Computação Gráfica e Interfaces

2017-2018

Fernando Birra

Projeções - WebGL

2017-2018

Fernando Birra

Objetivos

- Saber como definir o volume de visão pretendido
- Conhecer as funções: ortho(), frustum() e perspective()
- Perceber o conceito de normalização do volume de visão e as suas vantagens para a implementação do recorte (clipping)

Problema

- Com a função lookAt(), ou outro mecanismo semelhante, posicionamos a câmara e definimos a sua orientação.

- Falta agora selecionar a lente, usando a analogia da câmara fotográfica, a qual irá determinar a projeção usada e o enquadramento. Por outras palavras, o volume de visão.

- Volume de visão: Região do espaço capturada pela câmara. Objetos fora desta região não aparecerão na imagem final.

Estratégia

- Vamos decompor o problema em 3 casos:
  - projeção ortogonal
  - projeção oblíqua
  - projeção perspetiva

Projeção ortogonal

- A matriz de projeção ortogonal correspondente ao alçado principal não define qual a parte do plano que irá ser visualizada.

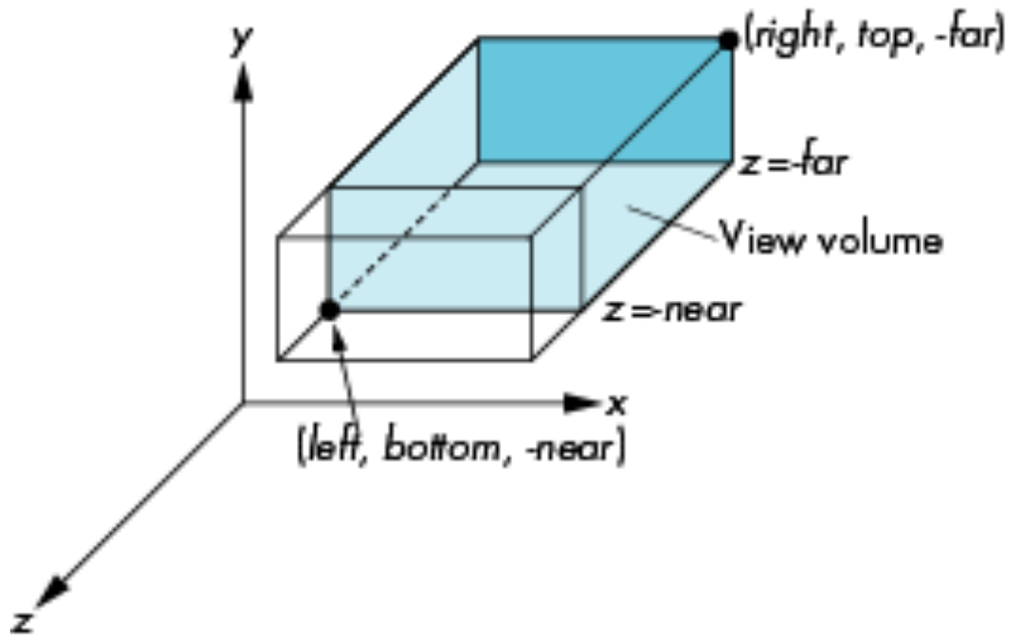
- Ao projetar no plano z=0, em WebGL, apenas a área do quadrado centrado na origem e de lado 2 irá ser visualizada.

- Recorde-se o volume de recorte de WebGL: cubo centrado na origem de lado 2.

Projeção ortogonal

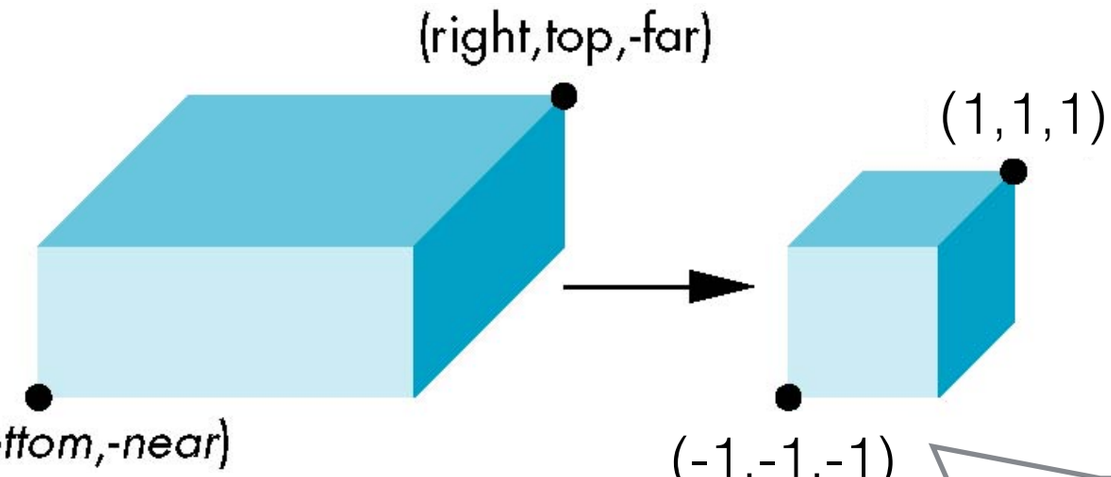
- Para definir um volume de visão correspondente a uma projeção paralela, com total controlo dos seus limites, podemos usar a função ortho() de MV.js:

