Lista 1 - Igor Flores

Link do repositório

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?**

   → GLSL é a linguagem de programação que usamos para criarmos shaders, baseada em C. Os tipos de shaders obrigatórios no pipeline programável da versão atual são os Vertex Shaders e Fragment Shaders. O primeiro é responsável basicamente pelo processamento dos vértices de um polígono, onde esses pontos estaram na tela, seu tamanho, etc. Nessa etapa o nosso polígono ainda não é desenhado na tela, ele apenas lida com as informações do nosso polígono. O fragment shader é o que faz a figura acontecer, esta na etapa de rasterização. Nele é onde temos as informações de cores dos pixels, também como serão os pixels que estão entre os vértices do nosso polígono, definidos anteriormente no Vertex Shader.

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

   → São o que dão forma para nossas imagens. Temos pontos, polígonos e malhas. Ambos vem um do outro, respectivamente. O armazenamento é feito dentro do nosso VBO. No exemplo abaixo, definimos somente posições para 3 vértices, por exemplo. Essas informações vão ser organizadas posteriormente pelo VAO para que a placa gráfica desenha esses pontos nas posições adequadas.

   ```
   float vertices[] = {
       -0.5f, -0.5f, 0.0f,
        0.5f, -0.5f, 0.0f,
        0.0f,  0.5f, 0.0f
   };
   ```

   Como a OpenGL é um ambiente 3D, precisamos sempre passar a coordenada Z que queremos. Como estamos trabalhando em imagens 2D, sempre passamos o Z como 0.

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: Vertex Buffer Object → São como as informações dos vértices são estruturadas e armazenadas. É como definimos onde será a posição, a cor, o material, etc.

   → VAO: É responsável pela organização e disponibilidade das informações que estão no buffer. É o VAO que informa para a nossa placa gráfica qual a posição do nosso Buffer é referente as posições de cada ponto do nosso polígono, a cor que ele vai ter, o material que vai usar, etc.

   → EBO: São como índices para as informações dos vértices. Servem basicamente para reutilizarmos as informações dos vértices.

4. **Analise o código fonte do projeto Hello Triangle. Localize e relacione os conceitos de shaders, VBOs e VAO apresentados até então. Não precisa entregar nada neste exercício.**

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

   a. **Apenas com o polígono preenchido**

   b. **Apenas com contorno**

   c. **Apenas como pontos**

   d. **Com as 3 formas de desenho juntas**

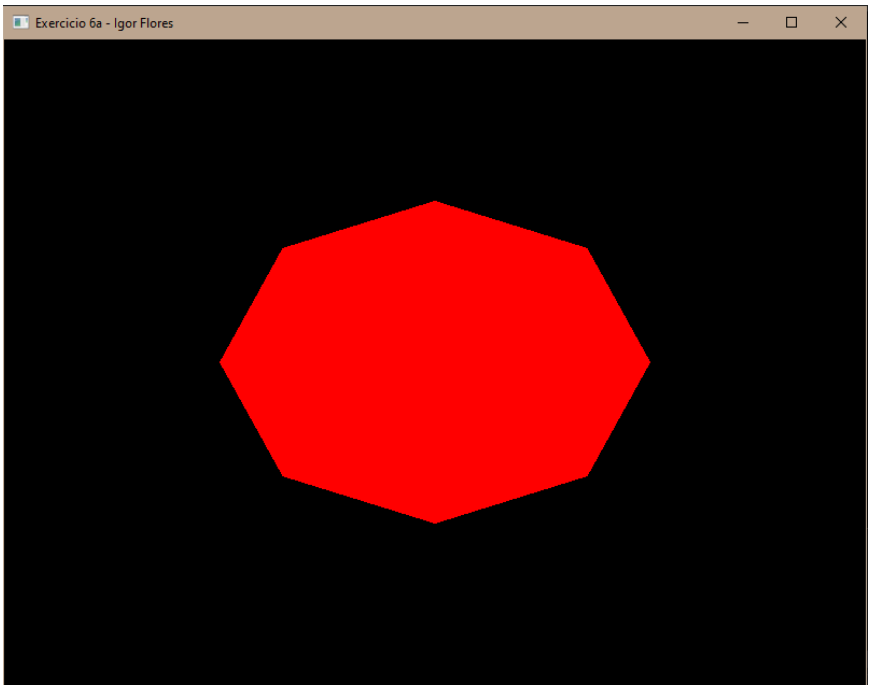
   e. **Atualize o shader para receber uma cor de contorno**

      → Exercício feito no repositório dentro da pasta Exercicio5. Código apresenta somente versão final, com todas as formas juntas.

