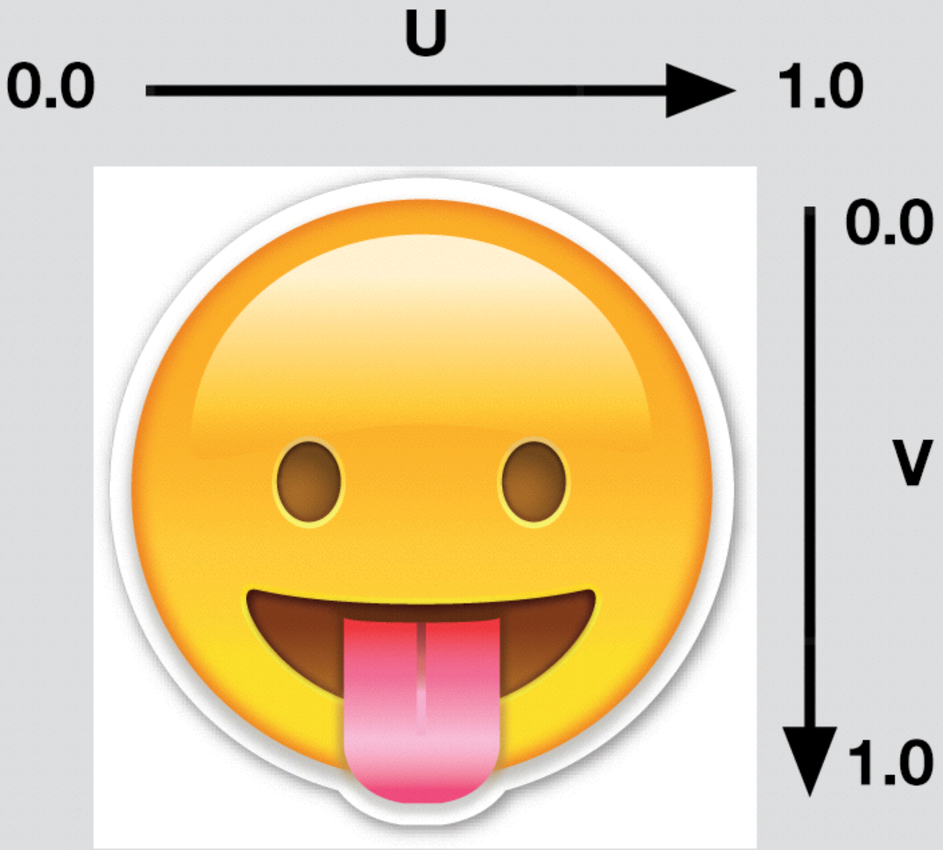
...

```
glUseProgram(program);
positionAttribute = glGetAttribLocation(program, "position");
texCoordAttribute = glGetAttribLocation(program, "texCoord");
```

