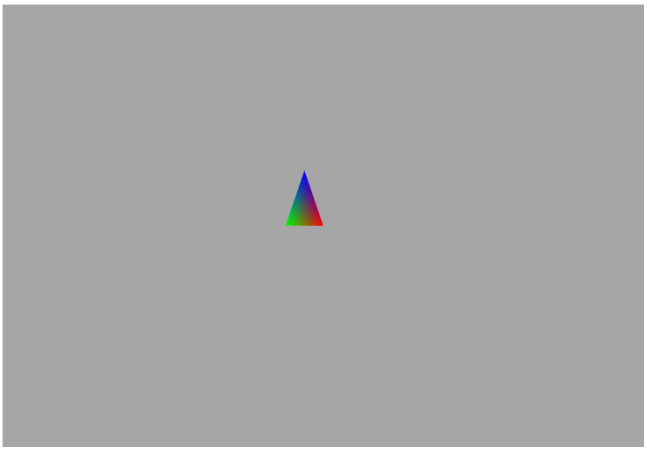
Tutorial 2

Transformações Geométricas

Objetivos:	O objetivo deste tutorial é aprenderes quais os passos necessários para rodares o triângulo que criaste no tutorial anterior.
Descrição:	Neste tutorial vais aprender a criar as diferentes matrizes de transformação geométricas para objetos 3D, utilizares variáveis "uniform" dentro dos shaders, para que servem essas mesmas variáveis e como enviar a matriz de transformação final para o vertexShader.
Resultados:	Como resultado final deverás ter um triângulo colorido a rodar sobre o eixo do Y.

No final deverás ter:



Conteúdo

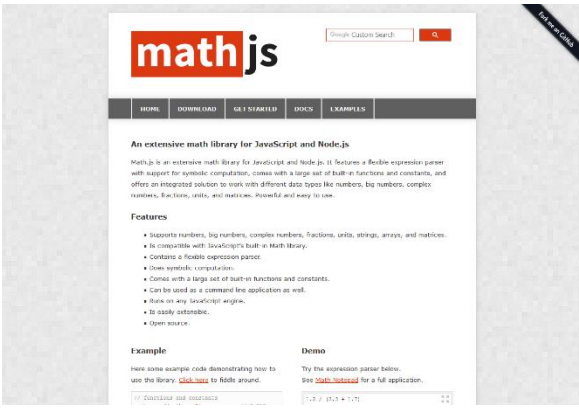
Ficheiros Necessários

Para facilitar a aprendizagem de WebGL vamos criar uma nova pasta para o segundo tutorial. Para isso dirige-te à pasta WebGL que criaste no Tutorial 2 e cria uma nova pasta com o nome "Tutorial 3". De seguida copia os ficheiros e pastas que criaste no tutorial anterior. Abre o Visual Studio Code, abre a pasta "Tutorial 3" e apaga todos os comentários (linhas que começam por //). Deste modo apenas terás os comentários deste tutorial e será mais fácil para ti perceberes o que está a ser feito.

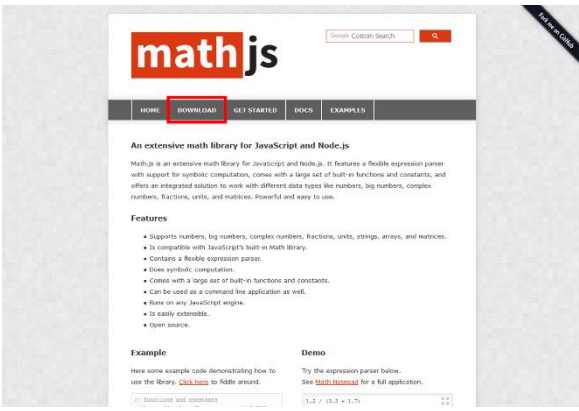
Depois de teres copiado os ficheiros e teres apagado todos os comentários vai ser necessário descarregares uma biblioteca de matemática que te irá permitir fazer multiplicação de matrizes. Para isso segue os passos seguintes:

1. Dentro da pasta "JavaScript" adiciona um novo ficheiro com o nome "matrizes.js". Neste ficheiro é onde irás crias os métodos para a criação de matrizes de transformações geométricas.

