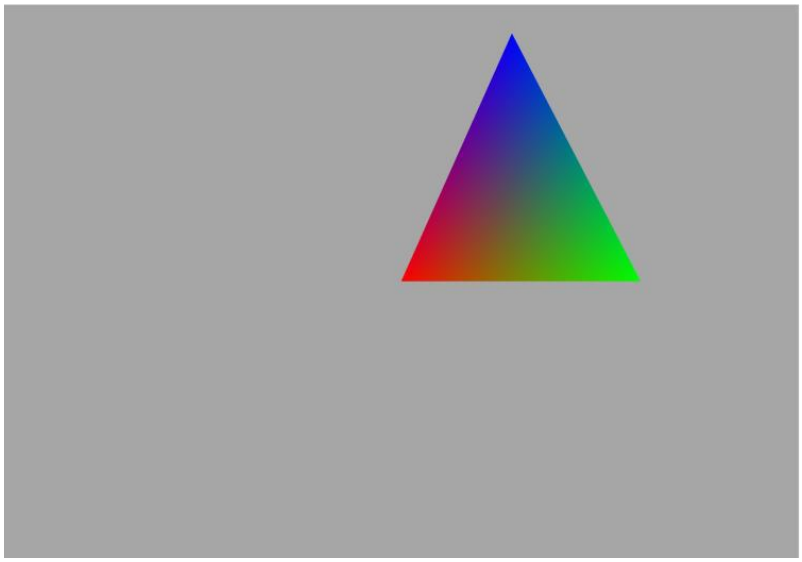
Tutorial 3 - Câmara Virtual

Objetivos:	O objetivo deste tutorial é aprenderes quais as matrizes que constituem uma câmara virtual.
Descrição:	Neste tutorial vais aprender a criar as diferentes matrizes que constituem uma câmara virtual e, utilizando o código do tutorial anterior, passar matrizes para dentro do vertex shader.
Resultados:	Como resultado final deverás conseguir criar uma câmara em perspetiva e ver o triângulo a rodar sobre o eixo do Y mas dando uma noção de perspetiva.

o final deverás ter:



Conteúdo

Introdução

Em Computação Gráfica, o processo de visualização é obtido através de uma superfície de visualização onde é desenhada toda a informação relativa a um determinado volume de visualização da cena (*i. e.* espaço do ambiente virtual que é considerado pela câmara virtual). Dadas as características da superfície de visualização e a forma como é obtida, esta pode ser comparada de forma análoga ao modo de funcionamento de uma câmara convencional e, por isso, adota-se a denominação de câmara virtual.

A câmara virtual pode ser parametrizada, sendo que existem os dois seguintes tipos de volume de visualização:

- **Volume de visualização ortogonal:** consiste num paralelepípedo cujo raios projetores são paralelos, sendo assim a profundidade fixa e não existindo o conceito de distância à câmara (Figura 1, à esquerda);

- **Volume de visualização perspetivo:** pode ser associado a uma pirâmide invertida cujo vértice se localiza na posição da câmara. Para delimitar a pirâmide de modo que o volume de visualização não seja infinito, este tipo de volume de visualização especifica o plano de recorte anterior e o plano de corte posterior (Figura 1, à direita). Este espaço delimitado abrangido pelo volume de visualização é também conhecido por *frustrum*. Este tipo de volume de visualização é definido ainda pelo campo de visão da câmara virtual (*field-of-view*) e a proporção do mesmo (relação entre a largura e altura da imagem da câmara virtual).

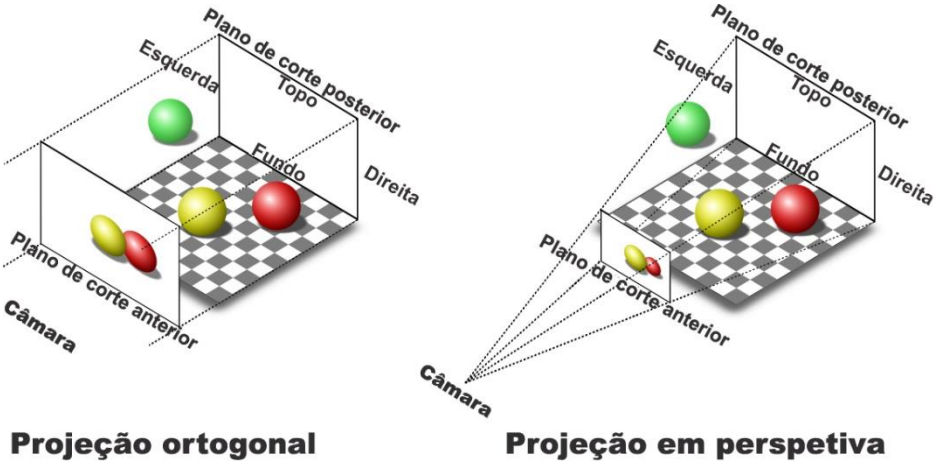


Figura 1 – Ilustração de volume de projeção ortogonal (esquerda) e perspetivo (direita) (adaptado de (Geo-F/X, 2017).

