UFABC

Computação Gráfica

André Brandão

Aula 03

Revisão de

Álgebra Linear

UFABC

Sumário

- Parte 1
  - Determinantes
  - Matrizes
- Parte 2
  - Extensões Opengl
  - Vertex Buffer Objects

Álgebra Linear

Os próximos slides foram baseados no material de autoria do

Professor Ravi Ramamoorthi

Matrizes

- Podem ser usadas para transformar pontos (vetores)
  - Translação, rotação, corte, escala

O que é uma matriz

- Lista de números (m×n = m linhas, n colunas)

$$\begin{pmatrix} 1 & 3 \\ 5 & 2 \\ 0 & 4 \end{pmatrix}$$

- Adição, multiplicação por um escalar simples: elemento por elemento.

Multiplicação matriz-matriz

- Numero de colunas da primeira matriz deve ser igual ao número de linhas da segunda

$$\begin{pmatrix} 1 & 3 \\ 5 & 2 \\ 0 & 4 \end{pmatrix} \begin{pmatrix} 3 & 6 & 9 & 4 \\ 2 & 7 & 8 & 3 \end{pmatrix}$$

- Elemento (i,j), em produtos, é o produto escalar da linha i da primeira matriz e coluna j da secunda matriz.

Multiplicação matriz-matriz

- Numero de colunas da primeira matriz deve ser igual ao número de linhas da segunda

$$\begin{pmatrix} 1 & 3 \\ 5 & 2 \\ 0 & 4 \end{pmatrix}\begin{pmatrix} 3 & 6 & 9 & 4 \\ 2 & 7 & 8 & 3 \end{pmatrix} = \begin{pmatrix} 9 & 27 & 33 & 13 \\ 19 & 44 & 61 & 26 \\ 8 & 28 & 32 & 12 \end{pmatrix}$$

- Elemento (i,j), em produtos, é o produto escalar da linha i da primeira matriz e coluna j da secunda matriz.

Multiplicação matriz-matriz

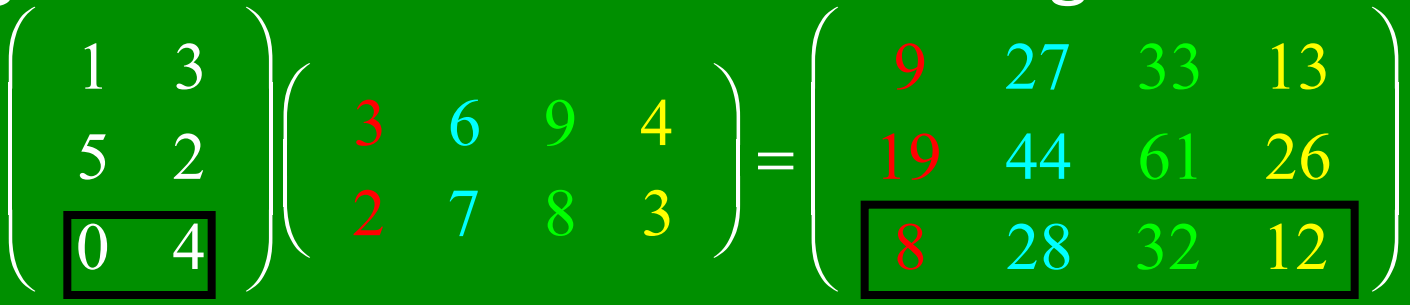
- Numero de colunas da primeira matriz deve ser igual ao número de linhas da segunda

$$\begin{pmatrix} 1 & 3 \\ 5 & 2 \\ 0 & 4 \end{pmatrix}\begin{pmatrix} 3 & 6 & 9 & 4 \\ 2 & 7 & 8 & 3 \end{pmatrix} = \begin{pmatrix} 9 & 27 & 33 & 13 \\ 19 & 44 & 61 & 26 \\ 8 & 28 & 32 & 12 \end{pmatrix}$$

- Elemento (i,j), em produtos, é o produto escalar da linha i da primeira matriz e coluna j da secunda matriz.

Multiplicação matriz-matriz

- Numero de colunas da primeira matriz deve ser igual ao número de linhas da segunda



- Elemento (i,j), em produtos, é o produto escalar da linha i da primeira matriz e coluna j da secunda matriz.

Multiplicação matriz-matriz

- Numero de colunas da primeira matriz deve ser igual ao número de linhas da segunda

$$\begin{pmatrix} 3 & 6 & 9 & 4 \\ 2 & 7 & 8 & 3 \end{pmatrix}\begin{pmatrix} 1 & 3 \\ 5 & 2 \\ 0 & 4 \end{pmatrix}$$ NOT EVEN LEGAL!!

