Intro OpenGL

Slides originais por Caio Hideo.

OpenGL

- Uma especificação de uma API, a API em si é implementada pelos fabricantes de hardware (intel, nvidia, amd, ...) ou do sistema operacional (apple)
- API é definida por um grupo de fabricantes (Khronos group) que decide qual o mínimo de requisitos para uma versão da API
- Planejada para ser extendida, assim fabricantes podem apresentar funcionalidades novas ou exclusivas sem modificar API base

OpenGL antigo vs moderno

Nas primeiras versões do OpenGL o hardware disponível era mais simples. O hardware era usado para acelerar partes do processo de rasterização, somente era possível alterar alguns parâmetros do processo [**pipeline fixo**]

Com a evolução das placas gráficas mais partes do processo de rasterização foram transferidas para o hardware. Também foi criada a possibilidade de programar o hardware para executar novos algoritmos [**pipeline programável**]

Essa mudança na API ocorreu durante a transição para a versão 3.x, dividindo a API em OpenGL antigo (1.x) e moderno (> 3.2)

Programando OpenGL

Para usar o OpenGL é necessário carregar a API no começo da execução do programa. O processo é diferente de simplesmente *linkar* uma biblioteca durante a compilação do programa. Isso é necessário porque API usada para rodar o programa está na máquina do usuário final, não do desenvolvedor

Primeiro programa - Contexto

O 1º passo para usar a OpenGL é criar um contexto. O contexto é a união de todas as informações que a OpenGL administra (configuração de partes da GPU, buffers de memória, *shaders*, etc)

Embora o contexto seja fundamental para OpenGL, a sua criação depende do sistema de janelas (Windows, XOrg, Android). Assim o processo para criar o contexto depende da plataforma.

Bibliotecas como GLFW e freeGLUT escondem os detalhes de cada plataforma para criação do contexto. **Utilizaremos GLFW.**

GLFW

http://www.glfw.org/

Uma biblioteca para criação de janelas e processamento de entrada, o objetivo é facilitar o desenvolvimento de programas OpenGL.

Primeiro programa - carregando funções

Como a API é *linkada* durante a execução do programa é preciso encontrar os endereços das funções. Assim como a criação do contexto esse processo depende da plataforma

Para essa tarefas existem várias bibliotecas disponíveis para esconder os detalhes (GLEW, glad, GL3W). **Utilizaremos glad.**

Primeiro programa - Visão Geral

Um resumo do programa

1. Criar, compilar e *linkar* os *shaders*
2. Criar e inicializar o buffer com os vértices
3. Criar os *bindings* entre o buffer e as variáveis de entrada dos *shaders*
4. No loop do seu programa:
    a. Ativar os *shaders* e *bindings*
    b. Mandar o comando para desenhar
    c. Desativar os *shaders* e *bindings*

Primeiro programa – Shaders

Shaders são pequenos programas que são executados em GPUs

Cada *shader* tem um tipo {***vertex***, ***fragment***, *geometry, tessellation control, tessellation evaluation*} que define entradas, saídas e função a ser implementada

Shaders de diferentes tipos são combinados para formar um programa, esse programa é um pipeline que descreve como vértices são transformados em uma imagem

Para um programa simples somente é necessário um *vertex* e um *fragment shader*

Primeiro programa - Shaders

Para criar um programa:

1. Criar um *shader object* via glCreateShader()
2. Carregar o código do *shader* via glShaderSource()
3. Compilar o *shader* via glCompileShader()
4. Criar um *program object* via glCreateProgram()
5. Ligar cada *shader* ao programa via glAttachShader()
6. *Linkar* o programa via glLinkProgram()

Primeiro programa - Buffers

A OpenGL assume a responsabilidade de administrar a memória usada pela GPU

Todos os dados passados para a OpenGL precisam ser copiados da memória do programa para buffers criados e alocados pela OpenGL. O principal tipo de buffer usado é o ***Vertex Buffer Object*** (**VBO**)

Um VBO pode guardar qualquer atributo de vértices, como cores, coordenadas de texturas, normais, etc. Isso depende da entrada do *shader* definido pelo programador

Primeiro programa - Buffers

Assim como muitos objetos na API os passos para criar um VBO são:

1. Criar um VBO via glGenBuffers(), essa função não aloca memória, somente reserva um “nome”, na verdade um índice | handle, que pode ser usado para se referir ao buffer

2. Alocar e inicializar o VBO via glBufferData(), essa função pode ser usada somente para alocar memória também, e chamadas a glBufferSubData() podem ser usadas para modificar o conteúdo da memória (sem alterar o tamanho do buffer)

Primeiro programa - Bindings

Após definir o programa a ser executado e o buffer de entrada é preciso definir como esses dados serão interpretados pelo programa. Esse processo de *binding* define um mapeamento de cada variável de entrada do *shader* para um componente do vértice

Na OpenGL as variáveis de entrada são especificadas por slots (índices), e dado o nome de uma variável é possível obter o seu índice no programa

O *binding* de uma variável é a associação de um slot com um atributo de um vértice. O atributo é dado como uma tripla (tamanho, *offset*, *stride*) em relação ao buffer

Primeiro programa - Bindings

Para auxiliar no processo de *binding* OpenGL define ***Vertex Array Object* (VAO)**

Um VAO guarda uma referência a um VBO e *bindings* entre o VBO e slots de um programa, de forma que esse processo de *binding* não tenha que ser feito toda vez antes de desenhar um buffer

Primeiro programa - Desenhando

O último passo é chamar a função que desenha o buffer, no programa exemplo é a função glDrawElements(). Essa função define como os vértices do buffer são agrupados (**pontos, linhas, triângulos**, ...) e qual o tamanho do buffer usado

Links úteis

http://docs.gl/ - documentação de funções da OpenGL e GLSL, divididas por versão das APIs