`ortho(left,right,bottom,top,near,far)`



A função ortho() devolve uma matriz, a usar como matriz de projeção, que transforma o volume definido pelos seus parâmetros, no que o WebGL define como volume de recorte (Clip space)

Normalização do volume de visão



1º passo: mover o centro do volume para a origem

2º passo: efetuar a escala para transformar no cubo de Clip Coordinates

Por omissão, o referencial em Clip Coordinates do WebGL é esquerdo! (Z aponta para dentro do ecrã)

Normalização do volume de visão

- 1º passo:

$$T(-\frac{left + right}{2}, -\frac{bottom + top}{2}, \frac{near + far}{2})$$

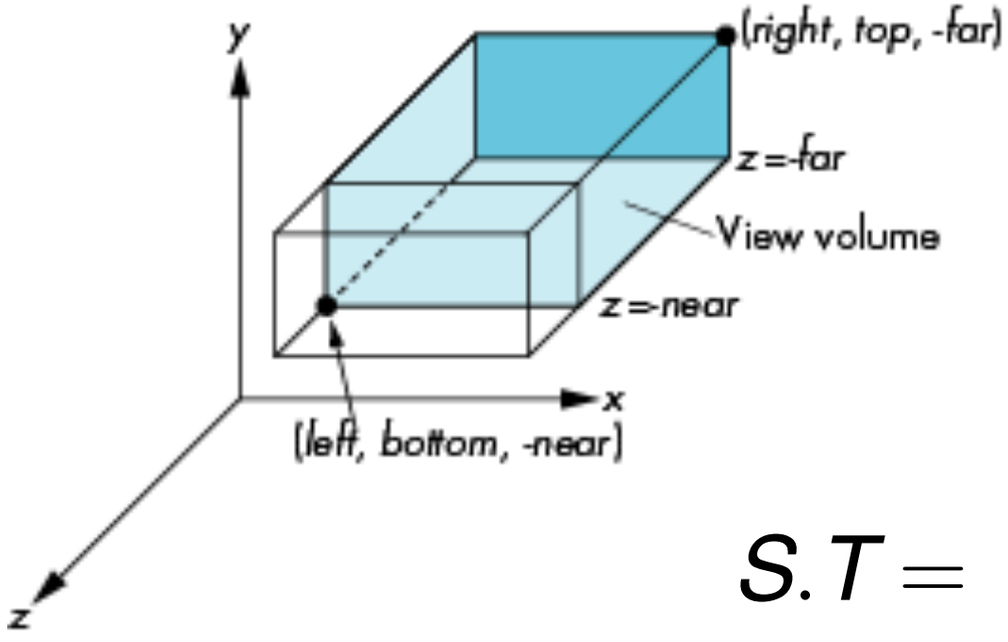
- 2º passo:

$$S(\frac{2}{right - left}, \frac{2}{top - bottom}, \frac{-2}{far - near})$$

Nota: A Transformação final é S.T

Normalização do volume de visão

`ortho(left,right,bottom,top,near,far)`



$$S.T = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{l+r}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{b+t}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{n+f}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projeção oblíqua