A configuração da câmara virtual é feita, essencialmente, através das seguintes matrizes:

- Matriz de visualização: permite movimentar ou rodar a câmara virtual;

- Matriz de projeção ortográfica: comprime tudo o que se encontra entre o plano anterior e o plano posterior dentro do volume de visualização;

- Matriz de projeção em perspetiva: comprime tudo o que se encontra entre o plano anterior e o plano posterior dentro do volume canónico (área que é renderizada) tendo em conta a distância entre a posição da câmara e a posição do objeto em coordenadas mundo (objetos mais próximos da câmara vão ser maiores que objetos mais longe da câmara);

- Matriz de *viewport*: transforma as coordenadas do *WebGL* (eixos *x* e *y* compreendidos entre -1 e 1) em coordenadas do dispositivo (eixos *x* e *y* compreendidos entre 0 e 1).

Tutorial

1. Ficheiros necessários

1.1. Assim como no tutorial anterior, vamos começar por criar uma nova pasta com o nome "Tutorial 3" e copiar os ficheiros e pastas que estão na pasta "Tutorial 2" para a pasta que acabaste de criar. Apaga todos os comentários (linhas que começam por //) que existem nesses ficheiros para perceberes melhor o que está a ser feito.

1.2. Agora abre o VSCode (Visual Studio Code), se aparecerem os ficheiros anteriores significa que ele guardou a pasta que estavas a trabalhar anteriormente e precisas de fechar essa pasta. Para fechares essa pasta vai a ***File -> Close Folder*** e serás levado para a página principal do VSCode. Seleciona "***Open Folder***" e seleciona a pasta "Tutorial 3" que criaste anteriormente.

1.3. Dentro da pasta "JavaScript" adiciona um novo ficheiro com o nome "camara.js". Neste ficheiro é onde irás crias os métodos para a criação de matrizes relativas à câmara.

1.4. Agora, abre o ficheiro "***index.html***" e adiciona as seguintes linhas de código assinaladas a vermelho:

```
<!DOCTYPE html>
<html>
    <head>
        <title>WebGL - Primeiro Cubo</title>
    </head>
    <body onload="Start()">
        <script src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/6.6.0/math.js"></script>
        <script src="./JavaScript/matrizes.js"></script>
        <!-- Script com as matrizes relativas à câmara -->
        <script src="./JavaScript/camara.js"></script>
        <script src="./JavaScript/shaders.js"></script>
        <script src="./JavaScript/app.js"></script>
    </body>
</html>
```

2. Criação das Matrizes da Câmara

Uma vez que já tens todos os ficheiros necessários, vamos começar por criar as diferentes matrizes pelas quais a câmara virtual é composta. Para isso abre o ficheiro "camara.js" que criaste no passo anterior e que se encontra dentro da pasta "JavaScript".

Como foi falado na aula teórica, o processo de renderização da câmara envolve diferentes matrizes tais como: matriz de visualização, matrizes de projeção e matriz de *viewport*. A seguir vamos ver para que servem cada uma delas.

Matriz de visualização

A matriz de visualização é a matriz que permite movimentar ou rodar a câmara virtual. Vamos então criar a matriz de visualização. Para isso copia o código da imagem seguinte para o ficheiro "camara.js".

```
/**
 * Função de devolve a matriz de visualização
 * @param {float[3]} rightVector Array direita da câmara
 * @param {float[3]} upVector Array cima da câmara
 * @param {float[3]} forwardVector Array frente da câmara
 * @param {float[3]} centerPoint array com a posição da câmara em coordenadas mundo.
 */
function MatrizDeVisualizacao(rightVector, upVector, forwardVector, centerPoint) {
    return [
        [rightVector[0],    rightVector[1],    rightVector[2],    -math.multiply(rightVector, centerPoint)],
        [upVector[0],       upVector[1],       upVector[2],       -math.multiply(upVector,centerPoint)],
        [forwardVector[0],  forwardVector[1],  forwardVector[2],  -math.multiply(forwardVector,centerPoint)],
        [0,                 0,                 0,                 1]
    ];
}

```

A função que acabaste de criar devolve um array de 2 dimensões com a matriz de visualização tendo em conta os parâmetros de entrada. Esses parâmetros de entrada são os vetores locais referentes à direita, cima e frente da câmara bem como a posição da câmara em coordenadas mundo.

