

LAB1: Programação Gráfica em WebGL

Prof. Antonio L. Apolinário Junior

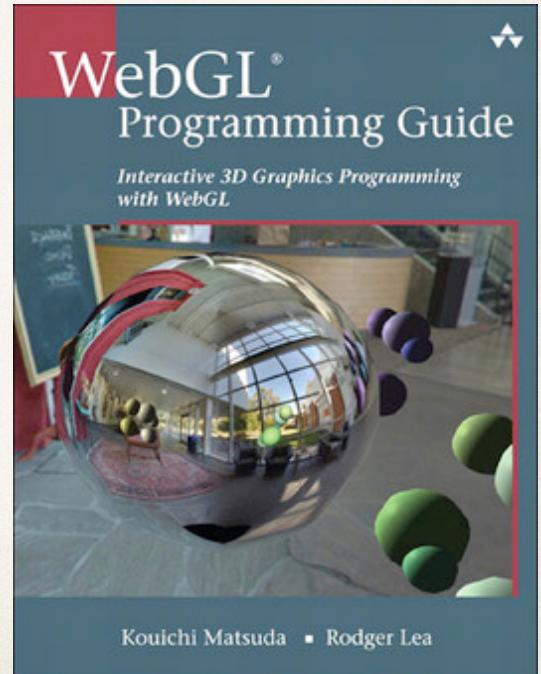
Ufba - Im - Dcc - Bcc - 2014.1

Roteiro

- O que é WebGL ?
- Como funciona ?
- O que se pode fazer?
- Como se faz?

Leitura de referencia

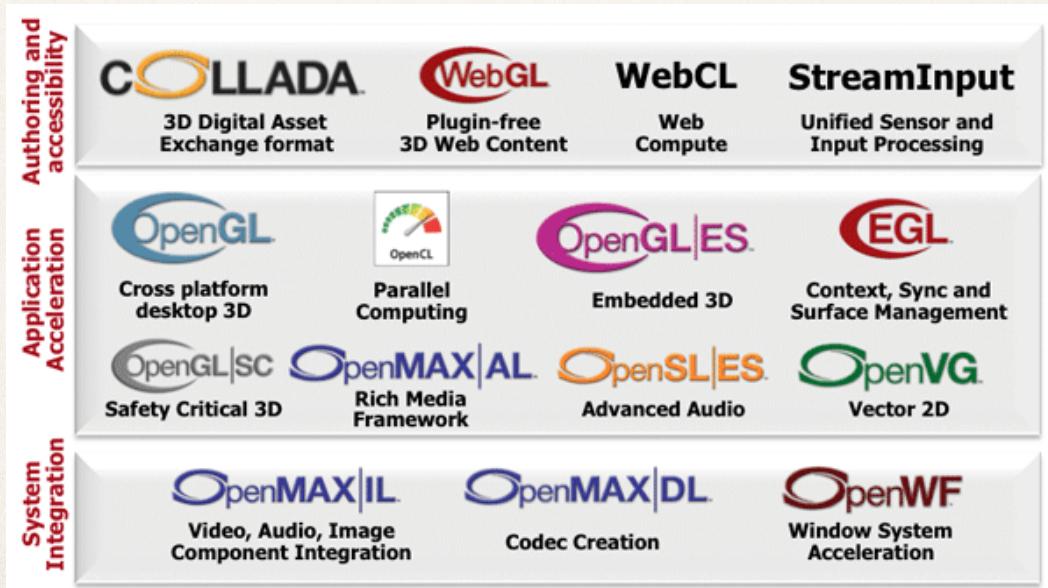
- Capitulos 1 e 2
- WebGL Programming Guide:
Interactive 3D Graphics Programming
with WebGL**
- 1st edition
- Kouichi Matsuda, Rodger Lea.
Addison-Wesley Professional - 2013.



O que é WebGL ?

