Luis Henrique Daltoe Dorr

Processamento Gráfico – Lista 1.

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?

GSL:

linguagem de programação de shaders usada para escrever programas que controlam o processamento gráfico em aplicativos que utilizam a API OpenGL.  
  
Tipos de Shaders:

* Vertex Shadder:

Processa cada vértice de um Objeto 3D.

**Função:** é transformar as coordenadas 3D em diferentes coordenadas 3D (falaremos mais sobre isso posteriormente) e o sombreador de vértice nos permite fazer algum processamento básico nos atributos do vértice.

* Fragment Shadder:

**Função:** calcular a cor final de um pixel e, geralmente, esse é o estágio em que ocorrem todos os efeitos avançados do OpenGL. Normalmente, o fragment shader contém dados sobre a cena 3D que podem ser usados para calcular a cor final do pixel.

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

Primitivas gráficas são formas geométricas básicas, onde esses são usados para gerar objetos mais complexos em uma cena 3D. Alguns exemplos:

* Pontos
* Arestas/linhas
* Triângulos
* Quadriláteros

Nós podemos usar buffers para realizar o armazenamento dos vértices. O mais apropriado para fazer o armazenamento de vértices seria o VBO – Vertex Buffer Object. Outro buffer importante seria o IBO - Index Buffer Object, mas ele serve mais para fazer a ordenação.

1. 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):

O VBO é uma buffer de dados que armazena informações sobre os vértices de objetos, como por exzemplo:

* + Corndenada
  + Cores
  + normais
* VAO (Vertex Array Object):  
  É uma espécie de Objeto que encapsula informações de VBOs e outros estados de renderização. A função principal é auxiliar na organização e na especificação de como os dados dos VBOs serão usados durante a renderização.
* EBO (Element Buffer Object):  
  Um EBO é um buffer que armazena índices que representam a ordem em que os vértices dos objetos 3D serão processados para formar primitivas gráficas, como triângulos.
* Relacionamento:  
  VAO - > VBO -> EBO

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

Uma imagem contendo Ícone

Descrição gerada automaticamente

b. Apenas com contorno   
Ícone

Descrição gerada automaticamente com confiança média

c. Apenas com pontos   
Imagem de vídeo game

Descrição gerada automaticamente com confiança média

d. Com as 3 formas de desenho juntas

Forma

Descrição gerada automaticamente

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

a) Desenhe um octágono

b) Desenhe um pentágono

c) Desenhe um pac-man!

d) Desenhe uma fatia de pizza

e) DESAFIO: desenhe uma “estrela”  
  
  
Círculo:

Forma, Círculo

Descrição gerada automaticamente  
  
Octogono:  
Fundo preto com estrelas

Descrição gerada automaticamente

Pentágono:  
Forma

Descrição gerada automaticamente com confiança média

Pac-man:  
Logotipo

Descrição gerada automaticamente

Pizza:  
Uma imagem contendo Ícone

Descrição gerada automaticamente

Estrela:  
Desenho de uma pessoa

Descrição gerada automaticamente com confiança baixa

7. Desenhe uma espiral, assim:   
Espiral:  
Forma

Descrição gerada automaticamente

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.

**VBO:**  
float vertices[] = {

0.0f, 6.0f, 1.0f, 0.0f, 0.0f, // P1: Posição (x,y) e Cor (r,g,b)

-6.0f,-6.0f, 0.0f, 1.0f, 0.0f, // P2

6.0f,-3.0f, 0.0f, 0.0f, 1.0f // P3

};

**VAO:**  
int setupGeometry()

{

GLfloat vertices[] = {

//x y z r g b

0.0f, 0.6f, 0.0f, 1.0f, 0.0f, 0.0f, // P1: Posição (x,y,z) e Cor (r,g,b)

-0.6f,-0.6f, 0.0f, 0.0f, 1.0f, 0.0f, // P2

0.6f,-0.3f, 0.0f, 0.0f, 0.0f, 1.0f // P3

};

GLuint VBO, VAO;

glGenBuffers(1, &VBO);

glBindBuffer(GL\_ARRAY\_BUFFER, VBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices), vertices, GL\_STATIC\_DRAW);

glGenVertexArrays(1, &VAO);

glBindVertexArray(VAO);

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);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glBindVertexArray(0);

return VAO;

}

**b.**  
unsigned int indices[] = {

0, 1, 2 // Os índices dos vértices P1, P2 e P3 no VBO

};  
  
  
Uma imagem contendo Forma

Descrição gerada automaticamente