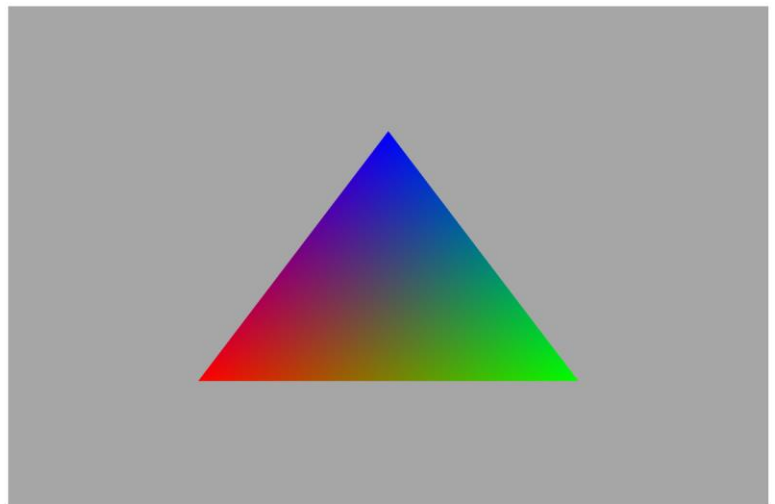


Tutorial 1

Primeiro Triângulo

No final deverás ter:



O canvas encontra-se acima deste texto!

Conteúdo

1	Ficheiros Necessários	2
2	Bases de WebGL	3

Objetivos: O objetivo deste tutorial é ensinar quais as funções base que todas as aplicações em WebGL necessitam e como utiliza-las.

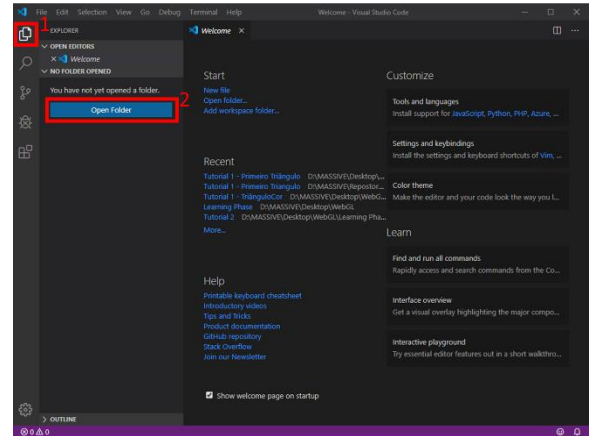
Descrição: Neste tutorial vais quais os ficheiros necessários para criares um projeto em WebGL. Vais aprender também a base de uma aplicação em WebGL, como criar um canvas, criar shaders, programas, e como passar informação para dentro dos shaders.

Resultados: A tua aplicação em WebGL deverá apresentar um triângulo colorido.

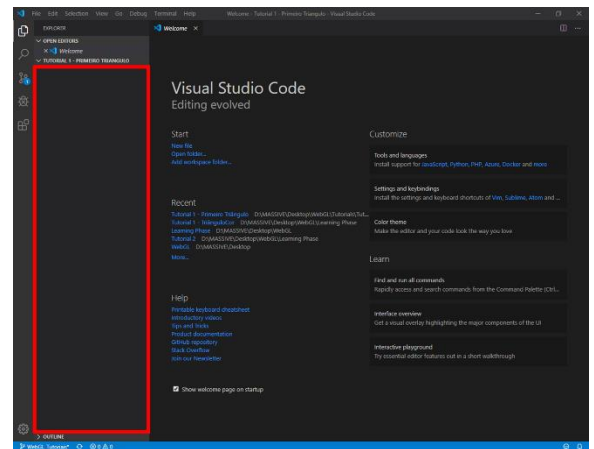
1 Ficheiros Necessários

Agora que já tens o software necessário instalado na tua máquina vais criar todos os ficheiros necessários para completares este tutorial. Para isso segue os passos seguintes:

1. No Visual Studio Code, abre a pasta que criaste no tutorial anterior (Provavelmente com o nome “WebGL - Tutoriais”). Depois de aberto, clica no segundo icon junto do nome da pasta para **criar** uma nova pasta com o nome “Tutorial 1 - Primeiro Triângulo”.