O que é WebGL

- Parte do ecossistema baseado em OpenGL (*Open Graphics Library*)



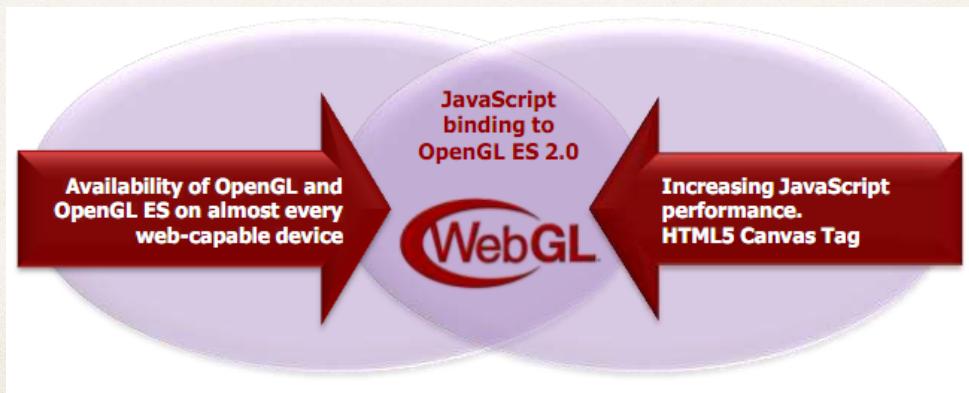
O que é WebGL

- WebGL = *OpenGL ES 2.0 for the Web*



O que é WebGL

- WebGL:
 - versão “leve” da OpenGL
 - *Binding* com JavaScript
 - Suportado pelo objeto *canvas* do HTML5



O que é WebGL

- Derivado do OpenGL | ES
(*OpenGL for Embedded Systems*)
 - Suporta:
 - *Vertex shaders*
 - *Fragment shaders*
 - *Vertex buffers*
 - *Textures*
 - *Framebuffers*
 - *Render states*
 - ...
 - Não suporta:
 - *Geometry shaders*
 - *Tessellation shaders*
 - *Vertex Array Objects*
 - *Multiple render targets*
 - *Floating-point textures*
 - *Compressed textures*
 - *FS depth writes*
 - ...

O que é WebGL

- Suportado por quase todos os navegadores mais atuais:

HTML5 Graphics & Embedded Content												
	WIN						MAC					
	Firefox	Safari	IE				Chrome	Opera	Firefox	Safari	Opera	
	3.6	4	5	6	7	8	9	10	11.1	4	5	11.1
Canvas	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	87%
Canvas Text	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	85%
SVG	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	86%
SVG Clipping Paths	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	86%
SVG Inline	✗	✓	✗	✗	✗	✗	✓	✓	✗	✓	✗	23%
SMIL	✗	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	49%
WebGL	✗	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	32%
Audio	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	85%
Video	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	85%

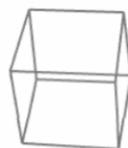
O que é WebGL

- Como verificar?
 - <http://get.webgl.org>

Your browser supports WebGL

However, it indicates that support is experimental; you might see issues with some content.

You should see a spinning cube. If you do not, please [visit the support site for your browser](#).



O que é WebGL

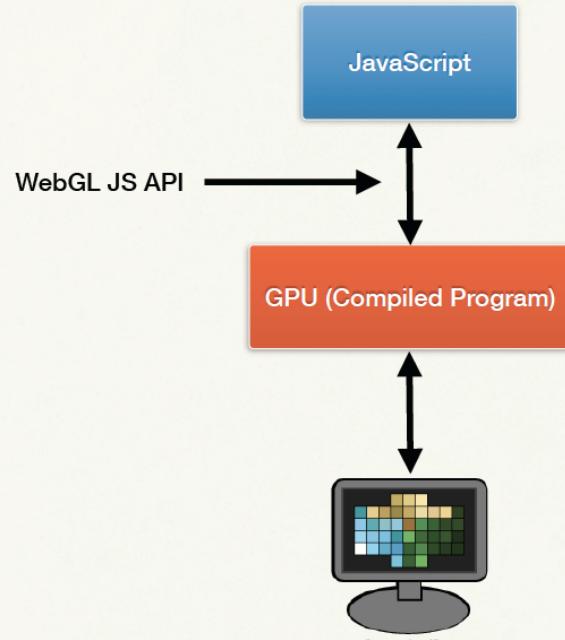
- ❖ Documentação, especificação, etc..
 - ❖ <http://www.khronos.org/webgl/>



Como funciona ?

