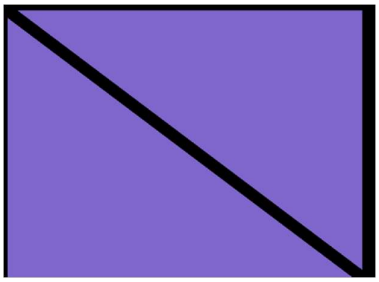
Using OpenGL:

One of my first OpenGL programs was to draw two triangles using GLFW library and a vertex/fragment shader. I followed the code from the OpengGL Programing Guide (9th Edition).



The output of the application

Then to further my knowledge I played around with basic colour of the triangles using the fragment shader.

```
#version 450 core

layout (location = 0) out vec4 fColor;

void main()
{
    fColor = vec4(0.5, 0.4, 0.8, 1.0);
}
```

Change this for RGBA

Next, I wanted to include some math's to see how you can translate, rotate and scale your rendered items. So, after some research you can manipulate the position data inside the vertex shader. Before assigning a value to the gl_Position variable. You can manipulate it causing either of these three changes.

To decide how much to rotate these variables I pass in two uniform variables called "TranslateX" and "TranslateY". Then I create a new vec4 in the vertex shader and add this to the gl_Position variable

```
#version 450 core
uniform Uniforms
{
    float xTranslate;
    float yTranslate;
};
layout (location = 0) in vec4 vPosition;

void main()
{
    pos.x = pos.x + xTranslate;
    pos.y = pos.y + yTranslate;

    gl_Position =  pos;
}
```

Next step is rotation. I knew the math to rotate around the Z axis is…

newX = x cos(b) – y sin(b)

newY = x sin(b) + y cos(b)

where b = angle to rotate by

I calculated what cosb and sinb would be inside the application and passed them in as Uniform values…

```
#version 450 core
uniform Uniforms
{
    float xTranslate;
    float yTranslate;
    float scale;
    float cosb;
    float sinb;
};
layout (location = 0) in vec4 vPosition;

vec4 rotation2D(vec4 _pos)
{
   vec4 position = vec4(0, 0, _pos.z, 1);

   // x' = x cos b - y sin b
   // y' = x sin b + y cos b
   position.x = _pos.x * cosb - _pos.y * sinb;
   position.y = _pos.x * sinb + _pos.y * cosb;

   return position;
}

void main()
{
   vec4 pos = rotation2D(vPosition);
  // pos.x = pos.x + xTranslate;
  // pos.y = pos.y + yTranslate;

   gl_Position =  pos;
}
```

The finale step was to turn this into matrices. I already understood matrix math and new you can turn expressions like the two above into a matrix…

cosb	-sinb	0
Sinb	cosb	0
0	0	0

x
y
z

Then I needed to add the ability to translate with this matrix…

cosb	-sinb	0	0
sinb	cosb	0	0
0	0	0	0
xT	yT	zT	1

Then the scale matrix…

xS	0	0	0
0	yS	0	0
0	0	zS	0
0	0	0	1

Using these two matrices I can do any transform, rotation, scale.

In code this looks like…

```
//trying to make column major
 GLfloat xformMatrix[] = {
      cosb, -sinb, 0.0, 0.0,
      sinb, cosb, 0.0, 0.0,
       0.0, 0.0, 1.0, 0.0,
       xT, yT, zT, 1.0
      };


 GLfloat scaleMatrix[] = {
       xS, 0.0, 0.0, 0.0,
       0.0, yS, 0.0, 0.0,
       0.0, 0.0, zS, 0.0,
       0.0, 0.0, 0.0, 1.0
 };
```

The vertex shader looks like this…

```
#version 450 core

uniform Uniforms
{
    mat4 u_xformMatrix;
    mat4 u_scaleMatrix;
};

layout (location = 0) in vec4 vPosition;

void main()
{
    gl_Position = u_xformMatrix * u_scaleMatrix * vPosition;
}
```

To produce something like this...