Matriz de Projeção (Ortográfica)

A matriz de projeção ortográfica é a matriz que comprime tudo o que se encontra entre o plano anterior e o plano posterior dentro do volume canónico (área que é renderizada). Para criares a matriz de projeção ortográfica, copia o código da imagem seguinte para o ficheiro "camara.js".

```
16
17  /**
18   * Função que devolve a matriz de projeção Ortográfica
19   * @param {float} width Indica qual o comprimento da câmara que deve ser renderizada
20   * @param {float} height Indica qual a altura da câmara que deve ser renderizada
21   * @param {float} nearPlane Indica qual o plano de corte anterior da câmara
22   * @param {float} farPlane Indica qual o plano de corte posterior da câmara
23   */
24  function MatrizOrtografica(width, height, nearPlane, farPlane){
25      var matrizOrtografica = [
26          [1/width,   0,          0,                              0],
27          [0,         1/height,   0,                              0],
28          [0,         0,          1/((farPlane/2) - nearPlane),   -nearPlane/((farPlane/2)-nearPlane)],
29          [0,         0,          0,                              1]
30      ];
31
32      return math.multiply(matrizOrtografica, CriarMatrizTranslacao(0,0,-(nearPlane + farPlane / 2)));
33  }
34
```

A função que acabaste de criar devolve um array de 2 dimensões com a matriz de projeção ortográfica tendo em conta os parâmetros de entrada. Esses parâmetros de entrada são o comprimento da câmara, a altura da câmara, o plano de corte anterior e o plano de corte posterior, respetivamente.

NOTA: o *standard* de volumes canónicos tem os seus eixos X e Y compreendidos entre -1 e 1 e o eixo do Z compreendido entre 0 e 1. Isto não acontece em WebGL, uma vez que nesta tecnologia todos os eixos estão compreendidos entre -1 e 1. Foi então necessário ajustar esta matriz para comprimir os objetos entre 0 e 1 em vez de comprimir entre -1 e 1, daí a matriz ser diferente daquela dada na aula.

Matriz de Projeção (Perspetiva)

A matriz de projeção em perspetiva é a matriz que comprime tudo o que se encontra entre o plano anterior e o plano posterior dentro do volume canónico (área que é renderizada) tendo em conta a distância entre a posição da câmara e a posição do objeto em coordenadas mundo (objetos mais próximos da câmara vão ser maiores que objetos mais longe da câmara). Esta matriz tem também em conta qual a distância à qual a imagem vai ser renderizada. Para criares a matriz de projeção em perspetiva, copia o código da imagem seguinte para o ficheiro “camara.js”.

```
34
35  /**
36   * Função que devolve a matriz de projeção em perspetiva
37   * @param {float} distance Distância do centro que a imagem deve ser renderizada
38   * @param {float} width Indica qual o comprimento da câmara que deve ser renderizada
39   * @param {float} height Indica qual a altura da câmara que deve ser renderizada
40   * @param {float} nearPlane Indica qual o plano de corte anterior da câmara
41   * @param {float} farPlane Indica qual o plano de corte posterior da câmara
42   */
43  function MatrizPerspetiva (distance,width, height, nearPlane, farPlane)
44  {
45      return [
46          [distance/width,    0,              0,                                  0],
47          [0,                 distance/height,0,                                  0],
48          [0,                 0,              farPlane /(farPlane - nearPlane),   -nearPlane*farPlane/(farPlane- nearPlane)],
49          [0,                 0,              1,                                  0]
50      ];
51  }
52
```

A função que acabaste de criar devolve um array de 2 dimensões com a matriz de projeção em perspetiva tendo em conta os parâmetros de entrada. Esses parâmetros de entrada são a distância do plano ao qual os objetos serão projetados (para o mesmo tamanho de câmara, quanto maior a distância maior serão os objetos), o comprimento da câmara, a altura da câmara, o plano de corte anterior e o plano de corte posterior, respetivamente.