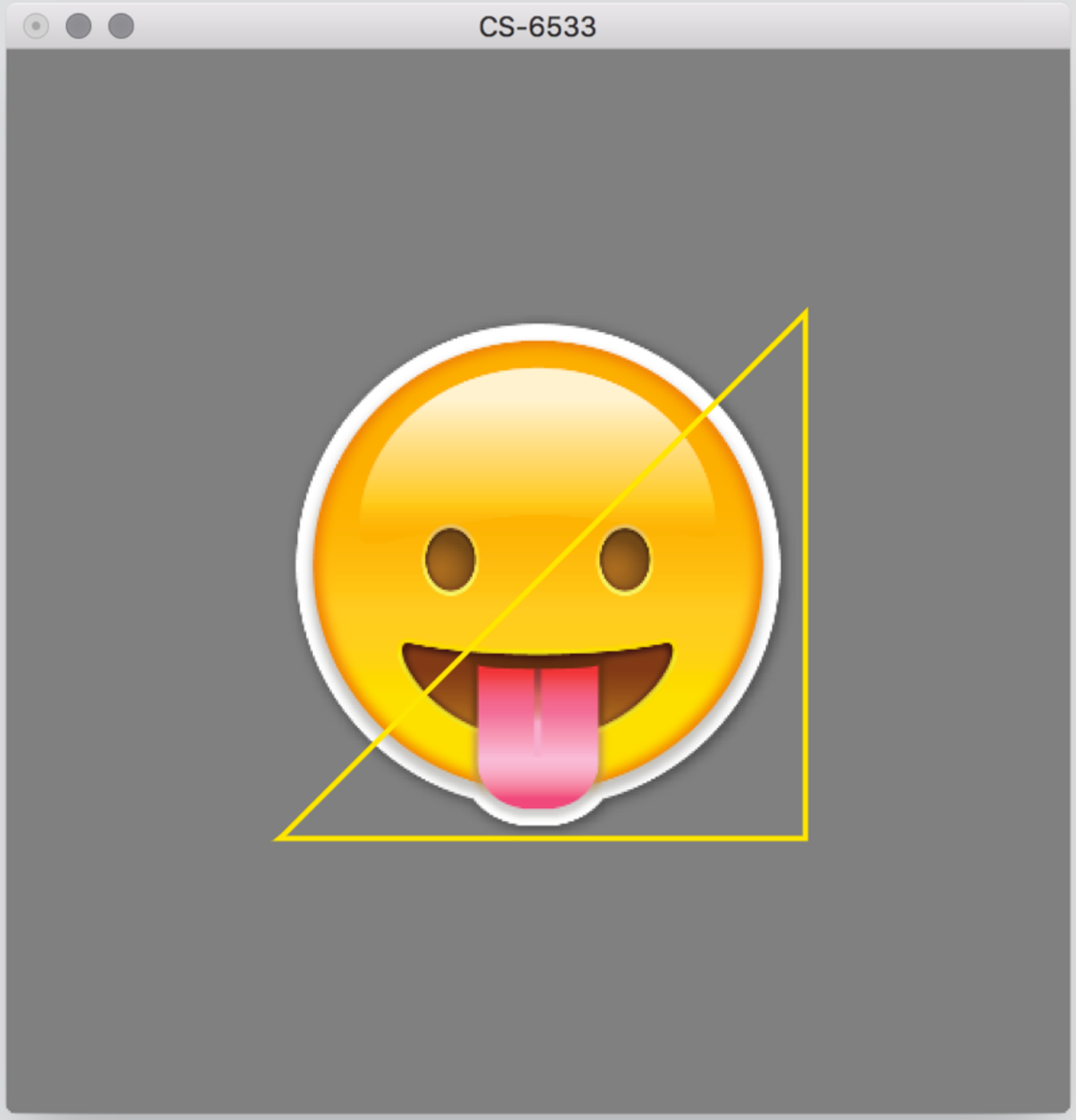
Creating our texture coordinate attribute buffer.

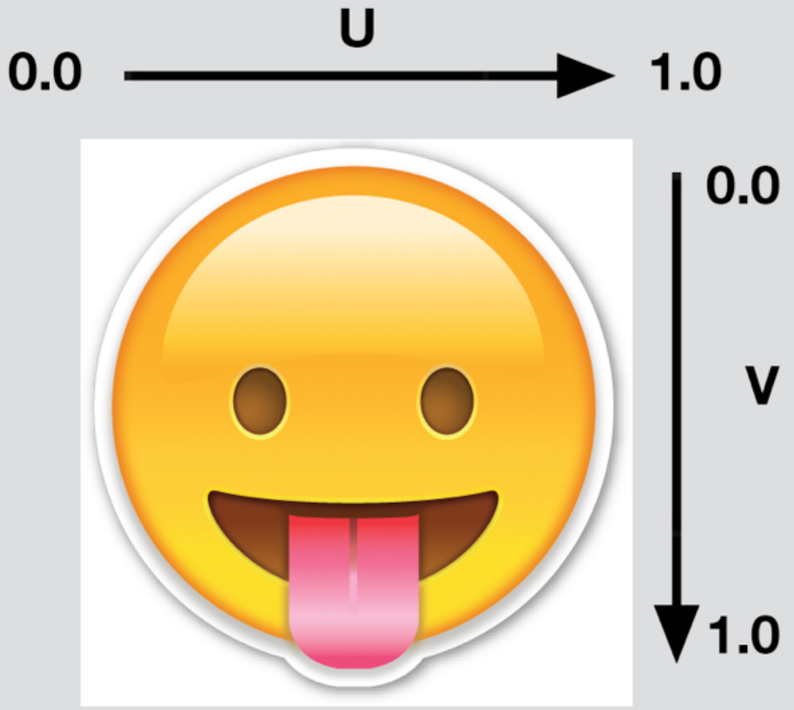
Describing image coordinates.

