1. O que é a GLSL? Quais os dois tipos de shaders são obrigatórios no pipeline programável da versão atual que trabalhamos em aula e o que eles processam?

A GLSL é uma linguagem de programação usada no desenvolvimento de shaders dentro do pipeline da OpenGL.

Os dois tipos de shaders obrigatórios são:

Vertex shader: Processa cada vértice separadamente. É quem descreve onde cada posição de vértice estará na tela.

Fragment shader: Processa cada fragmento separadamente. Descreve o tratamento dos pixels entre os vértices.

2. O que são primitivas gráficas? Como fazemos o armazenamento dos vértices na OpenGL?

São os elementos mais básicos que podem ser criados na aplicação gráfica, como por exemplo: pontos que são conectados para formar polígonos, que são conectados para formar o objeto final. Na OpenGL, os vértices são armazenados pelo buffer (VBO), um array de dados que envia os dados dos vértices para a GPU, onde os dados são alocados na própria memória da GPU.

3. Explique o que é VBO, VAO e EBO, e como se relacionam (se achar mais fácil, pode fazer um gráfico representando a relação entre eles).

VBO: Buffer que guarda um array de dados que manda informação dos vértices para a GPU.

VAO: Faz a conexão dos atributos de um vértice. É quem gerencia posição, cores, normais... O VAO descreve a forma como esses atributos dos vértices são armazenados num VBO.

EBO: Buffer que permite armazenar índices que a OpenGL usa para decidir quais vértices desenhar.

4. Considerando o seguinte triângulo abaixo, formado pelos vértices P1, P2 e P3, respectivamente com as cores vermelho, verde e azul.

a. Descreva uma possível configuração dos buffers (VBO, VAO e EBO) para representá-lo.

```
//layout dos vértices
GLfloat vertices[] = {
    0.0,  0.6, 0.0, 1.0, 0.0, 0.0,
   -0.6, -0.5, 0.0, 0.0, 1.0, 0.0,
    0.6, -0.3, 0.0, 0.0, 0.0, 0.1,
};
```

```
//Buffer VBO
GLuint vVBO;
glGenBuffers(1, &vVBO);
glBindBuffer(GL_ARRAY_BUFFER, vVBO);
glBufferData(GL_ARRAY_BUFFER, 18 * sizeof(GLfloat), vertices, GL_STATIC_DRAW);
```

```
//Buffer VAO
GLuint VAO;
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);

glBindBuffer(GL_ARRAY_BUFFER, vVBO);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)(3 * sizeof(GLfloat)));
glEnableVertexAttribArray(1);
```

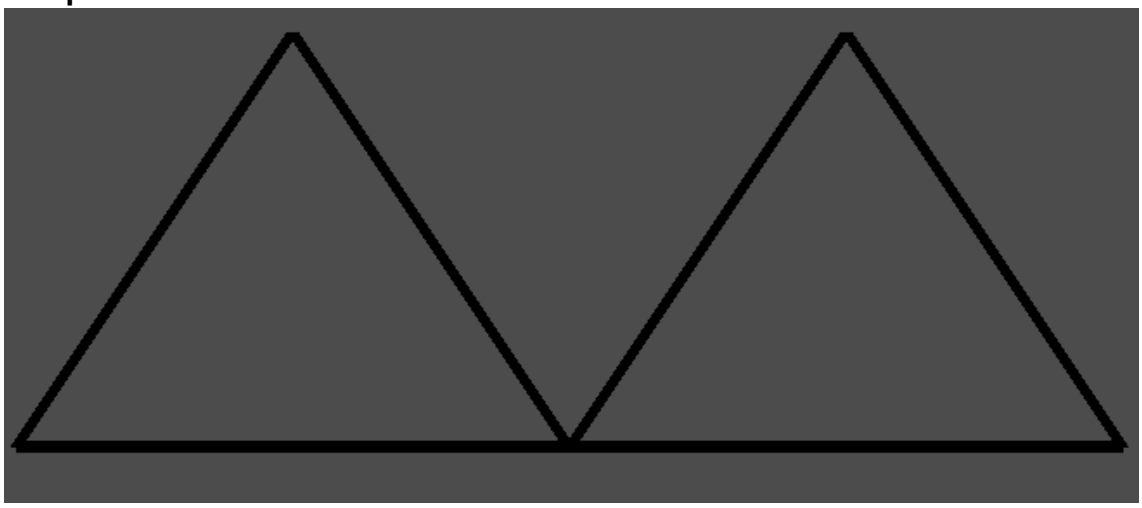
b. Como estes atributos seriam identificados no vertex shader?

```
// Código fonte do Vertex Shader (em GLSL): ainda hardcoded
const GLchar* vertexShaderSource = "#version 400\n"
"layout (location = 0) in vec3 position;\n"
"layout (location = 1) in vec3 color;\n"

"out vec4 vertexColor;\n"

"void main()\n"
"{\n"
"gl_Position = vec4(position, 1.0);\n"
"vertexColor = vec4(color, 1.0);\n"
"}\0";
```