Matriz de Viewport

A matriz de *wiewport* é a matriz que transforma as coordenadas do webGl (eixos X e Y compreendidos entre -1 e 1) em coordenadas do dispositivo (eixos X e Y compreendidos entre 0 e 1). Para criares a matriz de *viewport* copia o código da imagem seguinte para o ficheiro "*camara.js*".

```
52
53  /**
54   * Função que devolve a matriz de Viewport
55   * @param {float} minX Valor mínino do volume canónico no eixo do X
56   * @param {float} maxX Valor máximo do volume canónico no eixo do X
57   * @param {float} minY Valor mínimo do volume canónico no eixo do Y
58   * @param {float} maxY Valor máximo do volume canónico no eixo do Y
59   */
60  function MatrizViewport(minX, maxX, minY, maxY){
61      return [
62          [(maxX - minX)/2,          0,                0,                (maxX + minX)/2],
63          [0,                        (maxY - minY)/2,  0,                (maxY + minY)/2],
64          [0,                        0,                1,                0],
65          [0,                        0,                0,                1]
66      ];
67  }
68
```

A função que acabaste de ciar devolve um array de duas dimensões com a matriz de *viewport* tendo em conta os parâmetros de entrada especificados. Estes parâmetros são o máximo e mínimo de cada eixo (X e Y entre -1 e 1).

3. Aplicar aos Objetos

Uma vez que agora tens todas as matrizes que são necessárias para para o bom funcionamento da câmara, é necessário que estas sejam aplicadas a todos os objetos. Para isso é necessário passar essas informações para dentro de cada Shader que exista dentro do programa. Vamos então adaptar o nosso *vertex shader* para receber estas informações.

```
1   var codigoVertexShader = [
2       'precision mediump float;'
3       ,
4       'attribute vec3 vertexPosition;',
5       'attribute vec3 vertexColor;',
6       ,
7       'varying vec3 fragColor;',
8       ,
9       'uniform mat4 transformationMatrix;',
10      'uniform mat4 visualizationMatrix;', // Matriz de Visualização
11      'uniform mat4 projectionMatrix;', // Matriz de Projeção
12      'uniform mat4 viewportMatrix;' // Matriz de Viewport
13      ,
14      'void main(){',
15      '    fragColor= vertexColor;',
16          // Depois de transformação geométrica é necessário multiplicar pelas matrizes de visualização, projeção e viewport.
17      '    gl_Position = vec4(vertexPosition, 1.0) * transformationMatrix * visualizationMatrix * projectionMatrix * viewportMatrix;',
18      '}'
19  ].join('\n');
20
```

Abre o ficheiro com o nome “shaders.js” e altera o código do *vertex shader* para ficar igual à imagem abaixo.

Agora que o *vertex shader* já está pronto para receber estas informações é necessário irmos ao nosso programa (app.js) e mandar essa mesma informação para o *shader*. Vamos então começar por criar três variáveis, uma por cada matriz criada no *vertex shader*, como mostra a imagem abaixo.

```
11
12  // Localização da variável 'visualizationMatrix'
13  var visualizationMatrixLocation;
14
15  // Localização da variável 'projectionMatrix'
16  var projectionMatrixLocation;
17
18  // Localização da mariável 'viewportMatrix'
19  var viewportMatrixLocation;
20
```

Estas três variáveis vão guardar a posição de cada uma das respetivas variáveis que se encontram dentro do *vertex shader*.

Agora que já temos onde guardar a localização de cada variável, é necessário ir buscar as respetivas posições. Para isso iremos utilizar a mesma função que utilizamos no tutorial anterior e que é demonstrada na imagem abaixo. Copia o código assinalado a vermelho na imagem abaixo para o fim da função “*SendDataToShaders()*”.

```
107
108     finalMatrixLocation = GL.getUniformLocation(program, 'transformationMatrix');
109
110     // Vai buscar qual o localização da variável 'visualizationMatrix' ao vertexShader
111     visualizationMatrixLocation = GL.getUniformLocation(program, 'visualizationMatrix');
112
113     // Vai buscar qual o localização da variável 'projectionMatrix' ao vertexShader
114     projectionMatrixLocation = GL.getUniformLocation(program, 'projectionMatrix');
115
116     // Vai buscar qual o localização da variável 'viewportMatrix' ao vertexShader
117     viewportMatrixLocation = GL.getUniformLocation(program, 'viewportMatrix');
118
119 }
120
```