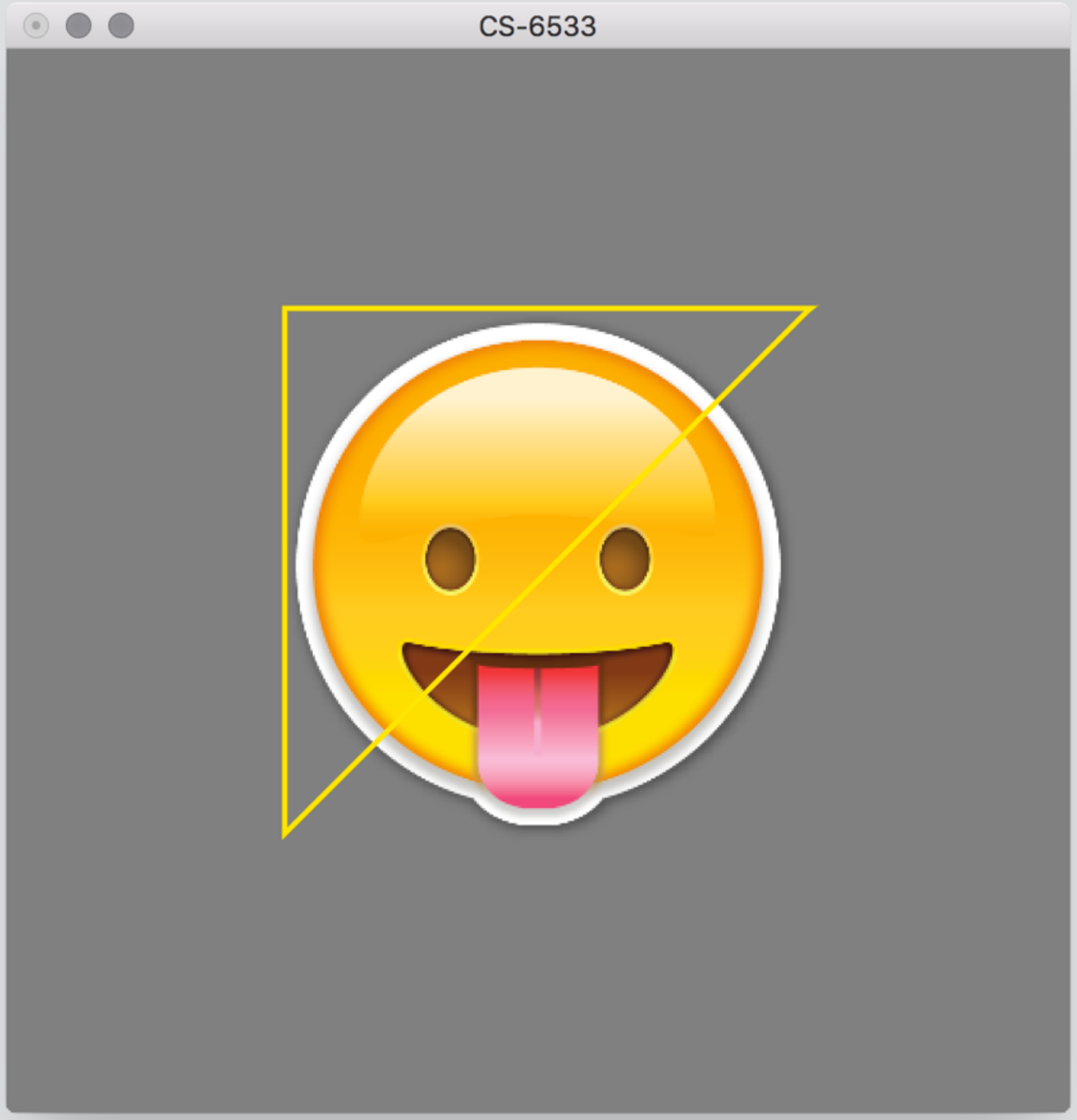
Texture coordinates.