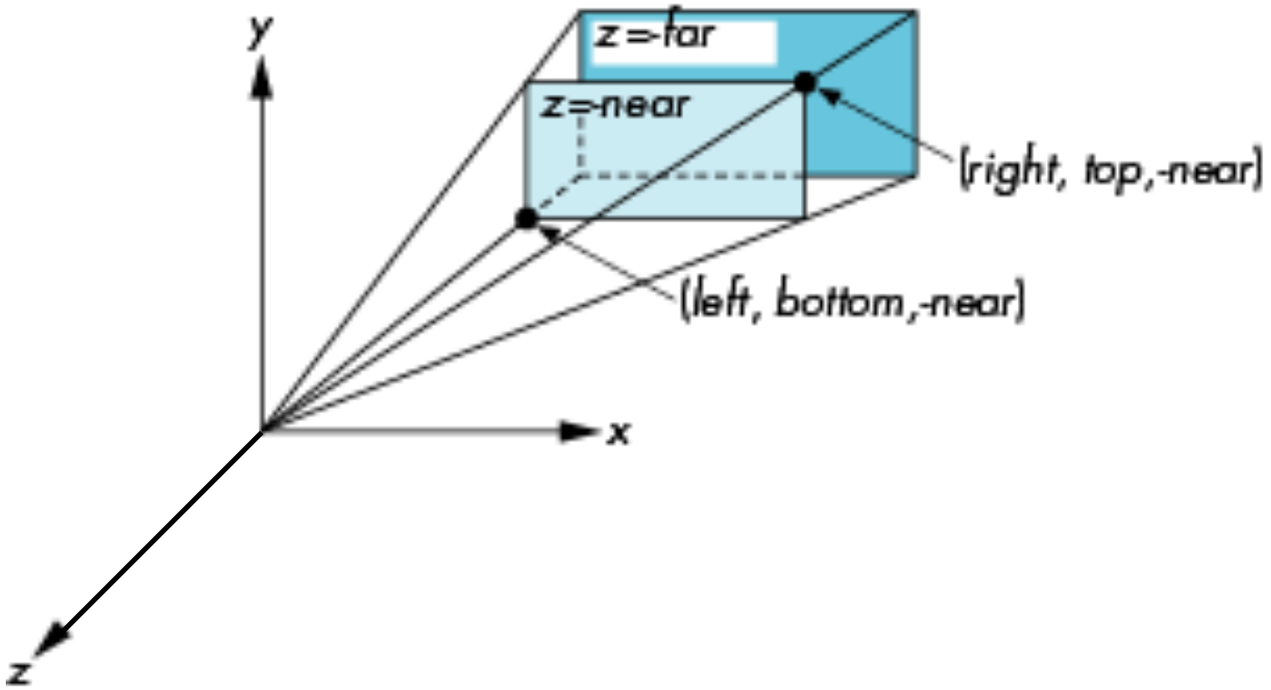
- A projeção oblíqua pode transformar-se numa projeção paralela, aplicando uma transformação de *shearing* para alinhar a direção de projeção com o eixo Z. De seguida aplica-se o resultado proveniente de ortho().

- Em alternativa, poderemos aplicar a matriz de projeção oblíqua deduzida anteriormente (mantendo a coordenada z) e aplicar de seguida uma matriz resultando duma chamada a ortho().

- O efeito combinado destas duas transformações é o da transformação do volume de visão da projeção oblíqua no volume de visão correspondente ao espaço Clip Coordinates do WebGL.

Projeção perspetiva (caso geral)

`frustum(left,right,bottom,top,near,far)`

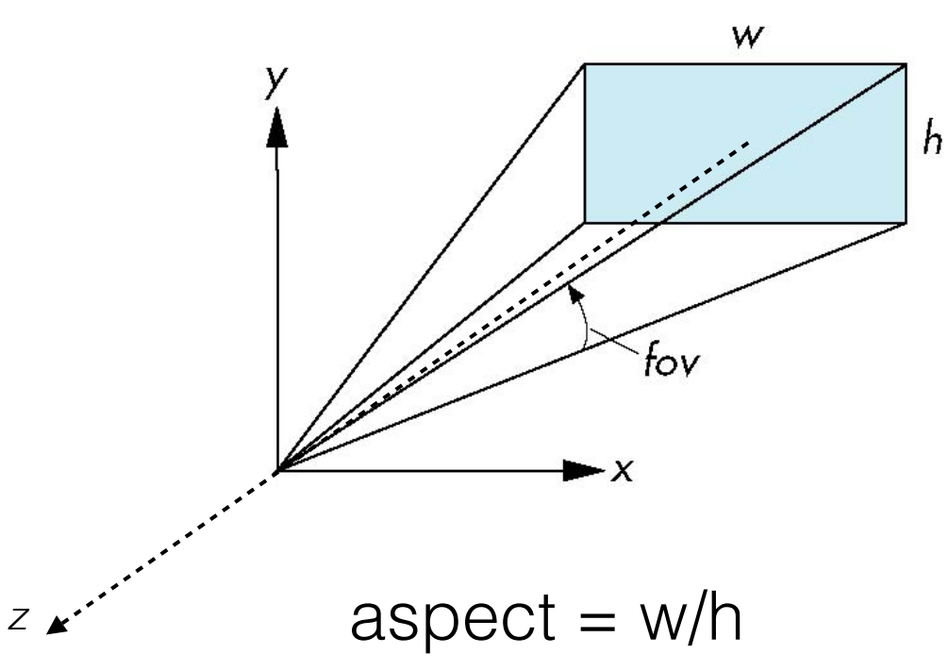


A função frustum() devolve uma matriz, a usar como matriz de projeção, que transforma o volume definido pelos seus parâmetros, no que o WebGL define como volume de recorte (Clip space)

O tronco de pirâmide é transformado num cubo.