2. De seguida, carrega com o **botão do lado direito** do rato na área que está assinalada na imagem ao lado a vermelho. Nesse menu de contexto, clica com o botão direito do rato na pasta relativa ao tutorial 1 e seleciona “**New File**”, dando o nome de “**index.html**” ao ficheiro. Agora faz o mesmo procedimento, mas em vez de adicionares um ficheiro cria uma pasta (seleciona “**New Folder**”) e dá-lhe o nome de “JavaScript”. Carregando com o botão do lado direito em cima da pasta que acabaste de criar vais criar dois ficheiros, um com o nome “**app.js**” e outro com o nome “**shaders.js**”.



2 Bases de WebGL

Agora que já tens os ficheiros necessários criados, vais aprender as bases de WebGL necessárias para criares o teu primeiro projeto.

1. Vamos começar pelo ficheiro “*index.html*” e criar uma simples página web. Selecciona o ficheiro “*index.html*” e adiciona o texto da imagem seguinte:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>WebGL -- Primeiro Triângulo</title>
5   </head>
6   <body onload="Start()">
7     <!-- Scripts que correrão na página -->
8     <script src="./JavaScript/shaders.js"></script>
9     <script src="./JavaScript/app.js"></script>
10  </body>
11 </html>
```

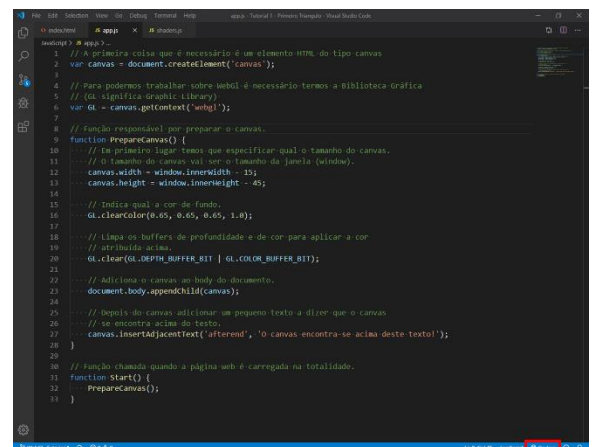
OBS1: Como podes ver pela imagem acima, a tag “*body*” tem um parâmetro “*onLoad*”, que indica qual a função que deverá chamar quando terminar de carregar a página.

OBS2: A tag “*script*” indica que os scripts que se encontram no caminho passado pelo parâmetro “*src*” serão utilizados nesta página.

2. De seguida seleciona o ficheiro “*app.js*” e copia o código que esta na imagem abaixo:

```
1 //A primeira coisa que é necessário é um elemento HTML do tipo canvas
2 var canvas = document.createElement('canvas');
3
4 //Em primeiro lugar temos que especificar qual o tamanho do canvas.
5 //O tamanho do canvas vai ser o tamanho da janela (window).
6 canvas.width = window.innerWidth - 15;
7 canvas.height = window.innerHeight - 45;
8
9 //Para podermos trabalhar sobre WebGL é necessário termos a Biblioteca Gráfica
10 //(GL significa Graphic Library)
11 var GL = canvas.getContext('webgl');
12
13 //Função responsável por preparar o canvas.
14 function PrepareCanvas() {
15
16     //Indica qual a cor de fundo.
17     GL.clearColor(0.65, 0.65, 0.65, 1.0);
18
19     //Limpa os buffers de profundidade e de cor para aplicar a cor
20     //atribuída acima.
21     GL.clear(GL.DEPTH_BUFFER_BIT | GL.COLOR_BUFFER_BIT);
22
23     //Adiciona o canvas ao body do documento.
24     document.body.appendChild(canvas);
25
26     //Depois do canvas adicionar um pequeno texto a dizer que o canvas
27     //se encontra acima do texto.
28     canvas.insertAdjacentText('afterend', 'O canvas encontra-se acima deste texto!');
29 }
30
31 //Função chamada quando a página web é carregada na totalidade.
32 function Start() {
33     PrepareCanvas();
34 }
35
```

Agora que copiaste o código da imagem acima, podes ver a página web. Tens duas maneiras de ver a página web, a primeira é ires a pasta do tutorial e fazeres duplo clique no ficheiro “*index.html*” a segunda é através do plugin que instalaste carregando no botão com o nome “*Go Live*” assinalado a vermelho na imagem ao lado (ou pelo atalho **Alt+L+O** ou **CMD+L+O**). Este plugin irá criar um servidor de HTTP local e, para além disso, quando guardares algum ficheiro, a página web recarregará automaticamente para refletir as



alterações feitas. Saberás que o servidor está ligado quando aparecer “Port:5500” (ou outro número) no lugar do botão.

Ao abrir a página web é suposto veres uma caixa cinzenta e no fundo da página o texto a indicar que o canvas está acima desse texto. Caso não apareça nada, abre as ferramentas do programador no browser (p. ex., no Chrome carrega na tecla **F12** e na aba “**Console**”) e vê qual o erro que te dá (se aparecer algum erro certifica-te que os passos anteriores estão corretos). Se não aparecer nada faz recarrega a página só para teres a certeza que não existem erros.

3. Depois de teres testado a página web e não teres erros, seleciona o ficheiro “**shaders.js**” no VSCode e copia o código que se encontra na imagem abaixo