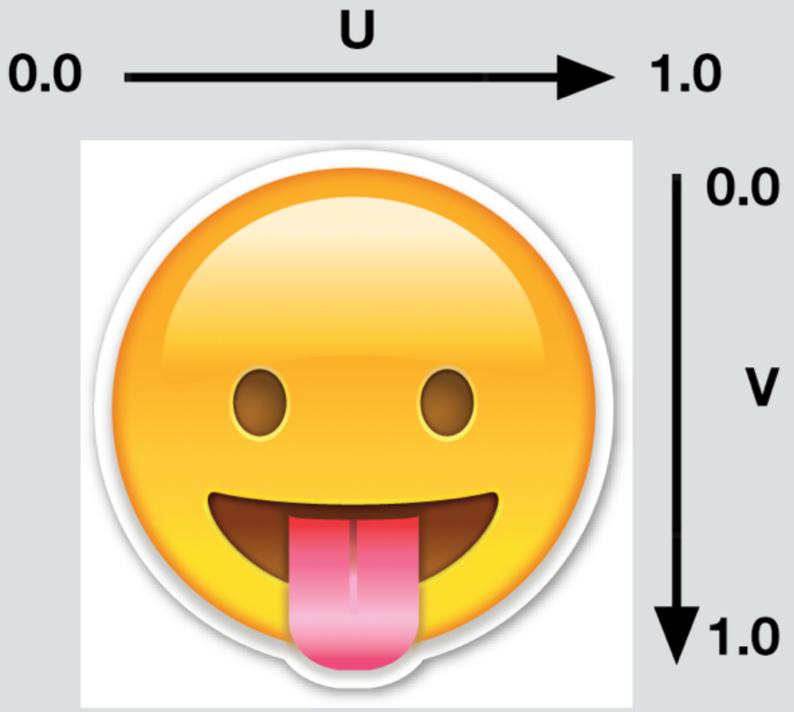


- The horizontal image coordinate is usually called U and vertical coordinate is usually called V. (Sometimes they are called UV coordinates).
- 2-dimensional coordinate independent of the image resolution.
- Normalized 0.0 to 1.0 floating point values. (If you go beyond 1.0, the texture repeats).


CS-6533


U
0.0
1.0
0.0
V
1.0


CS-6533


U
0.0
1.0
0.0
V
1.0

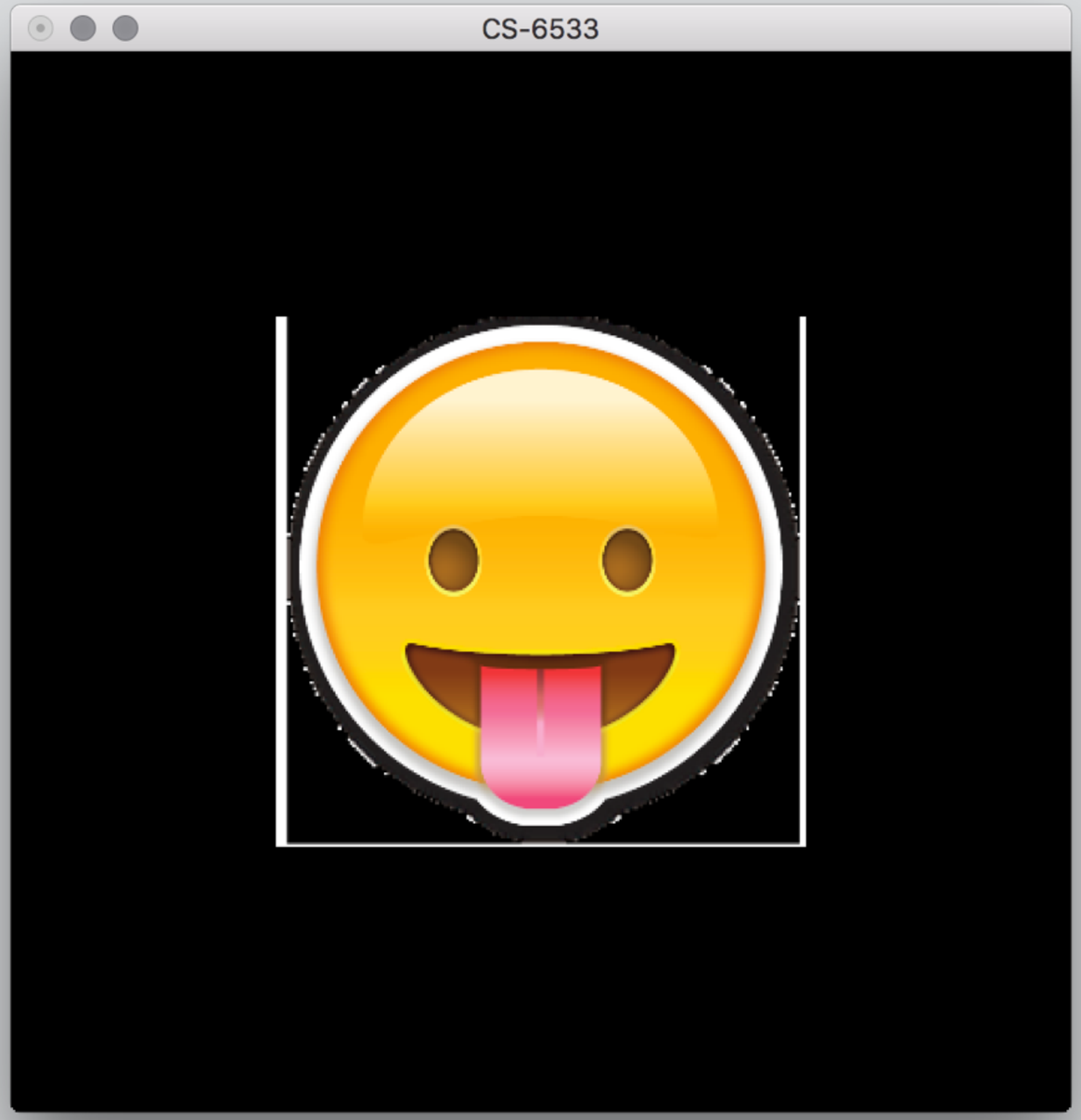
```
GLuint vertPositionVBO;
GLuint vertTexCoordVBO;


...


glGenBuffers(1, &vertTexCoordVBO);
glBindBuffer(GL_ARRAY_BUFFER, vertTexCoordVBO);
GLfloat sqTexCoords[12] = {
    0.0f, 1.0f,
    1.0f, 0.0f,
    1.0f, 1.0f,

    0.0f, 1.0f,
    0.0f, 0.0f,
    1.0f, 0.0f,
};
glBufferData(GL_ARRAY_BUFFER, 12*sizeof(GLfloat), sqTexCoords, GL_STATIC_DRAW);
```