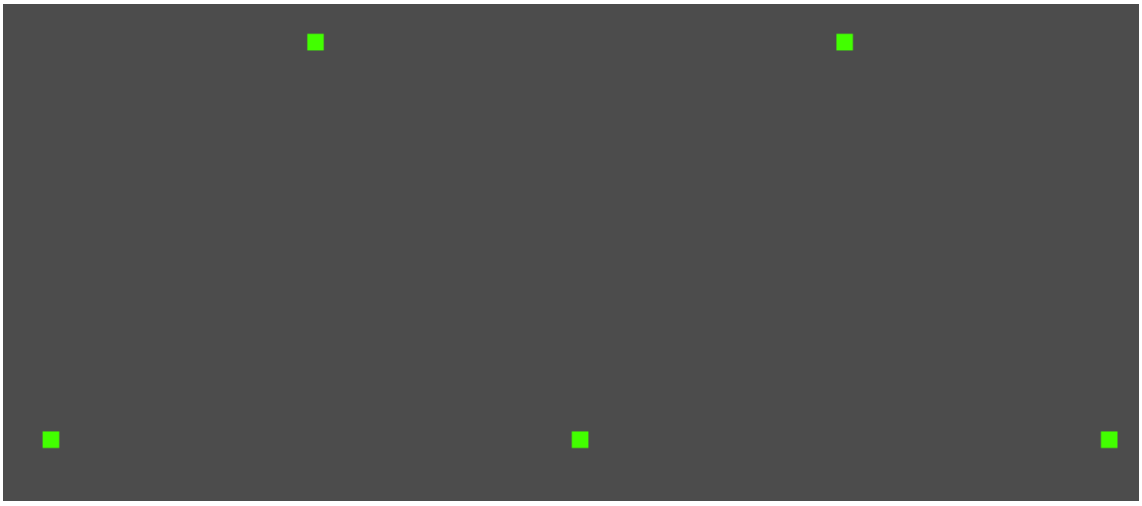
6. Faça o desenho de 2 triângulos na tela. Desenhe eles:

a. Apenas com o polígono preenchido.

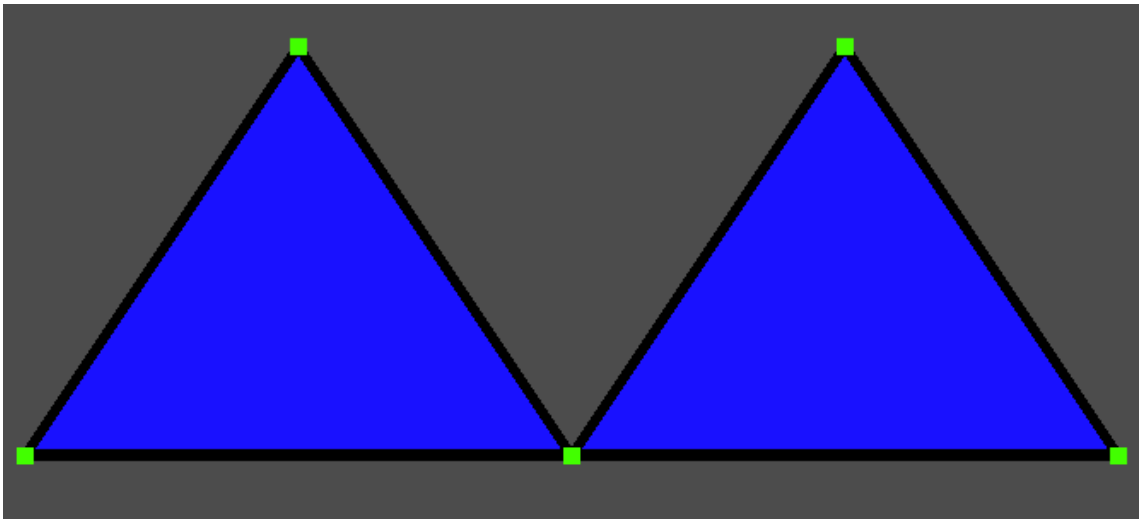
b. Apenas com o contorno.