Agora é necessário ir à função loop() e, utilizando as funções que criaste no início deste tutorial, criares uma câmara virtual. Para isso copia o código assinalado a vermelho das imagens seguintes para a mesma posição onde estão ilustradas.

```
finalMatrix = math.multiply(CriarMatrizRotacaoY(anguloDeRotacao), finalMatrix);

// Foi adicionada esta transformação de translação para podermos mexer na posição do objeto no eixo do Z
finalMatrix = math.multiply(CriarMatrizTranslacao(0, 0, 1), finalMatrix);

var newarray = [];
for(i = 0; i< finalMatrix.length; i++)
{
    newarray = newarray.concat(finalMatrix[i]);
}

// Utilizando a função MatrizDeVisualização vamos passar os parametros normais dos eixos e colocar
// a câmara no x=0, y=0 e z=0 do mundo. De seguida e como vimos anteriormente é necessário
// converter um array de 2 dimensões para um array de 1 dimensão
var visualizationMatrix = MatrizDeVisualizacao([1,0,0],[0,1,0],[0,0,1], [0,0,0]);
var newVisualizationMatrix = [];
for(i = 0; i< visualizationMatrix.length; i++)
{
    newVisualizationMatrix = newVisualizationMatrix.concat(visualizationMatrix[i]);
}

// Utilizando a função MatrizPerspetiva vamos passar os parametros de distância=10,
// comprimento da camera de 4 unidades, altura de 3 unidades, plano anterior de 0.1 unidades e
// plano posterior de 100 unidades. De seguida e como vimos anteriormente é necessário
// converter um array de 2 dimensões para um array de 1 dimensão
var projectionMatrix = MatrizPerspetiva(10,4,3,0.1,100);
var newProjectionMatrix = [];
for(i = 0; i< projectionMatrix.length; i++)
{
    newProjectionMatrix = newProjectionMatrix.concat(projectionMatrix[i]);
}
```

```
// Utilizando a função MatrizViewport vamos passar os parametros do volume canónico do webGl.
// O volume canónico do webGL tem o valor de x, y e z compreendidos entre -1 e 1. De seguida
// e como vimos anteriormente é necessário converter um array de 2 dimensões para um array de 1 dimensão
var viewportMatrix = MatrizViewport(-1,1,-1,1);
var newViewportMatrix = [];
for(i = 0; i< viewportMatrix.length; i++)
{
    newViewportMatrix = newViewportMatrix.concat(viewportMatrix[i]);
}

GL.uniformMatrix4fv(finalMatrixLocation, false, newarray);

// Função utilizada para passar o array de 1 dimensão relativo à matriz de visualização para o vertexShader.
GL.uniformMatrix4fv(visualizationMatrixLocation, false, newVisualizationMatrix);

// Função utilizada para passar o array de 1 dimensão relativo à matriz de projeção para o vertexShader.
GL.uniformMatrix4fv(projectionMatrixLocation, false, newProjectionMatrix);

// Função utilizada para passar o array de 1 dimensão relativo à matriz de viewport para o vertexShader.
GL.uniformMatrix4fv(viewportMatrixLocation, false, newViewportMatrix);

GL.drawArrays(
    GL.TRIANGLES,
    0,
    3
);
```

Se testares agora o teu jogo, verás um triângulo a rodar em perspetiva.

4. Desafio

Primeiro, corre o tutorial e verifica com atenção o que está a acontecer. De seguida, altera a linha de código seguinte:

```
// Foi adicionada esta transformação de translação para podermos mexer na posição do objeto no eixo do Z
finalMatrix = math.multiply(CriarMatrizTranslacao(0, 0, 1), finalMatrix);
```

para:

```
// Foi adicionada esta transformação de translação para podermos mexer na posição do objeto no eixo do Z
finalMatrix = math.multiply(CriarMatrizTranslacao(0, 0, 19), finalMatrix);
```

1 - Indica o que aconteceu e o porquê de ter acontecido.

De seguida, em vez de criares uma câmara em perspetiva vais criar uma matriz ortográfica. Para isso altera a linha de código abaixo:

```
var projectionMatrix = MatrizPerspectiva(10,4,3,0.1,100);
```

para:

```
var projectionMatrix = MatrizOrtografica(4,3,0.1,100);
```

Depois, disto altera o código:

```
finalMatrix = math.multiply(CriarMatrizTranslacao(0, 0, 99), finalMatrix);
```

para:

```
finalMatrix = math.multiply(CriarMatrizTranslacao(0, 0, 1), finalMatrix);
```

2 – Indica o que aconteceu ao triângulo e o porquê de ter acontecido.

ENTREGA

O trabalho deve ser submetido no MOODLE até dia **2/04/2021.**

A submissão é individual e deve conter todos os ficheiros necessários à correta execução da aplicação: uma pasta contendo o ficheiro .html e a pasta JavaScript com os respetivos ficheiros. A resposta aos desafios deve estar num ficheiro de texto que acompanha os ficheiros submetidos.