```
1 //Código correspondente ao vertex shader
2 var codigoVertexShader = [
3     'precision mediump float; // indica qual a precisão do tipo float
4     ',
5     '//Variável read-only do tipo vec3 que indicará a posição de um vértice
6     'attribute vec3 vertexPosition;',
7     '//Variável read-only do tipo vec3 que indicará a cor de um vértice
8     'attribute vec3 vertexColor;',
9     ',
10    '//Variável que serve de interface entre o vertex shader e o fragment shader
11    'varying vec3 fragColor;',
12    ',
13    ',
14    'void main(){',
15    '//Dizemos ao fragment shader qual a cor do vértice.
16    '    fragColor = vertexColor;',
17    '//gl_Position é uma variável própria do Shader que indica a posição do vértice.
18    '//Esta variável é do tipo vec4 e a variável vertexPosition é do tipo vec3.
19    '//Por esta razão temos que colocar 1.0 como último elemento.
20    '    gl_Position = vec4(vertexPosition, 1.0);',
21    '}',
22 ].join('\n');
23
24 //Código correspondente ao fragment shader
25 var codigoFragmentShader = [
26     'precision mediump float; // indica qual a precisão do tipo float
27     ',
28     '//Variável que serve de interface entre o vertex shader e o fragment shader
29     'varying vec3 fragColor;',
30     ',
31     'void main(){',
32     '//gl_FragColor é uma variável própria do Shader que indica qual a cor do vértice
33     '//Esta variável é do tipo vec4 e a variável fragColor é do tipo vec3.
34     '//Por esta razão temos que colocar 1.0 como último elemento.
35     '    gl_FragColor = vec4(fragColor, 1.0);',
36     '}',
37 ].join('\n');
```

4. O próximo passo é criar os shaders que utilizem o código acima, para isso seleciona o ficheiro “*app.js*” e adiciona as variáveis que estão assinaladas a vermelho na imagem abaixo por baixo da definição da variável “*GL*”.

```
4 //Para podermos trabalhar sobre WebGL é necessário termos a Biblioteca Gráfica
5 //(GL significa Graphic Library)
6 var GL = canvas.getContext('webgl');
7
8 //Criar o vertex shader. Este shader é chamado por cada vértice do objeto
9 //de modo a indicar qual a posição do vértice.
10 var vertexShader = GL.createShader(GL.VERTEX_SHADER);
11
12 //Criar o fragment shader. Este shader é chamado para todos os píxeis do objeto
13 //de modo a dar cor ao objeto.
14 var fragmentShader = GL.createShader(GL.FRAGMENT_SHADER);
```

5. Depois de criadas as variáveis, é necessário dizeres qual o código que os shaders irão utilizar. Para isso vais criar uma função para preparar os shaders. Antes da função “*Start()*” copia o código da função “*PrepareShaders()*” da imagem abaixo e na função “*Start()*” adiciona a linha de código assinalada a vermelho.