Loading an image into a texture.

```
GLuint emojiTexture;

...

emojiTexture = loadGLTexture("emoji.png");
```

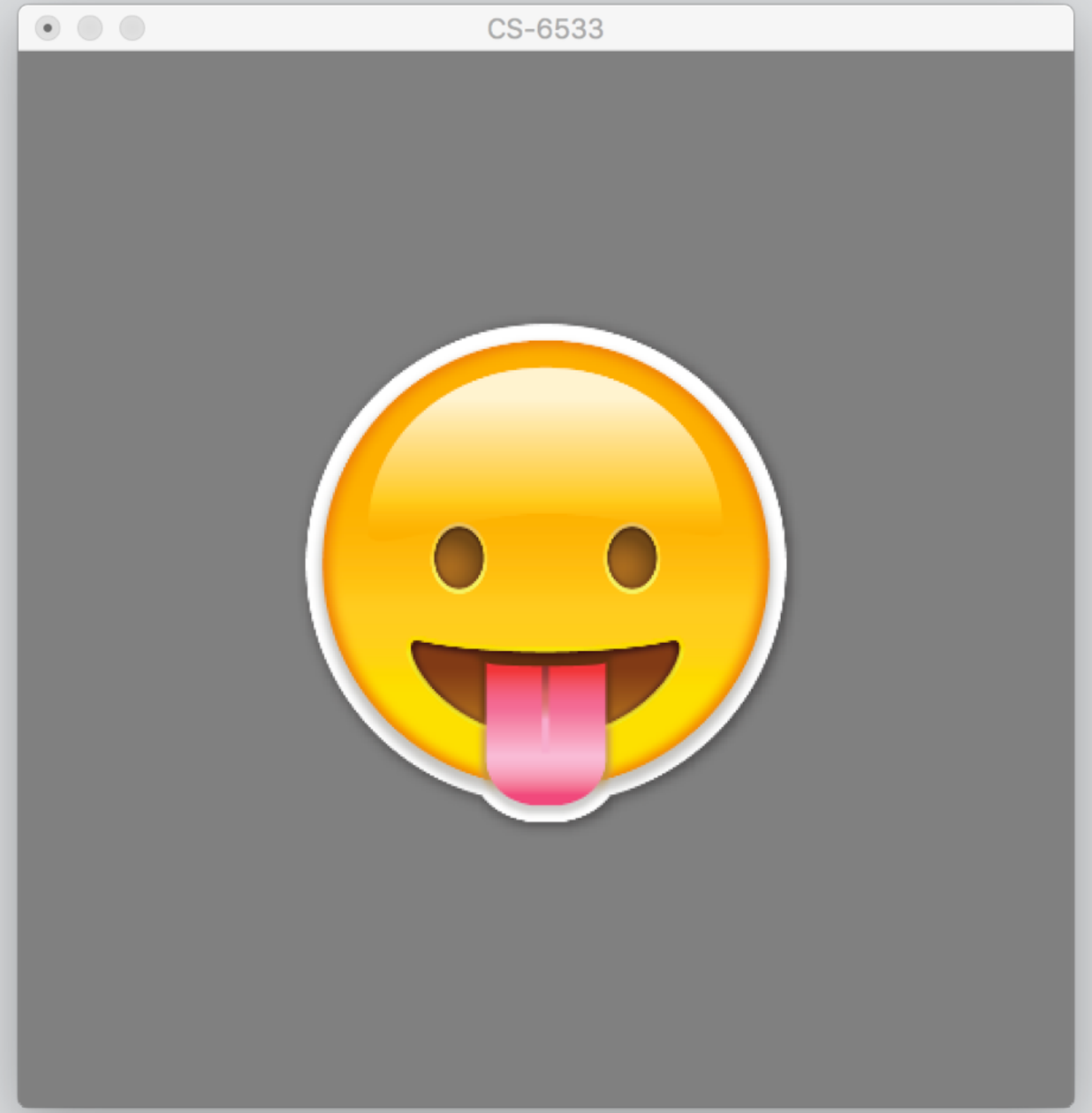
Binding our texture and texture coordinate buffer.