- Não-comutativa (AB e BA são diferentes)
- Associativa e distributiva
  - A(B+C) = AB + AC
  - (A+B)C = AC + BC

Multiplicação matriz-vetor

- Importante para transformações de pontos
- Trata o vetor como uma matriz-coluna (m×1)

- Exemplo: reflexão 2D no eixo y

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -x \\ y \end{pmatrix}$$

Matriz transposta

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}^T = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

$$(AB)^T = B^T A^T$$

Matriz identidade e inversa

$$I_{3\times 3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$AA^{-1} = A^{-1}A = I$$

$$(AB)^{-1} = B^{-1}A^{-1}$$

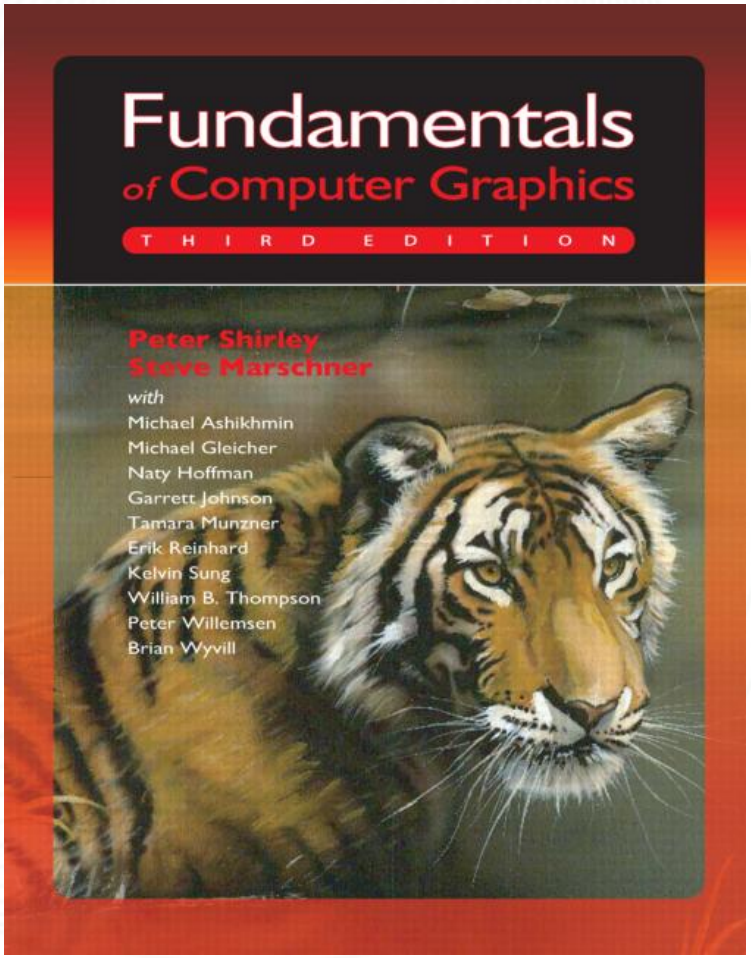
Sumário

- ~~**Parte 1**~~
  - ~~**Determinantes**~~
  - ~~**Matrizes**~~
- Parte 2
  - Extensões Opengl
  - Vertex Buffer Objects

Aula de hoje

Shirley, Peter, Michael Ashikhmin, and Steve Marschner. Fundamentals of computer graphics. CRC Press, 3rd Edition, 2009.

- **Capítulo 5**



Parte 02

Slides de autoria do
Professor André Balan

Conteúdo

- Extensões Opengl
- Vertex Buffer Objects

Extensões OpenGL

- A partir da versão 1.1 do OpenGL, todas as novas funções passaram a não ser mais incluídas na API **gl.h**

- Motivações:
  - OpenGL em qualquer linguagem, não só em C/C++
  - Tornar mais dinâmica a evolução do OpenGL permitindo que as várias produtoras de GPUs contribuíssem sem burocracia com novas funcionalidades: **extensões**
  - Permitir que as aplicações descobrissem, em tempo de execução, quais as funções OpenGL estavam disponíveis no computador hospedeiro (host).

Extensões OpenGL

- Consequentemente, **uma aplicação moderna em OpenGL** precisa ***vasculhar os Drivers da GPU*** e ***ligar-se, em tempo de execução***, às funções disponíveis que ela precisa.
  Como? :: ***ponteiros para funções***

- Caso a GPU em questão não forneça as funções que a aplicação necessita, a aplicação pode fechar sem travar, informando o usuário o problema

  - Isso é importante, por exemplo, em linguagens que não tenham mecanismos de exceção, como a linguagem C.

Extensões OpenGL

- Para fazer um programa em OpenGL usando funcionalidades modernas é preciso:
  - A API 1.1 (gl.h) e sua biblioteca de implementações pré-compiladas (libopengl.h)
  - Programar sua aplicação para buscar, em tempo de execução, nos drivers da GPU, todas as funções modernas do OpenGL que ela precisar, usando ponteiros de funções para se referir a elas. Como?

1. Usando uma função que busca e retorna **ponteiros para funções**

ou

2. Usando bibliotecas específicas para isso (**GLEW**)

Extensões OpenGL

1. Usando uma das funções que retornam **ponteiros para funções.** A biblioteca GLFW tem uma função que faz isso:

   glfwGetProcAddress("*nome da função OpenGL*")

- Desvantagem:
  - Muitas linhas de código
  - Exemplo para uma única função OpenGL: **glGenBuffers**

```
typedef void (* GLGENBUFFERSP) (GLsizei, GLuint*);  // Tipo ponteiro p/ função
GLGENBUFFERSP glGenBuffers = (GLGENBUFFERSP) glfwGetProcAddress("glGenBuffes");
```

Type Casting

Nome da função OpenGL buscada

Extensões OpenGL

2. Usando bibliotecas específicas: GLEW, GLee, GL3w

- Muito prático

```
#include <GL/glew.h>
   .
   .
   .
  glewInit();
   .
   .
//pronto,  todas as funções OpenGL suportadas pela GPU
  estão prontas para serem usadas
```

Extensões OpenGL

- Neste curso vamos utilizar GLFW + GLEW
- Siga o Tutorial 03 para configuração do projeto com GLEW

Vertex Buffer Objects

Vertex Buffer Object

- Motivação
  - Substituir as funções de definição geométrica da versão 1.1
    - glBegin, glEnd, glVertex, glColour, glNormal, etc...
    - Invocar estas funções milhões e milhões de vezes exigia muito esforço da CPU e muito tráfego de dados entre a memória RAM e a GPU, tornando a aplicação extremamente ineficiente

- Os Buffer Objects permitem o armazenamento de dados geométricos na própria GPU:
  - Vértices, Cores, Normais, Texturas, Programação dos Shaders
  - Muitos dados são enviados para a GPU uma única vez, com poucas chamadas de funções OpenGL

Vertex Buffer Object

- Funções para o VBO:
  - **glGenBuffers** - reserva um identificador (um nº inteiro) para uma região de memória da GPU (Buffer Object). Pode reservar vários em uma única chamada
  - **glBindBuffers** – define o tipo de um buffer a partir do identificador
  - **glBufferData** – aloca memória para um buffer na GPU e transfere dados da memória RAM para esse buffer, a partir do identificador

Vertex Buffer Object

- Funções para o VBO:

```
void glGenBuffers(GLsizei n, GLuint
*buffers);
```

- *Define n identificadores distintos para buffers na GPU, colocando-os no vetor buffers.*

Vertex Buffer Object

- Funções para o VBO:

```
void glBindBuffer(GLenum target, GLuint buffer);
```

- *Define o tipo (target) de um determinado buffer a partir do identificar inteiro (buffer)*

- *Opções:*
  - *GL_ARRAY_BUFFER: buffer de vértices e seus atributos*
  - *GL_ELEMENT_ARRAY_BUFFER: buffer de índices para definição de polígonos*

Vertex Buffer Object

- Funções para o VBO:

```
void glBufferData(GLenum target, GLsizeiptr size,
                  const GLvoid *data, GLenum usage);
```

- *Aloca memória e transfere dados para a GPU*

Vertex Buffer Object

- Exemplo

```
float v[] = {
          +0.0, +0.5,
          -0.5, -0.5,
          +0.5, -0.5
    };

GLuint VBO1;
glGenBuffers(1, &VBO1);
glBindBuffer(GL_ARRAY_BUFFER, VBO1);
glBufferData(GL_ARRAY_BUFFER, sizeof(v), v, GL_STATIC_DRAW);
```

- Isso deve ser feito uma única vez no programa, diferentemente das funções antigas, glVertex, que precisavam ser executadas toda vez que a primitiva fosse desenhada.

Vertex Buffer Object

- Precisamos definir e habilitar o ponteiro de vértices

```
glVertexPointer(2, GL_FLOAT, 0, nullptr);
glEnableClientState(GL_VERTEX_ARRAY);
```

- Isso também deve ser feito uma única vez no programa

Vertex Buffer Object

- Finalmente, invocamos a função para desenhar a primitiva (neste caso, um triângulo)

```
while (...) {
 glClearColor(0.0, 0.0, 0.0, 1.0);
 glClear(GL_COLOR_BUFFER_BIT);

 /* Código de renderização OpenGL vai aqui */
 glDrawArrays(GL_TRIANGLES, 0, 3);

 /* Troca o buffer de fundo com o buffer de exibição */
 glfwSwapBuffers(window);
}
```

Fim da Aula 03

André Luiz Brandão