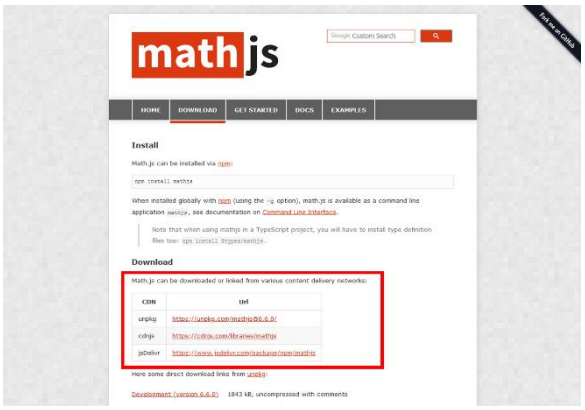
2. Abre o teu navegador, procura por "math js" e abre o primeiro link ou então carrega aqui. Deverás entrar num website igual à imagem ao lado.



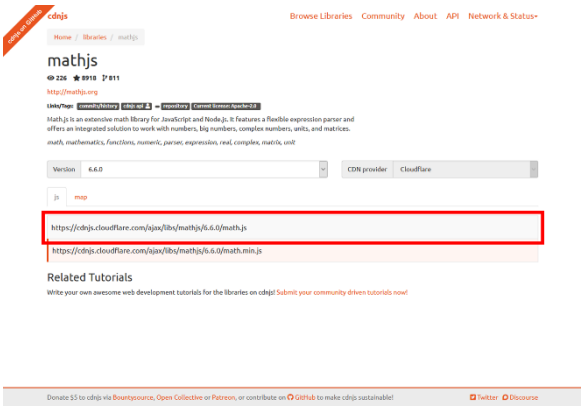
3. De seguida clica no botão "Download" ilustrado na imagem ao lado.



4. Na página que acabaste de entrar existe uma tabela (assinala na imagem ao lado), clica no segundo link.



5. Copia o link que esta assinalado a vermelho na imagem ao lado.



6. 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()">
        <!-- Biblioteca de matemática -->
        <script src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/6.6.0/math.js"></script>
        <script src="./JavaScript/matrizes.js"></script>
        <script src="./JavaScript/shaders.js"></script>
        <script src="./JavaScript/app.js"></script>
    </body>
</html>
```

Criar as Matrizes

Agora que já tens os ficheiros necessários vamos começar por criar as diferentes matrizes de transformações geométricas. Para isso abre o ficheiro que acabaste de criar com o nome "matrizes.js".
Como já deves ter aprendido na teórica, existem 3 transformações geométricas que podem ser aplicadas aos objetos, nomeadamente translações, rotações e modificações de escala. Cada uma delas tem a respetiva matriz de transformação em coordenadas homogéneas 3D.

Matriz de translação

Vamos começar pela matriz de translação, para isso copia o código da imagem abaixo para o ficheiro "matrizes.js".

```
/**
 * @param {float} x Valor para translação no eixo do X
 * @param {float} y Valor para translação no eixo do Y
 * @param {float} z Valor para translação no eixo do Z
 * Devolve um 2D array com a matriz de translação pedida
 */
function CriarMatrizTranslacao(x,y,z)
{
    // Matriz de translação final
    return[
        [1, 0, 0, x],
        [0, 1, 0, y],
        [0, 0, 1, z],
        [0, 0, 0, 1]
    ];
}
```

A função acabaste de copiar devolve um **array de duas dimensões** com a matriz de translação tendo em conta os parâmetros de entrada. Os parâmetros de entrada indicam as unidades de deslocação em cada eixo.

Matriz de mudança de Escala

A segunda matriz que vamos criar é referente à matriz de modificação de escala. No mesmo ficheiro adiciona a função demonstrada na imagem abaixo.