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

    glBindBuffer(GL_ARRAY_BUFFER, vertPositionVBO);
    glVertexAttribPointer(positionAttribute, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(positionAttribute);

    glBindBuffer(GL_ARRAY_BUFFER, vertTexCoordVBO);
    glVertexAttribPointer(texCoordAttribute, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(texCoordAttribute);

    glBindTexture(GL_TEXTURE_2D, emojiTexture);

    glDrawArrays(GL_TRIANGLES, 0, 6);

    glDisableVertexAttribArray(positionAttribute);
    glDisableVertexAttribArray(texCoordAttribute);

    glutSwapBuffers();
}
```


CS-6533

Blending

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glClearColor(0.5, 0.5, 0.5, 1.0); // set the clear color
```


CS-6533

Part 4

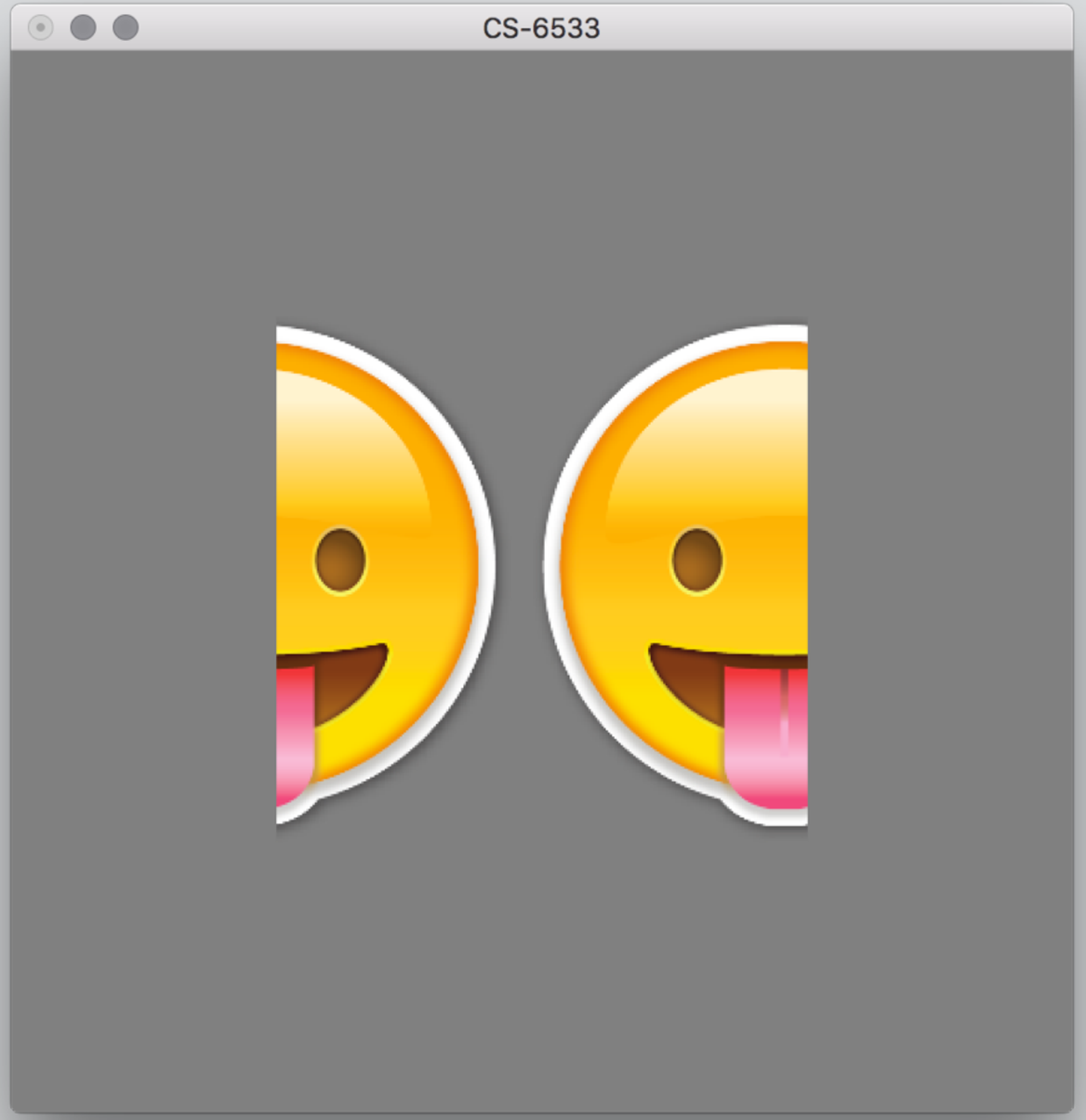
Shader uniforms and keeping time.