```
40
41 //Função responsável por preparar os shaders.
42 function PrepareShaders()
43 {
44     ...//Atribui o código que está no ficheiro "shaders.js" ao vertexShader.
45     ... GL.shaderSource(vertexShader, codigoVertexShader);
46
47     ...//Atribui o código que está no ficheiro "shaders.js" ao fragmentShader.
48     ... GL.shaderSource(fragmentShader, codigoFragmentShader);
49
50     ...//Esta linha de código compila o shader passado por parâmetro.
51     ... GL.compileShader(vertexShader); //Compila o vertexShader.
52     ... GL.compileShader(fragmentShader); //Compila o fragmentShader.
53
54     ...//Depois de compilado os shaders é necessário verificar se ocorreu algum erro
55     ...//durante a compilação. Para o vertex shader usamos o código abaixo.
56     ... if(!GL.getShaderParameter(vertexShader, GL.COMPILE_STATUS)){
57     ...     ... console.error("ERRO :: A compilação do vertex shader lançou uma exceção!",
58     ...     ...     ... GL.getShaderInfoLog(vertexShader));
59     ... }
60
61     ...//Depois de compilado os shaders é necessário verificar se ocorreu algum erro
62     ...//durante a compilação. Para o fragment shader usamos o código abaixo.
63     ... if(!GL.getShaderParameter(fragmentShader, GL.COMPILE_STATUS)){
64     ...     ... console.error("ERRO :: A compilação do fragment shader lançou uma exceção!",
65     ...     ...     ... GL.getShaderInfoLog(fragmentShader));
66     ... }
67 }
68
69 //Função chamada quando a página web é carregada na totalidade.
70 function Start() {
71     ... PrepareCanvas();
72     ... PrepareShaders();
73 }
74
```


6. Os shaders por si só não são suficientes para que a GPU os utilize, é necessário criar programas que usem esses shaders. Para isso vais criar uma variável que guarde o programa que irá usar os *shaders* utilizando o código assinalado a vermelho na imagem abaixo.

```
8  // Criar o vertex shader. Este shader é chamado por cada vértice do objeto
9  // de modo a indicar qual a posição do vértice.
10 var vertexShader = GL.createShader(GL.VERTEX_SHADER);
11
12 // Criar o fragment shader. Este shader é chamado para todos os píxeis do objeto
13 // de modo a dar cor ao objeto.
14 var fragmentShader = GL.createShader(GL.FRAGMENT_SHADER);
15
16 // Criar o programa que utilizará os shaders.
17 var program = GL.createProgram();
18
```

7. De seguida vais acrescentar uma função que tem como objetivo atribuir e verificar se o programa se encontra em condições de ser executado na GPU. Copia a função “*PrepareProgram()*” e na função “*Start()*” acrescenta a linha de código assinalada a vermelho.

```
68
69 // Função responsável por preparar o Programa que irá correr sobre a GPU
70 function PrepareProgram(){
71     // Depois de teres os shaders criados e compilados é necessário dizeres ao program
72     // para utilizar esses mesmos shaders. Para isso utilizamos o código seguinte.
73     GL.attachShader(program, vertexShader);
74     GL.attachShader(program, fragmentShader);
75
76     // Agora que já atribuíste os shaders, é necessário dizeres à GPU que acabaste de
77     // configurar o program. Uma boa prática é verificar se existe algum erro no program
78     GL.linkProgram(program);
79     if(!GL.getProgramParameter(program, GL.LINK_STATUS)){
80         console.error("ERRO :: O linkProgram lançou uma exceção!", GL.getProgramInfoLog(program));
81     }
82
83     // É boa prática verificar se o programa foi conectado corretamente e se pode ser
84     // utilizado.
85     GL.validateProgram(program);
86     if(!GL.getProgramParameter(program, GL.VALIDATE_STATUS)){
87         console.error("ERRO :: A validação do program lançou uma exceção!", GL.getProgramInfoLog(program));
88     }
89
90     // Depois de tudo isto, é necessário dizer que queremos utilizar este program. Para isso
91     // utilizamos o seguinte código
92     GL.useProgram(program);
93 }
94
95
96 // Função chamada quando a página web é carregada na totalidade.
97 function Start() {
98     PrepareCanvas();
99     PrepareShaders();
100     PrepareProgram();
101 }
```

8.

Agora que já tens o programa a utilizar os shaders, é necessário criares uma variável que guarde o *buffer da GPU* para onde vais mandar os dados. Para isso utiliza o código assinalado a vermelho na imagem a baixo.

```
15
16 // Criar o programa que utilizará os shaders.
17 var program = GL.createProgram();
18
19 // Criar um buffer que está localizado na GPU para receber os pontos que
20 // os shaders irão utilizar.
21 var gpuArrayBuffer = GL.createBuffer();
22
```

9. Depois de criares o *buffer*, vais criar uma função que tem como objetivo guardar a posição XYZ de cada vértice bem como a *cor RGB* de cada ponto. Copia a função *“PrepareTriangleData()”* da imagem abaixo e na função *“Start()”* adiciona a linha de código assinalada a vermelho.