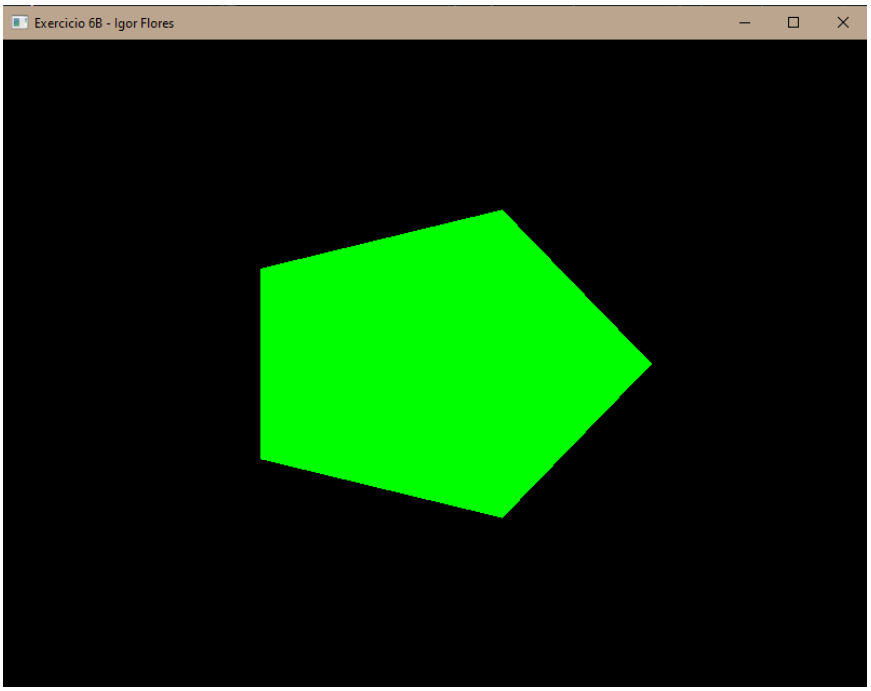
6. **Faça o desenho de um círculo na tela, utilizando a equação paramétrica do círculo para gerar os vértices. Depois disso:**

   → Exercício 06 apresenta desenho do círculo, para os exercício A, B, C e D, foram reduzidos os números de pontos para 8, 5, 10 (começando o primeiro ponto em 0.0) e 3, respectivamente.