fragment.glsl

```
varying vec2 varyingTexCoord;
uniform sampler2D texture;

uniform float time;

void main() {
    vec2 texCoord = vec2(varyingTexCoord.x + time, varyingTexCoord.y);
    gl_FragColor = texture2D(texture, texCoord);
}
```

```
GLuint timeUniform;

timeUniform = glGetUniformLocation(program, "time");
```

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

    int timeSinceStart = glutGet(GLUT_ELAPSED_TIME);
    glUniform1f(timeUniform, (float)timeSinceStart/1000.0f);

    glBindBuffer(GL_ARRAY_BUFFER, vertPositionVBO);
    glVertexAttribPointer(positionAttribute, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(positionAttribute);

    glBindBuffer(GL_ARRAY_BUFFER, vertTexCoordVBO);
    glVertexAttribPointer(texCoordAttribute, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(texCoordAttribute);
    glBindTexture(GL_TEXTURE_2D, emojiTexture);

    glDrawArrays(GL_TRIANGLES, 0, 6);

    glDisableVertexAttribArray(positionAttribute);
    glDisableVertexAttribArray(texCoordAttribute);

    glutSwapBuffers();
}
```


CS-6533

Part 5

Input and more uniforms.

Registering input functions.

```
void keyboard(unsigned char key, int x, int y) {
}

void mouse(int button, int state, int x, int y) {
}

void mouseMove(int x, int y) {
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("CS-6533");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(idle);

    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMotionFunc(mouseMove);

    init();
    glutMainLoop();
    return 0;
}
```

Keyboard input.

```
float textureOffset = 0.0;


void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);

    glUniform1f(timeUniform, textureOffset);

    // do our rendering

    glutSwapBuffers();
}


void keyboard(unsigned char key, int x, int y) {
    switch(key) {
        case 'a':
            textureOffset += 0.02;
        break;
        case 'd':
            textureOffset -= 0.02;
        break;
    }
}
```

Mouse input and positioning.

vertex.glsl

```
attribute vec4 position;
attribute vec2 texCoord;

varying vec2 varyingTexCoord;

uniform vec2 modelPosition;

void main() {
    varyingTexCoord = texCoord;
    gl_Position = vec4(modelPosition.x, modelPosition.y, 0.0, 0.0) + position;
}
```