```
// Função responsável por criar/guardar a posição XYZ e cor RGB de cada um dos vértices do triângulo.
// Esta função é também responsável por copiar essa mesma informação para um buffer que se encontra na GPU.
function PrepareTriangleData() {
    // Variável que guardará os pontos de cada vértice (XYZ) bem como a cor de cada um deles (RGB)
    // Nesta variável, cada vértice é constituída por 6 elementos, lembra-te disso para os passos a seguir.
    // Outra coisa que te deves saber é que a área do canvas vai de -1 a 1 tanto em altura como em largura
    // com centro no meio da área do canvas. O código RGB tem valores compreendidos entre 0.0 e 1.0
    var triangleArray = [
        // X Y Z R G B
        -0.5, -0.5, 0.0, 1.0, 0.0, 0.0, // Vértice 1 da "imagem" ao lado --> ..... / \
        0.5, -0.5, 0.0, 0.0, 1.0, 0.0, // Vértice 2 da "imagem" ao lado --> ..... / \
        0.0, 0.5, 0.0, 0.0, 0.0, 1.0, // Vértice 3 da "imagem" ao lado --> ..... 1 ---- 2
    ];

    // Esta linha de código indica à GPU que o gpuArrayBuffer é do tipo ARRAY_BUFFER
    GL.bindBuffer(GL.ARRAY_BUFFER, gpuArrayBuffer);

    // Esta linha de código copia o array que acabamos de criar (triangleArray)
    // para o buffer que está localizado na GPU (gpuArrayBuffer).
    GL.bufferData(
        // Tipo de buffer que estamos a utilizar.
        GL.ARRAY_BUFFER,
        // Dados que pretendemos passar para o buffer que se encontra na GPU.
        // Importante saber que no CPU os dados do tipo float utilizam 64bits mas a GPU só trabalha com
        // dados de 32bits. O JavaScript permite-nos converter floats de 64bits para floats de 32bits utilizando
        // a função a baixo.
        new Float32Array(triangleArray),
        // Este parâmetro indica que os dados que são passados não vão ser alterados dentro da GPU.
        GL.STATIC_DRAW
    );

    // Função chamada quando a página web é carregada na totalidade.
    function Start() {
        PrepareCanvas();
        PrepareShaders();
        PrepareProgram();
        PrepareTriangleData();
    }
}
```


10. A última coisa que falta é inserires os dados que estão no buffer no vertex shader de modo a que estes sejam desenhados. Para isso vais criar uma função com o nome de “**SendDataToShaders()**” e copiares o código das 3 imagens seguintes (as três imagens seguintes são todas da mesma função). Quando acabares de copiar o código, adiciona a função que acabaste de criar à função de “**Start()**” como mostra a última imagem.

```
130
131 // Esta função é responsável por pegar na informação que se encontra no gpuArrayBuffer
132 // e atribuí-la ao vertex shader.
133 function SendDataToShaders(){
134     ....// A primeira coisa que é necessário fazer é ir buscar a posição de cada uma das variáveis dos Shaders.
135     ....// Se verificares o código dos shaders, é necessário passar informação para duas variáveis (vertexPosition
136     ....// e vertexColor). Para isso vamos utilizar o código abaixo.
137     ....var vertexPositionAttributeLocation = GL.getAttribLocation(program, "vertexPosition");
138     ....var vertexColorAttributeLocation = GL.getAttribLocation(program, "vertexColor");
139     ....
140     ....// Esta função utiliza o último buffer que foi feito binding. Como podes ver pela função anterior
141     ....// o último buffer ao qual foi feito bind foi o gpuArrayBuffer, logo ele vai buscar informação a esse
142     ....// buffer e inserir essa informação no vertex shader. Vamos inserir os dados para a variável vertexPosition.
143     ....GL.vertexAttribPointer(
144     ....// Localização da variável na qual pretendemos inserir a informação. No nosso caso a variável
145     ....// "vertexPosition"
146     ....vertexPositionAttributeLocation,
147     ....// Este parâmetro indica o quantos elementos vão ser usados pela variável. No nosso caso, a variável
148     ....// que irá utilizar estes valores é do tipo vec3 (XYZ) logo são 3 elementos.
149     ....3,
150     ....// Este parâmetro indica qual é o tipo dos objetos que estão nesse buffer. No nossa caso são FLOATs.
151     ....GL.FLOAT,
152     ....// Este parâmetro indica se os dados estão ou não normalizados. Para já este parametro pode ser false.
153     ....false,
154     ....// Este parametro indica qual o tamanho de objetos que constituem cada ponto do triângulo em bytes.
155     ....// Cada ponto do triângulo é constituído por 6 valores (3 para posição X.Y.Z e 3 para a cor R.G.B) e
156     ....// o array que está no buffer é do tipo Float32Array. Float32Array tem uma propriedade que indica
157     ....// qual o número de bytes que cada elemento deste tipo usa. Basta multiplicar 3 pelo numero de
158     ....// bytes de um elemento.
159     ....6 * Float32Array.BYTES_PER_ELEMENT,
160     ....// Este parâmetro indica quando elementos devem ser ignorados no início para chegar aos valores que
161     ....// pretendemos utilizar. No nosso caso queremos utilizar os primeiros 3 elementos. Este valor também
162     ....// é em bytes logo multiplicamos pelo número de bytes de um Float32Array.
163     ....0 * Float32Array.BYTES_PER_ELEMENT
164     ....);
165
```