```
17
18  /**
19   * @param {float} x Valor para escala no eixo do X
20   * @param {float} y Valor para escala no eixo do Y
21   * @param {float} z Valor para escala no eixo do Z
22   * Devolve um 2D array com a matriz de escala pedida
23   */
24  function CriarMatrizEscala (x, y, z)
25  {
26      // Matriz de escala final
27      return [
28          [x, 0, 0, 0],
29          [0, y, 0, 0],
30          [0, 0, z, 0],
31          [0, 0, 0, 1]
32      ];
33  }
34
```

Esta função devolve um **array de duas dimensões** com a matriz de modificação de escala tendo em conta os parâmetros de entrada. Os parâmetros de entrada indicam qual a rácio de que deve ser aplicada, isto quer dizer que se for passado o valor de 0.25 num dos eixos, esse eixo vai ser reduzido para ¼.

NOTA: Deves ter sempre em atenção que a escala normal é 1 unidade e não 0 unidades como o resto das transformações.

Matrizes de Rotação

Como deves saber, existem 3 matrizes de rotação, uma para cada um dos eixos (X, Y e Z). Copia o código das imagens seguintes para o ficheiro "**matrizes.js**".

```
34
35  /**
36   * @param {float} angulo Ângulo em graus para rodar no eixo do X
37   */
38  function CriarMatrizRotacaoX(angulo)
39  {
40      // Seno e cosseno são calculados em radianos, logo é necessário converter de graus
41      // para radianos utilizando a linha a baixo.
42      var radianos = angulo * Math.PI/180;
43
44      // Matriz final de Rotação no eixo do X
45      return [
46          [1,             0,                          0,                          0],
47          [0,             Math.cos(radianos),         -Math.sin(radianos),        0],
48          [0,             Math.sin(radianos),         Math.cos(radianos),         0],
49          [0,             0,                          0,                          1]
50      ];
52
53  /**
54   * @param {float} angulo Ângulo em graus para rodar no eixo do Y
55   */
56  function CriarMatrizRotacaoY(angulo)
57  {
58      // Seno e cosseno são calculados em radianos, logo é necessário converter de graus
59      // para radianos utilizando a linha a baixo.
60      var radianos = angulo * Math.PI/180;
61
62      // Matriz final de rotação no eixo do Y
63      return [
64          [Math.cos(radianos),    0,          Math.sin(radianos),     0],
65          [0,                     1,          0,                      0],
66          [-Math.sin(radianos),   0,          Math.cos(radianos),     0],
67          [0,                     0,          0,                      1]
68      ];
69  }
70
71  /**
72   * @param {float} angulo Ângulo em graus para rodar no eixo do Z
73   */
74  function CriarMatrizRotacaoZ(angulo)
75  {
76      // Seno e cosseno são calculados em radianos, logo é necessário converter de graus
77      // para radianos utilizando a linha a baixo.
78      var radianos = angulo * Math.PI/180;
79
80      // Matriz final de rotação no eixo do Z
81      return[
82          [Math.cos(radianos),    -Math.sin(radianos),    0,      0],
83          [Math.sin(radianos),    Math.cos(radianos),     0,      0],
84          [0,                     0,                      1,      0],
85          [0,                     0,                      0,      1]
86      ];
87  }
88
```

Aplicar Transformações Geométricas

Agora que já tens as diferentes funções que criam cada uma das transformações geométricas, vais aprendes a utilizá-las. Para utilizares estas funções é necessário fazeres modificações ao vertexShader para receber a matriz final de transformação. Modifica então o código do vertexShader que se encontra no ficheiro "**shaders.js**" para ficar igual à imagem seguinte adicionando o código assinalado a vermelho.

```
var codigoVertexShader = [
    'precision mediump float;'
    ,
    'attribute vec3 vertexPosition;',
    'attribute vec3 vertexColor;',
    ,
    'varying vec3 fragColor;',
    ,
    // Matrix de 4x4 que indica quais as transformações que devem ser
    // feitas a cada um dos vértices.
    'uniform mat4 transformationMatrix;',
    ,
    'void main(){',
    '   fragColor= vertexColor;',
    '   gl_Position = vec4(vertexPosition, 1.0) * transformationMatrix;',
    '}'
].join('\n');

```

De seguida, no ficheiro "**app.js**" adiciona as seguintes variáveis.