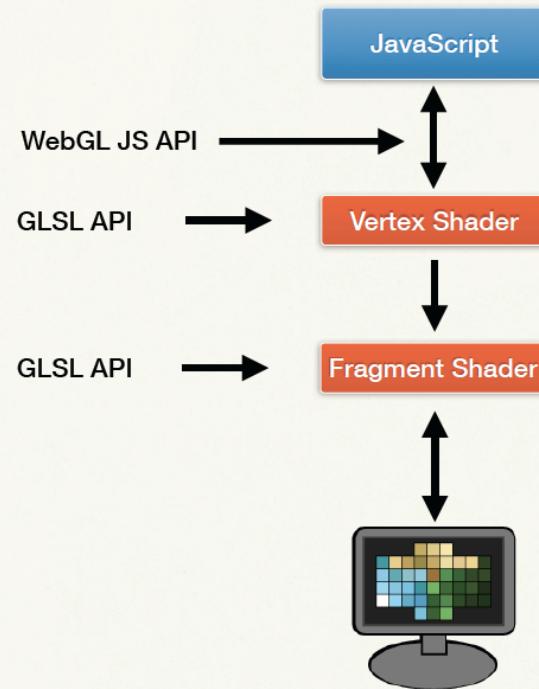
Como funciona ?

- Do ponto de vista da aplicação Web
 - Fornece uma API integrada a linguagem JavaScript
 - Fornece acesso a GPU



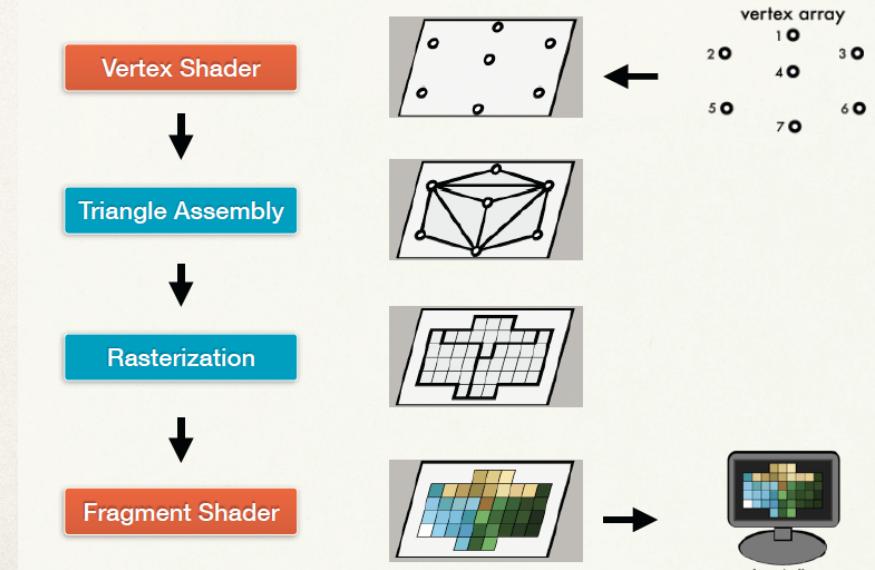
Como funciona ?

- Do ponto de vista da aplicação gráfica
 - API para acesso a GPU
 - Linguagem GLSL (*OpenGL Shading Language*)
 - programa na GPU => *shader*
 - 2 níveis de programação
 - *Vertex Shader*
 - *Fragment Shader*



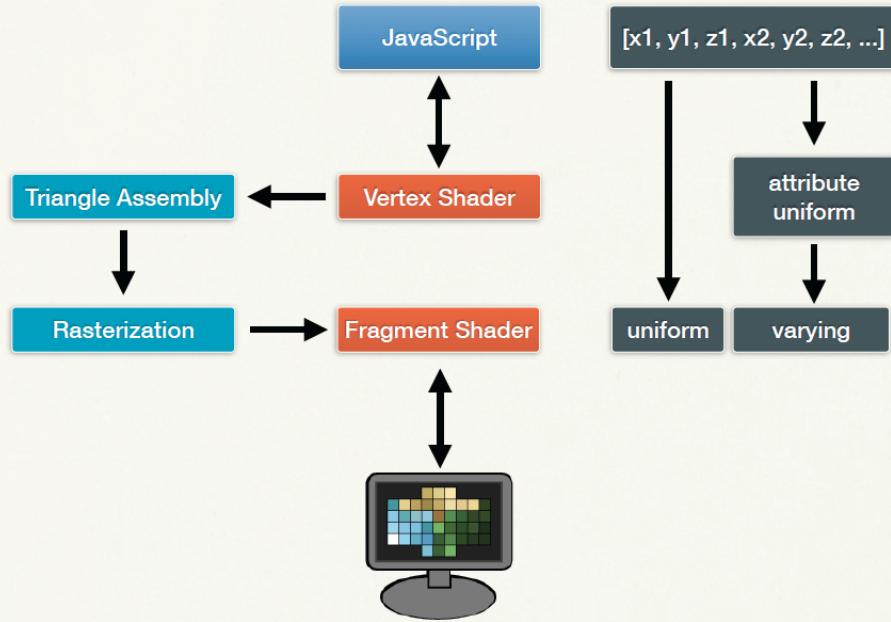
Como funciona ?

- Processo de Renderização



Como funciona ?

- Processo de Renderização



O que se pode fazer?

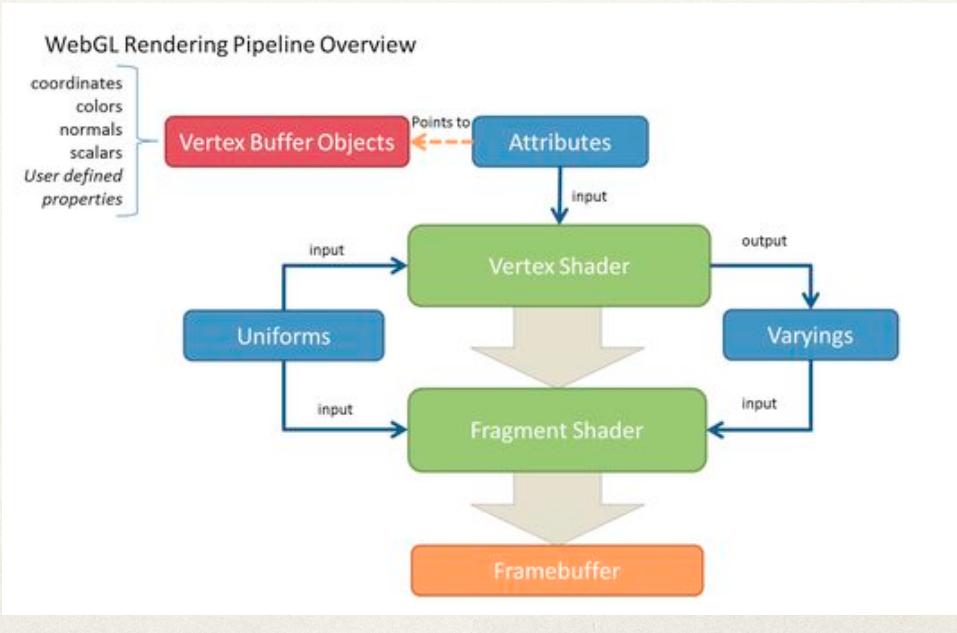
O que se pode fazer?