```
165  
166 // Agora utilizando o mesmo método acima, vamos inserir os dados na variável vertexColor.  
167 // Se prestares atenção nos parâmetros desta função, é bastante parecido ao método anterior, mudando apenas  
168 // a variável à qual pretendemos inserir os dados (vertexColor) e o último parâmetro (uma vez que agora  
169 // pretendemos ignorar os primeiros 3 valores que significam a posição de cada vértice)  
170 GL.vertexAttribPointer(  
171 // Localização da variável na qual pretendemos inserir a informação. No nosso caso a variável  
172 // "vertexPosition"  
173 vertexColorAttributeLocation,  
174 // Este parâmetro indica o quantos elementos vão ser usados pela variável. No nosso caso, a variável  
175 // que irá utilizar estes valores é do tipo vec3 (XYZ) logo são 3 elementos.  
176 3,  
177 // Este parâmetro indica qual é o tipo dos objetos que estão nesse buffer. No nosso caso são FLOATs.  
178 GL.FLOAT,  
179 // Este parâmetro indica se os dados estão ou não normalizados. Para já este parâmetro pode ser false.  
180 false,  
181 // Este parâmetro indica qual o tamanho de objetos que constituem cada ponto do triângulo em bytes.  
182 // Cada ponto do triângulo é constituído por 6 valores (3 para posição X.Y.Z e 3 para a cor R.G.B) e  
183 // o array que está no buffer é do tipo Float32Array. Float32Array tem uma propriedade que indica  
184 // qual o número de bytes que cada elemento deste tipo usa. Basta multiplicar 3 pelo número de  
185 // bytes de um elemento.  
186 6 * Float32Array.BYTES_PER_ELEMENT,  
187 // Este parâmetro indica quando elementos devem ser ignorados no início para chegar aos valores que  
188 // pretendemos utilizar. No nosso caso queremos utilizar os primeiros 3 elementos. Este valor também  
189 // é em bytes logo multiplicamos pelo número de bytes de um Float32Array.  
190 3 * Float32Array.BYTES_PER_ELEMENT  
191 );  
192  
193 // Agora é necessário ativar os atributos que vão ser utilizados e para isso utilizamos a linha seguinte.  
194 // Temos de fazer isso para cada uma das variáveis que pretendemos utilizar.  
195 GL.enableVertexAttribArray(vertexPositionAttributeLocation);  
196 GL.enableVertexAttribArray(vertexColorAttributeLocation);  
197  
198 // Indica que vais utilizar este programa  
199 GL.useProgram(program);
```

```
200  
201 // Indica à GPU que pode desenhar  
202 GL.drawArrays(  
203 GL.TRIANGLES, // Parâmetro que indica que tipo de objetos pretendes desenhar  
204 0, // Qual o primeiro elemento que deve ser desenhado (elemento na posição 0)  
205 3 // Quantos elementos devem ser desenhados  
206 );  
207 }  
208  
209 // Função chamada quando a página web é carregada na totalidade.  
210 function Start() {  
211 PrepareCanvas();  
212 PrepareShaders();  
213 PrepareProgram();  
214 PrepareTriangleData();  
215 SendDataToShaders();  
216 }
```

11. Vai ao teu browser e atualiza a página (caso estejas a utilizar o live server, não será necessário atualizar a página pois a aplicação corre em tempo real). O resultado final deve ser semelhante à imagem seguinte:

