UNISINOS

Atividade Acadêmica de Processamento Gráfico: Fundamentos

Aluna: Gabriela Bley Rodrigues
Professora: Rossana Baptista Queiroz

Exercícios 01:

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, sigla para OpenGL Shading Language, é a linguagem de programação usada para escrever shaders que rodam diretamente na GPU dentro do OpenGL. Essa linguagem substitui a rigidez do pipeline fixo, que permite que o programador controle como os dados gráficos serão transformados e renderizados. No pipeline programável que utilizamos, dois tipos de shaders são fundamentais: o vertex shader e o fragment shader. O vertex shader processa os vértices, aplicando transformações de posição e repassando informações adiante no pipeline, enquanto o fragment shader processa os fragmentos que podem se tornar pixels, definindo cores, texturas e efeitos visuais que aparecem na tela.

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

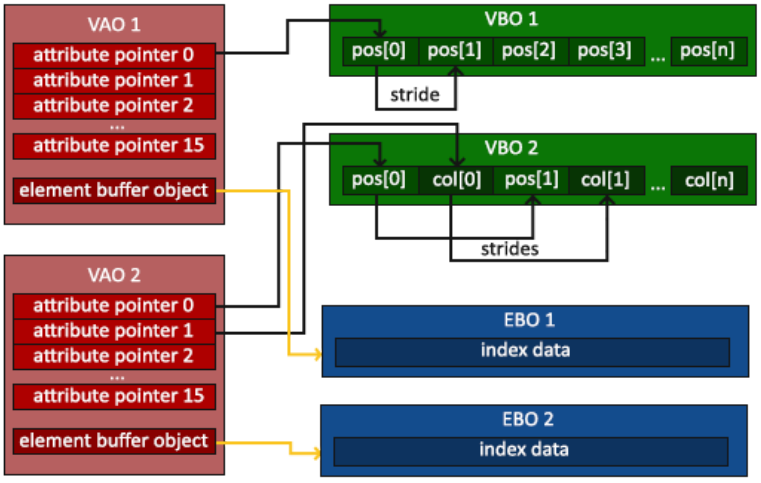
   As primitivas gráficas são os blocos de construção básicos para criar imagens, como pontos, linhas e triângulos, ou seja, são a forma mais simples que o hardware de vídeo consegue renderizar de forma eficiente. A OpenGL utiliza um método otimizado para armazenar os vértices desses objetos. Ao invés de enviar as informações de cada vértice várias vezes, a aplicação armazena todos os dados em um Buffer de Objeto de Vértice (VBO), que fica diretamente na memória da GPU. A CPU então apenas envia um comando para que a GPU use os dados do VBO para desenhar as primitivas, garantindo um processo de renderização mais rápido e de forma eficiente.

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).**

   O VBO é um buffer na memória da GPU que armazena os dados brutos dos vértices. Isso inclui as posições (coordenadas x, y, z), as cores, as coordenadas de textura e as normais. O VBO é o "conteúdo" de dados que descreve a geometria de um objeto. A principal vantagem de usá-lo é que ele permite que a CPU envie grandes quantidades de dados de vértice para a GPU de uma única vez, em vez de um por um, reduzindo drasticamente a sobrecarga de comunicação entre a CPU e a GPU, otimizando o desempenho.

Já o EBO é outro tipo de buffer da GPU que armazena os índices dos vértices. Em vez de duplicar os dados de um vértice que é compartilhado entre diferentes triângulos (como nos cantos de um cubo), se armazena o vértice uma única vez no VBO e usa o EBO para referenciar esse vértice várias vezes por meio de seu índice. O VAO atua como um "gerenciador de estado", armazenando as configurações de como a OpenGL deve interpretar os dados do VBO e do EBO. Para evitar ter que reconfigurar o pipeline de renderização toda vez que você for desenhar um objeto, a OpenGL permite que você simplesmente ative o VAO dele. O VAO "se lembra" de todas as chamadas de configuração que foram feitas enquanto ele estava ativo. Então:

- VBO e EBO contêm os dados reais (vértices e índices). Eles são os pacotes de dados.
- O VAO age como um container que agrupa as configurações para os dados do VBO e do EBO. Ele é o "manual de instruções".



(Fonte: LearnOpenGL - Hello Triangle)

A imagem acima demonstra como um VAO atua como um container para as configurações que definem como os dados em um ou mais VBOs devem ser lidos. No lado esquerdo, temos dois VAOs, rotulados como VAO 1 e VAO 2. Cada um deles possui 16 ponteiros de atributo. No lado direito, estão dois VBOs, chamados VBO 1 e VBO 2. Eles

contêm os dados reais dos vértices. VBO 1 armazena apenas dados de posição (posições pos[0] a pos[n]). O VAO 1 tem um de seus ponteiros de atributo apontando para este VBO, indicando que a posição do vértice pode ser encontrada ali. A seta aponta para o conceito de stride, que é a distância em bytes entre o início de um dado de vértice e o próximo dado do mesmo tipo. VBO 2 armazena dados de posição e cor (pos[0], col[0], pos[1], col[1], etc.). Note que os dados de posição e cor estão intercalados. O VAO 2 tem dois ponteiros de atributo apontando para este VBO. Um aponta para os dados de posição e o outro, para os de cor. A seta indica que o VAO 2 precisa de dois strides para pular de uma posição para a próxima e de uma cor para a próxima.

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.**

   O projeto "Hello Triangle" exemplifica a interação entre shaders, VBOs e VAOs. O código define os shaders, e em seguida, ele cria um VBO para armazenar os dados do triângulo na memória da placa de vídeo. No final o VAO é usado como um gerenciador de estado, que "lembra" as configurações de como os dados no VBO devem ser interpretados pelos shaders. No loop de renderização, o código simplesmente ativa o VAO e o programa de shader, permitindo que a GPU desenhe o triângulo, pois todas as configurações necessárias já estão prontas e armazenadas na memória de vídeo.