

Computação Gráfica e Interfaces

2017-2018
Fernando Birra

Transformações Geométricas em WebGL

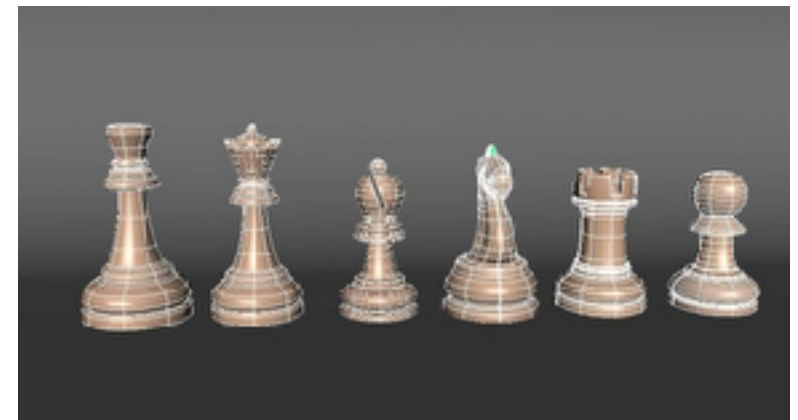
2017-2018
Fernando Birra

Objetivos

- Aprender a efetuar transformações geométricas em WebGL
 - Rotação
 - Translação
 - Mudança de escala
- Funções e conceitos relevantes oferecidos na biblioteca MV.js

Como modelar uma cena?

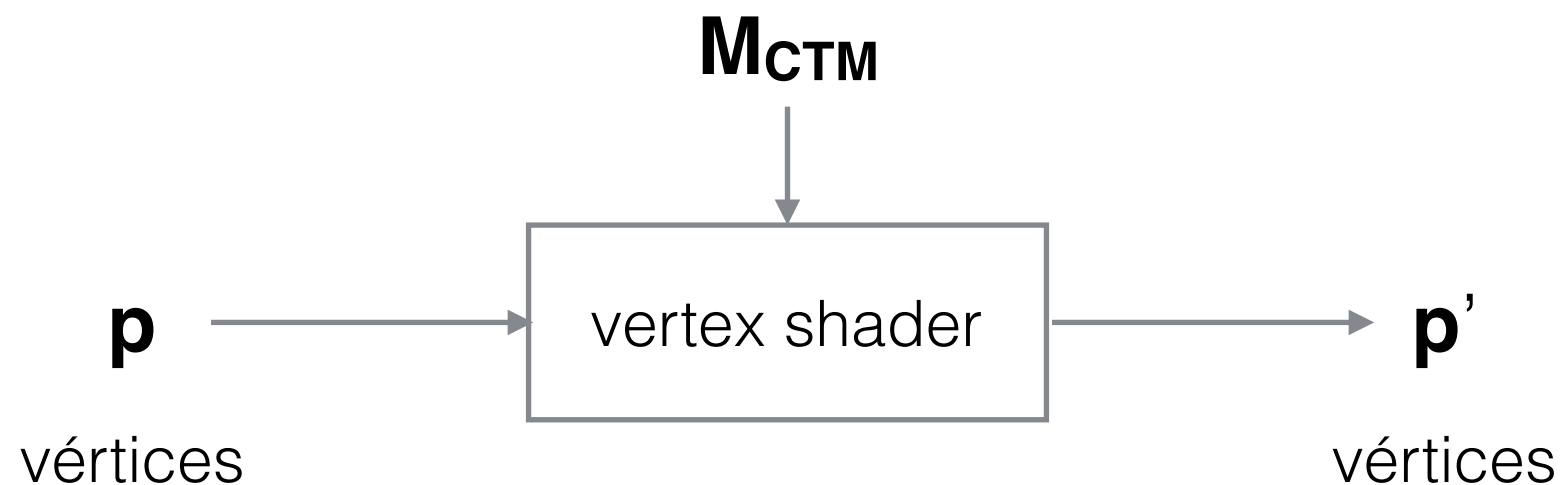
- Tipicamente, um programador duma aplicação gráfica 3D tem à sua disposição um conjunto de objetos ditos primitivos, os quais vai manipular para compor a sua cena.
- Cada (instância de um) objeto terá o seu tamanho ajustado, será orientado e posicionado na cena
- Em WebGL, usamos o vertex shader para transformar os vértices dos objetos