Projeção perspetiva (pirâmide centrada no eixo Z)

`perpective(fovy, aspect, near, far)`

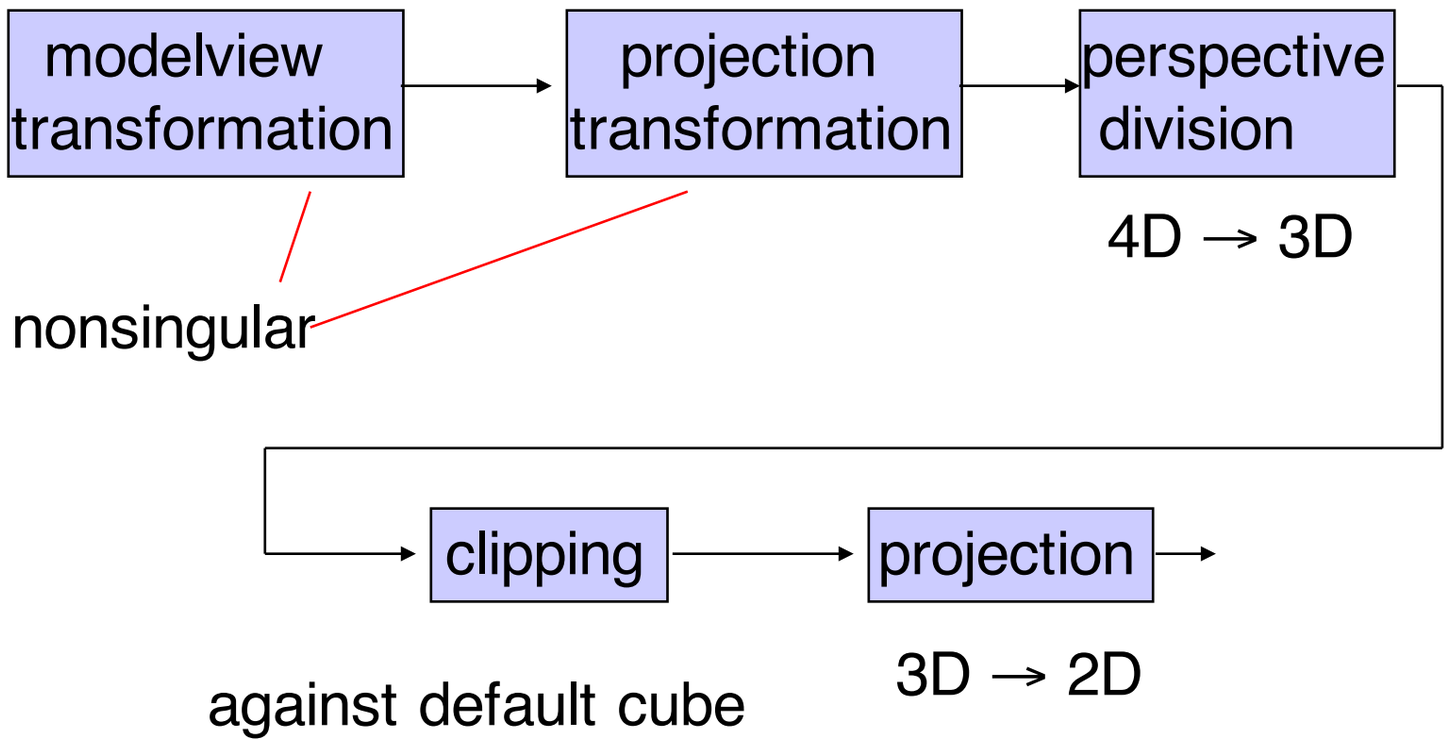


aspect = w/h

A função perspective() devolve uma matriz, a usar como matriz de projeção, que transforma o volume definido pelos seus parâmetros, no que o WebGL define como volume de recorte (Clip space)

O tronco de pirâmide é transformado num cubo.

Pipeline



Como as transformações de projeção transformam o volume de visão num cubo, centrado na origem e de lado 2 (Clip Space), pode usar-se sempre o mesmo procedimento de recorte!