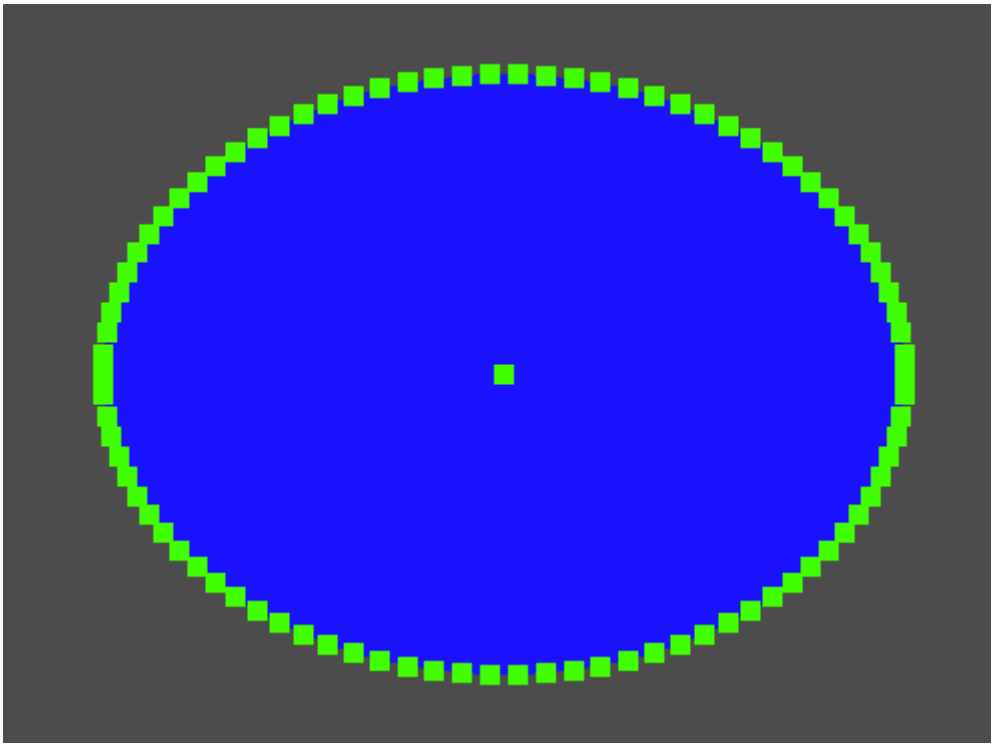
c. Apenas como pontos.



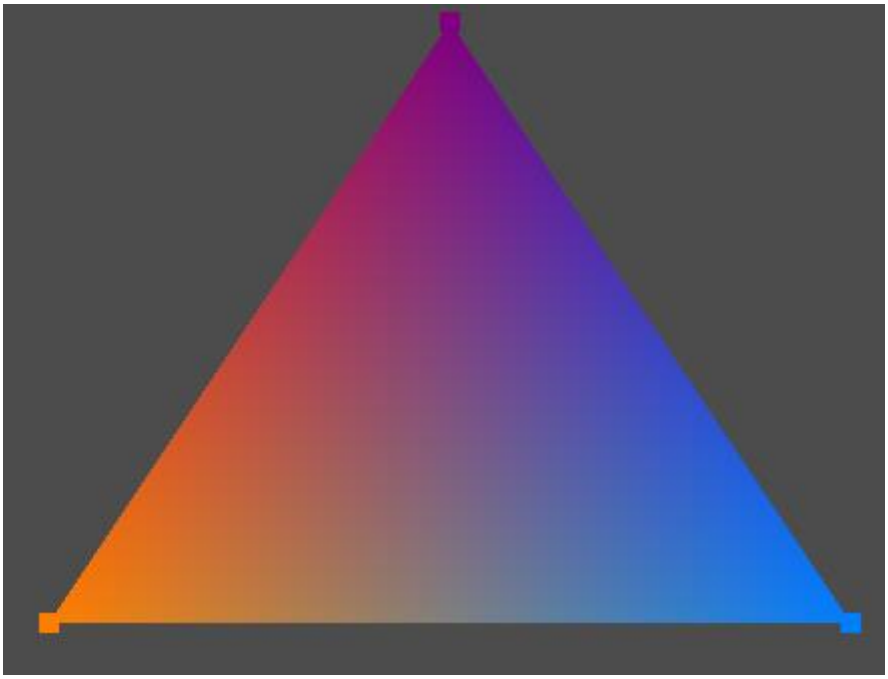
d. Com as 3 formas de desenho juntas.



7. Faça o desenho de um círculo na tela.



8. Faça o passo-a-passo para criar o triângulo com cores diferentes em cada vértice (prática do exercício 5).



9. Faça um desenho em um papel quadriculado (pode ser no computador mesmo) e reproduza-o utilizando primitivas em OpenGL. Neste exercício você poderá criar mais de um VAO e fazer mais de uma chamada de desenho para poder utilizar primitivas diferentes, se necessário.