images retrieved from <http://tf3dm.com/3d-model/chess-pieces-83083.html>

CTM-Current Transformation Matrix

- Dum ponto de vista conceptual, podemos pensar que todos os vértices que passam pelo pipeline são transformados por uma matriz de transformação corrente (M_{CTM}), a qual pode ir sendo manipulada.
- O valor desta matriz é determinado pela aplicação e colocado à disposição do programa GLSL (no vertex shader).



Operações sobre a M_{CTM}

- A matriz de transformação corrente pode ser alterada por afetação direta dum novo valor:

Afetar com a matriz identidade: $M_{CTM} \leftarrow I$

Afetar com uma dada matriz M : $M_{CTM} \leftarrow M$

Afetar com uma matriz de Translação: $M_{CTM} \leftarrow T$

Afetar com uma matriz de Rotação: $M_{CTM} \leftarrow R$

Afetar com uma matriz de Escala: $M_{CTM} \leftarrow S$

- ou por concatenação (exemplo com pós-multiplicação):

Pós-multiplicação com uma matriz dada: $M_{CTM} \leftarrow M_{CTM} \cdot M$

Pós-multiplicação com uma matriz de translação: $M_{CTM} \leftarrow M_{CTM} \cdot T$

Pós-multiplicação com uma matriz de rotação: $M_{CTM} \leftarrow M_{CTM} \cdot R$

Pós-multiplicação com uma matriz de mudança de escala: $M_{CTM} \leftarrow M_{CTM} \cdot S$

Rotação em torno dum eixo arbitrário (desviado da origem)

- Começar com a matriz identidade: $\mathbf{M}_{CTM} \leftarrow \mathbf{I}$
- Trazer um ponto \mathbf{p}_f do eixo para a origem: $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{T}(-\mathbf{p}_f)$
- Efetuar a rotação: $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{R}(\theta)$
- Mover o ponto \mathbf{p}_f de volta: $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{T}(\mathbf{p}_f)$
- Resultado Acumulado: $\mathbf{T}(-\mathbf{p}_f) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(\mathbf{p}_f)$

Ordem errada!
(com pós-
concatenação)

Não esquecer que os pontos são multiplicados do lado direito: $\mathbf{p}' = \mathbf{M}_{CTM} \cdot \mathbf{p}$

Rotação em torno dum eixo arbitrário (desviado da origem)

- É necessário inverter a ordem das transformações!

1. $\mathbf{M}_{CTM} \leftarrow \mathbf{I}$

2. $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{T}(\mathbf{p}_f)$

3. $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{R}(\theta)$

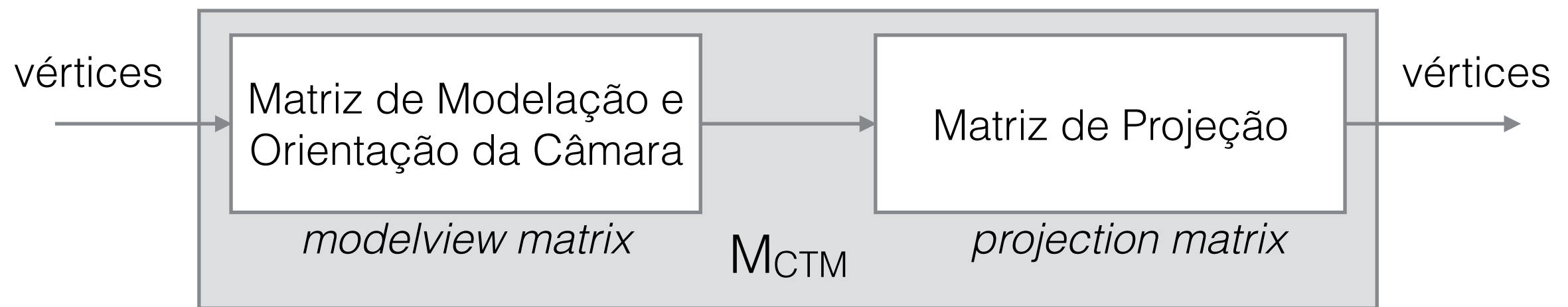
4. $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{T}(-\mathbf{p}_f)$