```
void mouse(int button, int state, int x, int y) {
    float newPositionX = (float)x/250.0f - 1.0f;
    float newPositionY = (1.0-(float)y/250.0);
    glUniform2f(positionUniform, newPositionX, newPositionY);
}

void mouseMove(int x, int y) {
    float newPositionX = (float)x/250.0f - 1.0f;
    float newPositionY = (1.0-(float)y/250.0);
    glUniform2f(positionUniform, newPositionX, newPositionY);
}
```

Mouse coordinates are in pixels based on the window.

Part 5

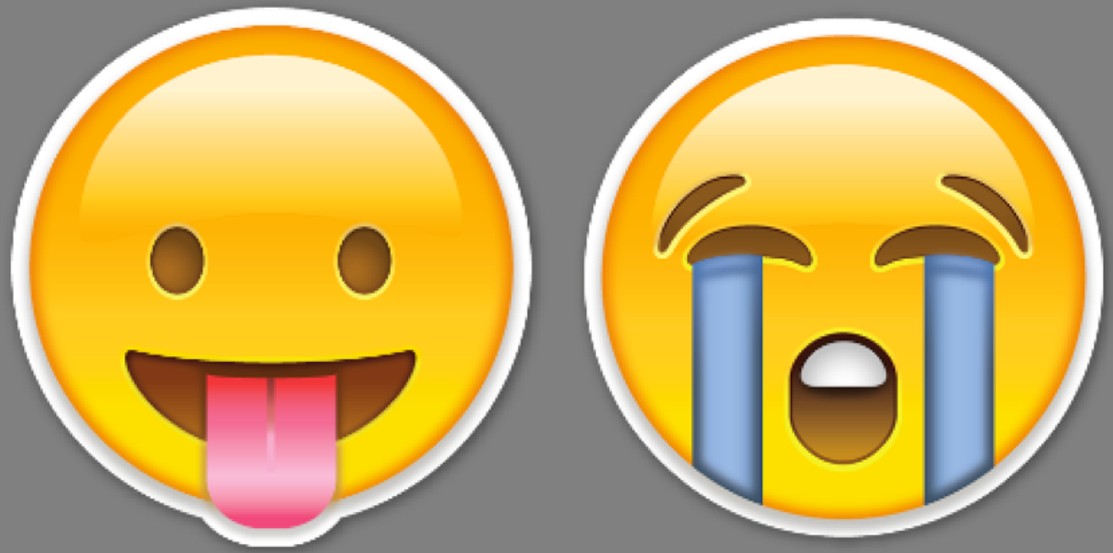
Draw calls again.

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

    glUniform1f(timeUniform, textureOffset);

    glBindBuffer(GL_ARRAY_BUFFER, vertPositionVBO);
    glVertexAttribPointer(positionAttribute, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(positionAttribute);

    glBindBuffer(GL_ARRAY_BUFFER, vertTexCoordVBO);
    glVertexAttribPointer(texCoordAttribute, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(texCoordAttribute);
    glBindTexture(GL_TEXTURE_2D, emojiTexture);

    glUniform2f(positionUniform, -0.5, 0.0);
    glDrawArrays(GL_TRIANGLES, 0, 6);

    glUniform2f(positionUniform, 0.5, 0.0);
    glDrawArrays(GL_TRIANGLES, 0, 6);

    glDisableVertexAttribArray(positionAttribute);
    glDisableVertexAttribArray(texCoordAttribute);

    glutSwapBuffers();
}
```


CS-6533

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

    glUniform1f(timeUniform, textureOffset);

    glBindBuffer(GL_ARRAY_BUFFER, vertPositionVBO);
    glVertexAttribPointer(positionAttribute, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(positionAttribute);

    glBindBuffer(GL_ARRAY_BUFFER, vertTexCoordVBO);
    glVertexAttribPointer(texCoordAttribute, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(texCoordAttribute);

    glBindTexture(GL_TEXTURE_2D, emojiTexture);
    glUniform2f(positionUniform, -0.5, 0.0);
    glDrawArrays(GL_TRIANGLES, 0, 6);

    glBindTexture(GL_TEXTURE_2D, cryingTexture);
    glUniform2f(positionUniform, 0.5, 0.0);
    glDrawArrays(GL_TRIANGLES, 0, 6);

    glDisableVertexAttribArray(positionAttribute);
    glDisableVertexAttribArray(texCoordAttribute);

    glutSwapBuffers();
}
```

CS-6533



Assignment 1

- Render a rectangle as a pair of triangles textured with an image of your choice in two different positions on screen.
- You must use the same buffers for both, but change the position using a shader uniform in the vertex shader.
- Bonus: Add a uniform to the fragment shader and change the color or texture coordinate of the final pixel based on input.