```
var canvas = document.createElement('canvas');
canvas.width = window.innerWidth - 15;
canvas.height = window.innerHeight - 45;
var GL = canvas.getContext('webgl');
var vertexShader = GL.createShader(GL.VERTEX_SHADER);
var fragmentShader = GL.createShader(GL.FRAGMENT_SHADER);
var program = GL.createProgram();
var gpuArrayBuffer = GL.createBuffer();

// Variável que guarda a localização da variável 'transformationMatrix' do
// vertexShader.
var finalMatrixLocation;

// Variável que guarda a rotação que deve ser aplicada ao objeto.
var anguloDeRotacao = 0;

```

De seguida, na função com o nome "**SendDataToShaders()**", altera a função para ficar igual à imagem seguinte.

```
78
79  function SendDataToShaders() {
80      var vertexPositionAttributeLocation = GL.getAttribLocation(program, "vertexPosition");
81      var vertexColorAttributeLocation = GL.getAttribLocation(program, "vertexColor");
82
83      GL.vertexAttribPointer(
84          vertexPositionAttributeLocation,
85          3,
86          GL.FLOAT,
87          false,
88          6 * Float32Array.BYTES_PER_ELEMENT,
89          0 * Float32Array.BYTES_PER_ELEMENT
90      );
91
92      GL.vertexAttribPointer(
93          vertexColorAttributeLocation,
94          3,
95          GL.FLOAT,
96          false,
97          6 * Float32Array.BYTES_PER_ELEMENT,
98          3 * Float32Array.BYTES_PER_ELEMENT
99      );
100
101     GL.enableVertexAttribArray(vertexPositionAttributeLocation);
102     GL.enableVertexAttribArray(vertexColorAttributeLocation);
103
104     // Guarda a localização da variável 'transformationMatrix' do vertexShader
105     finalMatrixLocation = GL.getUniformLocation(program, 'transformationMatrix');
106
107     // Foi removido o código GL.useProgram(program); e GL.drawArrays(GL.TRIANGLES, 0, 3);
108 }
109
```

NOTA: Não te esqueças do fazer o que te é dito no último comentário da imagem a cima (Apagar as linhas GL.useProgram(progam); e GL.drawArrays(GL.TRIANGLES, 0, 3);)

Vais criar uma nova função com o nome "**loop()**" e copiar o código das imagens abaixo. Esta função é a função que irá utilizar as matrizes que criaste anteriormente.

```
109
110 // Função resposável pela animação(no nosso caso rodar ao triângulo)
111 function loop ()
112 {
113     // O código abaixo faz resize ao canvas de modo a ajustar-se ao tamanho
114     // da página web.
115     canvas.width = window.innerWidth - 15;
116     canvas.height = window.innerHeight - 45;
117     GL.viewport(0,0,canvas.width,canvas.height);
118
119     // É necessário dizer que program vamos utilizar.
120     GL.useProgram(program);
121
122     // a cada frame é ncessário limpar os buffers de profundidade e de cor
123     GL.clearColor(0.65, 0.65, 0.65, 1.0);
124     GL.clear(GL.DEPTH_BUFFER_BIT | GL.COLOR_BUFFER_BIT);
125
126     // Inicialização da variável que guarda a combinação de matrizes que
127     // vão ser passadas para o vertexShader.
128     var finalMatrix = [
129         [1,0,0,0],
130         [0,1,0,0],
131         [0,0,1,0],
132         [0,0,0,1]
133     ];
134
135     // A matriz final vai ser igual à multiplicação da matriz de translação com a matriz final.
136     // Esta matriz faz uma translação de 0.5 unidades no eixo do X, 0.5 unidades
137     // no eixo do Y e 0.0 unidades no eixo do Z.
138     finalMatrix = math.multiply(CriarMatrizTranslacao(0.5,0.5, 0), finalMatrix);
139
140     // A matriz final vai ser igual à multiplicação da matriz de escala com a matriz final.
141     // Esta matriz faz uma modificação na escala de 0.25 unidades no eixo do X, 0.25 unidades
142     // no eixo do Y e 0.25 unidades no eixo do Z. Quer isto dizer que o objeto irá ficar
143     // 4 vezes mais pequeno, sendo que para um objeto ter uma escala normal
144     // deverá ter 1 unidade em todos os eixos.
145     finalMatrix = math.multiply(CriarMatrizEscala(0.25,0.25,0.25),finalMatrix);
146
147     // A matriz final vai ser igual à multiplicação da matriz de rotação no eixo do Z
148     // com a matriz final. Esta matriz faz uma rotação anguloDeRotação unidades
149     // no eixo do Y.
150     finalMatrix = math.multiply(CriarMatrizRotacaoY(anguloDeRotacao), finalMatrix);
151
```

*continua na próxima página.