- Resultado Acumulado: $\mathbf{T}(\mathbf{p}_f) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(-\mathbf{p}_f)$



M_{CTM} em WebGL

- As versões de OpenGL antigas expunham a matriz M_{CTM} sob a forma do produto de duas matrizes com finalidades diferentes:



- É do nosso interesse emular este processo, mantendo uma matriz dedicada à projeção, separada da matriz usada para instanciar os objetos primitivos e orientar a câmara.

Modelview+Projection

- A matriz ModelView (M_{MV}) é usada para:
 - posicionar a câmara e orientá-la em relação ao restante da cena
 - Aplicar transformações de modelação/instanciação aos objetos primitivos
 - Pode ser criada por composição de rotações e translações mas a biblioteca MV.js disponibiliza a função `lookAt()` para este efeito.
- A matriz de projeção (M_{PROJ}) serve para definir o volume de visão (escolha da lente)
- Embora estas matrizes já não façam parte do estado do sistema, é uma boa estratégia usá-las na nossa aplicação:

$$\mathbf{p}' = \mathbf{M}_{PROJ} \cdot \mathbf{M}_{MV} \cdot \mathbf{p}$$

Suporte em MV.js

- Criar uma matriz identidade:

```
var m = mat4();
```

- Criar uma matriz de rotação:

```
r = rotate(angle, vx, vy, vz);  
r = rotateX(angle);  
r = rotateY(angle);  
r = rotateZ(angle);
```

- Criar uma matriz de mudança de escala:

```
m = scalem(sx, sy, sz);
```

- Criar uma matriz de translação:

```
m = translate(dx, dy, dz);
```

- Efetuar a pós-multiplicação:

```
m = mult(m, m1);
```

Exemplo

- Rodar 30° em torno dum eixo paralelo a Z, que passa pelo ponto (1,2,3)

```
var m = mult(translate(1,2,3),  
             mult(rotateZ(30), translate(-1,-2,-3)));
```

- Aumentar a dimensão x para o dobro e fazer um deslocamento de (2,0,3):

```
var m = mult(translate(2,0,3), scalem(2,1,1));
```

Transformações (Matrizes) Arbitrárias

- É possível enviar para o programa GLSL uma matriz 4x4 arbitrária, cujo conteúdo foi previamente definido pela aplicação.
- As matrizes são guardadas como arrays unidimensionais de 16 elementos, mas podem ser acedidas por índices linha, coluna usando o tipo de dados `mat4`.
- O formato nativo usado pelo WebGL (e pelo OpenGL) requer os elementos dispostos em memória percorrendo primeiro as colunas.
- A função `flatten()`, da biblioteca MV.js trata de converter o tipo de dados `mat4` (elementos dispostos por linhas), da biblioteca MV.js para o formato requerido pelas funções da API WebGL (elementos dispostos por colunas).
- A função `gl.uniformMatrix4fv()` tem um parâmetro para transpor automaticamente a matriz, mas na versão corrente tem que estar a **false**.

Envio duma matriz de transformação para um programa GLSL

- Supondo que temos uma matriz `m`, de tipo `mat4`, o seu envio para que um shader a possa usar é feito com:

`gl.uniformMatrix4fv(loc, false, flatten(m))`

sempre false!

obtido previamente com
`gl.getUniformLocation(...)`

Variável Javascript que
contém a matriz que a
aplicação pretende enviar

Pilhas de Transformações Geométricas

- Em muitas ocasiões (especialmente ao percorrer a base de dados da cena), pretende-se preservar a transformação corrente para uso posterior (Ver capítulo 9).
- Nas versões de OpenGL anteriores à versão 3.1, eram oferecidas várias pilhas de transformações para diferentes usos:
 - ModelView stack, Projection stack , Color stack, Texture stack
- A mesma funcionalidade pode ser facilmente recriada em Javascript usando objetos de tipo Array:

```
var stack = []  
stack.push(modelViewMatrix);  
...  
modelViewMatrix = stack.pop()
```