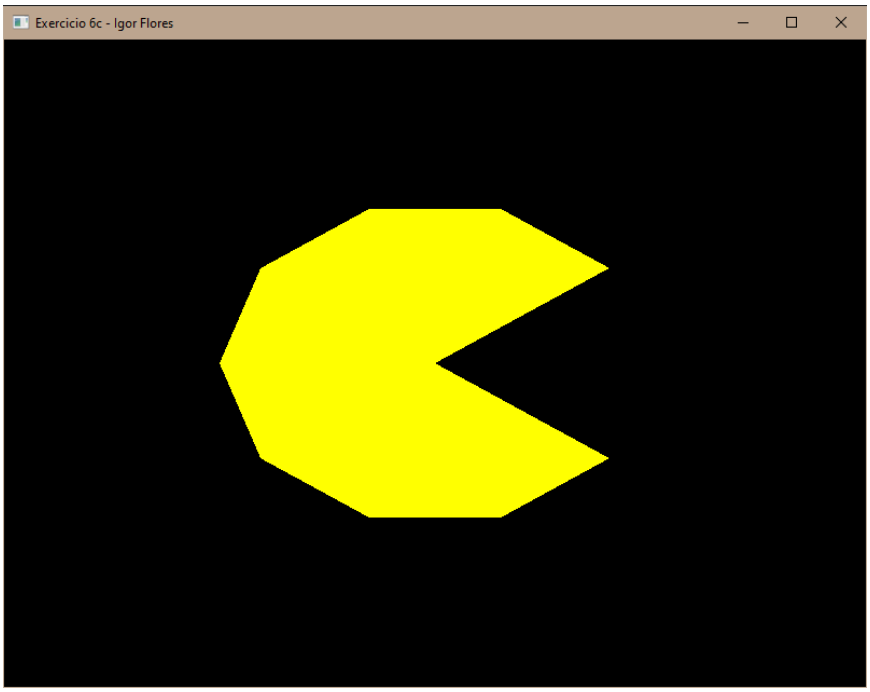
   1. Desenhe um octágono



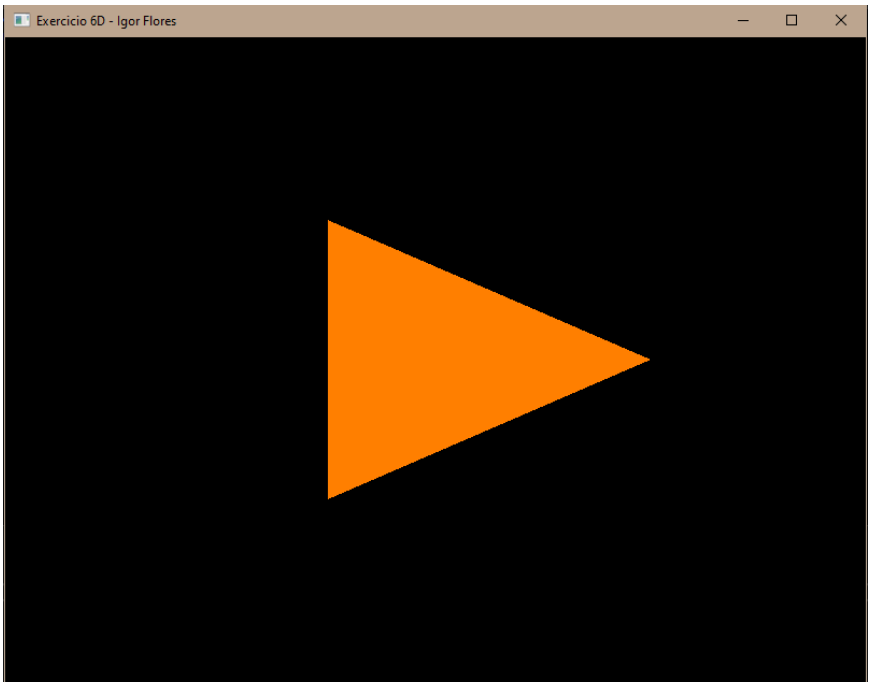
2. Desenhe um pentágono



3. Desenhe um pac-man!



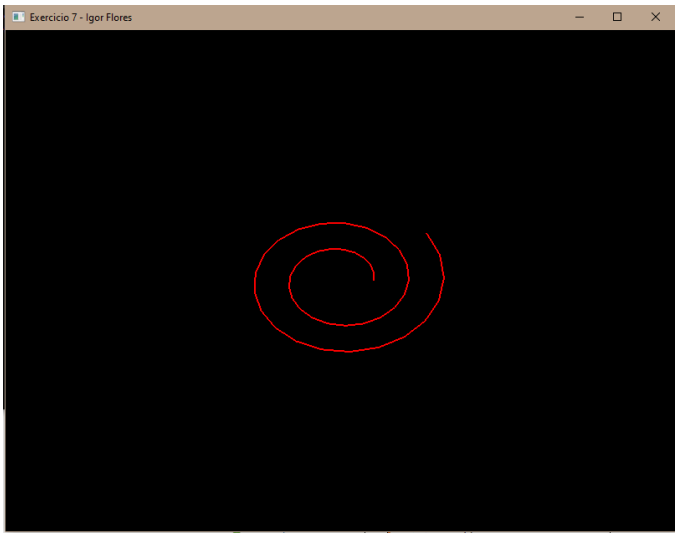
4. Desenhe uma fatia de pizza



5. **DESAFIO: desenhe uma “estrela”**

   Não consegui desenhar a estrela.

7. **Desenhe uma espiral**



→ Implementado dentro da Solução do exercício 6. Mudanças para desenhar o espiral feito estão comentadas nas linhas 96, 132, 155 e 162.

8. **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.**

```
GLfloat vertices[] = {
    // cord x, cord y, cord z, cor r, cor g, cor b
     0.0,  0.5, 0.0, 1.0, 0.0, 0.0, //v0
    -0.3, -0.3, 0.0, 0.0, 1.0, 0.0, //v1
     0.3, -0.15, 0.0, 0.0, 0.0, 1.0, //v2
};
```

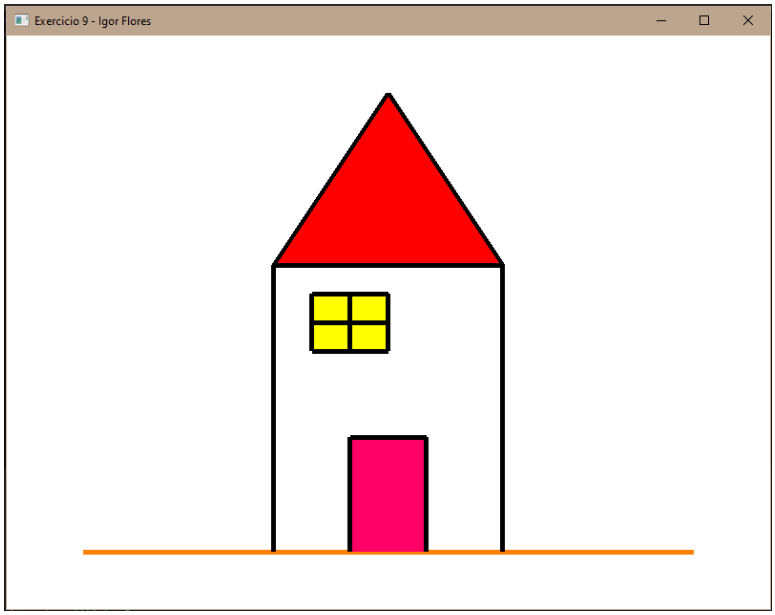
```
//Atributo 1 - Layout 0 Definido no ShaderVS
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);

//Atributo 2 - Layout 1 Definido no ShaderVS
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? Agora implemente!**

```
#version 400

layout (location = 0) in vec3 position;
layout (location = 1) in vec3 color;

out vec3 vertexColor;


void main()
{
    //...pode ter mais linhas de código aqui!
    gl_Position = vec4(position, 1.0);
    vertexColor = color;
}
```

→ Implementação na pasta de exercício 8.

9. **Faça um desenho em um papel quadriculado (pode ser no computador mesmo) e reproduza 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.**



→ Implementação na pasta exercício 9.

10. **Implemente (pode pegar do tutorial) uma classe para tratar os shaders a partir de arquivos.**

→ Implementado em todos os exercícios.