```
151
152     // Agora que já temos a matriz final de transformação, temos que converter de 2D array
153     // para um array de uma dimensão. Para isso utilizamos o código a baixo.
154     var newarray = [];
155     for(i = 0; i< finalMatrix.length; i++)
156     {
157         newarray = newarray.concat(finalMatrix[i]);
158     }
159
160     // Depois de termos os array de uma dimensão temos que enviar essa matriz para
161     // o vertexShader. Para isso utilizamos o código abaixo.
162     GL.uniformMatrix4fv(finalMatrixLocation,false ,newarray);
163
164     // Agora temos que mandar desenhar os triângulos
165     GL.drawArrays(
166         GL.TRIANGLES,
167         0,
168         3
169     );
170
171     // A cada frame é preciso atualizar o angulo de rotação.
172     anguloDeRotacao +=1;
173
174     // O código abaixo indica que no próximo frame tem que chamar
175     // a função passada por parametro. No nosso caso é a mesma função
176     // criando um loop de animação.
177     requestAnimationFrame(loop);
178 }
179
```

Depois de copiares o código da função "**loop()**" vais à função "**Start()**" e adicionas a linha de código assinalada a vermelho na imagem abaixo.

```
179
180 function Start() {
181     PrepareCanvas();
182     PrepareShaders();
183     PrepareProgram();
184     PrepareTriangleData();
185     SendDataToShaders();
186
187     // Quando acabar de preparar tudo chama a função de loop.
188     // Ao chamar o loop vai criar um loop de animação.
189     loop();
190 }
```

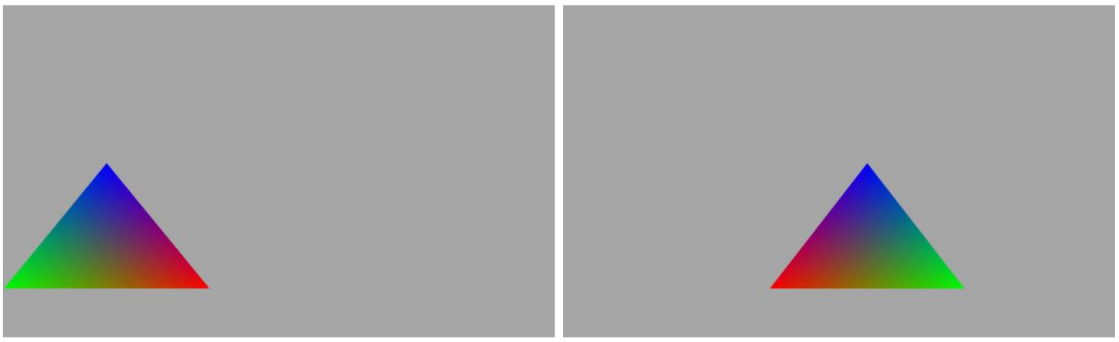
Depois disto tudo, se abrires o ficheiro "**index.html**" no teu navegador deverás ver o triangulo que criaste no tutorial anterior a rodar sobre o vértice vermelho. Caso não consigas ver esse triângulo, vai à consola do navegar ver qual o erro que está a acontecer.

DESAFIO

Sem apagar nenhuma linha de código resultante da realização do Tutorial 2, implementa as seguintes operações:

1. Aplica uma operação de escala, fazendo que o triângulo fique com 75% do tamanho do triângulo original

2. Aplica uma translação ao triângulo de forma a que ele fique a rodar posicionado do lado inferior direito tal como ilustra a imagem:



3. Faz com que o triângulo rode em dois eixos sem que saia do canvas

ENTREGA

O trabalho deve ser submetido no MOODLE até dia **26/03/2021.**

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.