- ❖ Aplicações nas mais diversas áreas:
 - ❖ Jogos
 - ❖ Visualização de dados

Como se faz?

Pipeline Gráfico WebGL

- Baseado em *shaders*



Programa mínimo em WebGL

- Código HTML5:

- Cria o canvas no qual será definido o contexto **WebGL**:

```
<html>
  <head>
    <title>MATA65 - Computação Gráfica</title>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <script type="text/javascript" src="../lib/webgl-utils.js"></script>
    <script type="text/javascript" src="soCanvas.js">
    </script>
  </head>
  <body onload="webGLStart();">
    <h1>Laboratorio 1 - Exemplo 1</h1><br />
    <p>Verifica se ha suporte a WebGL e abre um canvas.</p>
    <canvas id="SoCanvas" style="border: none;" width="500" height="500"></canvas>
  </body>
</html>
```

Programa mínimo em WebGL

- No carregamento da página HTML:

```
// ****
// ****
function webGLStart() {
  var canvas = document.getElementById("SoCanvas");
  var gl = initGL(canvas);

  if (!gl) {
    alert("Could not initialise WebGL, sorry :-(");
    return;
  }
  drawScene(gl);
}
```

Programa mínimo em WebGL

- ✿ Inicializando o contexto WebGL:

```
// ****
// ****
function initGL(canvas) {
    gl = canvas.getContext("webgl");
    if (!gl) {
        return null;
    }
    gl.viewportWidth = canvas.width;
    gl.viewportHeight = canvas.height;
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    return gl;
}
```

Programa mínimo em WebGL

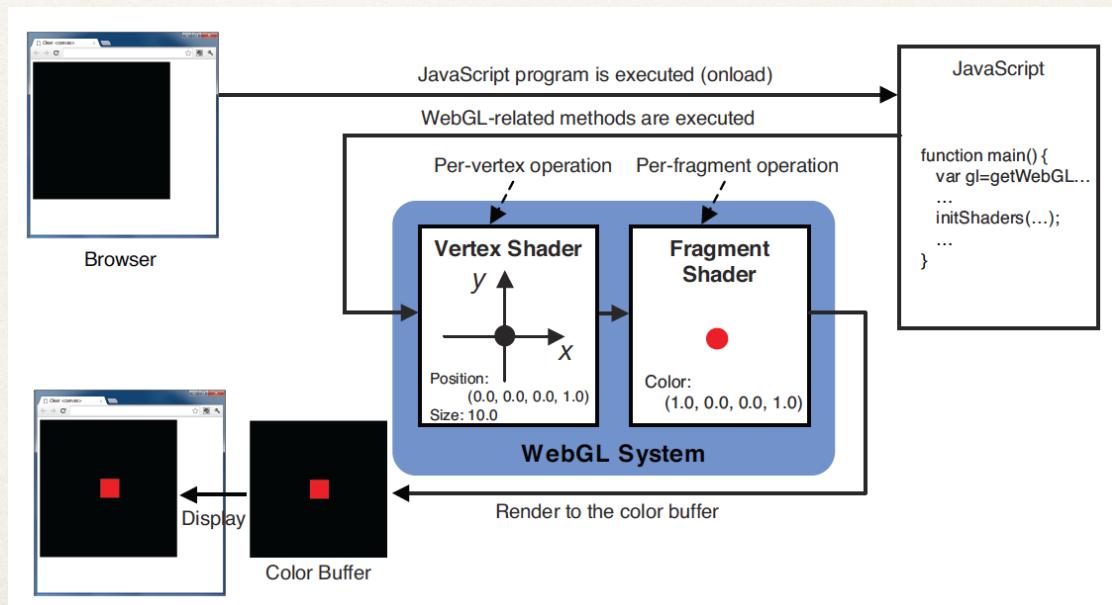
- ✿ Desenhando no contexto WebGL:

```
// ****
// ****
function drawScene(gl) {
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
}
```

- ✿ Como não há geometria envolvida
 - ✿ *shaders* não foram definidos

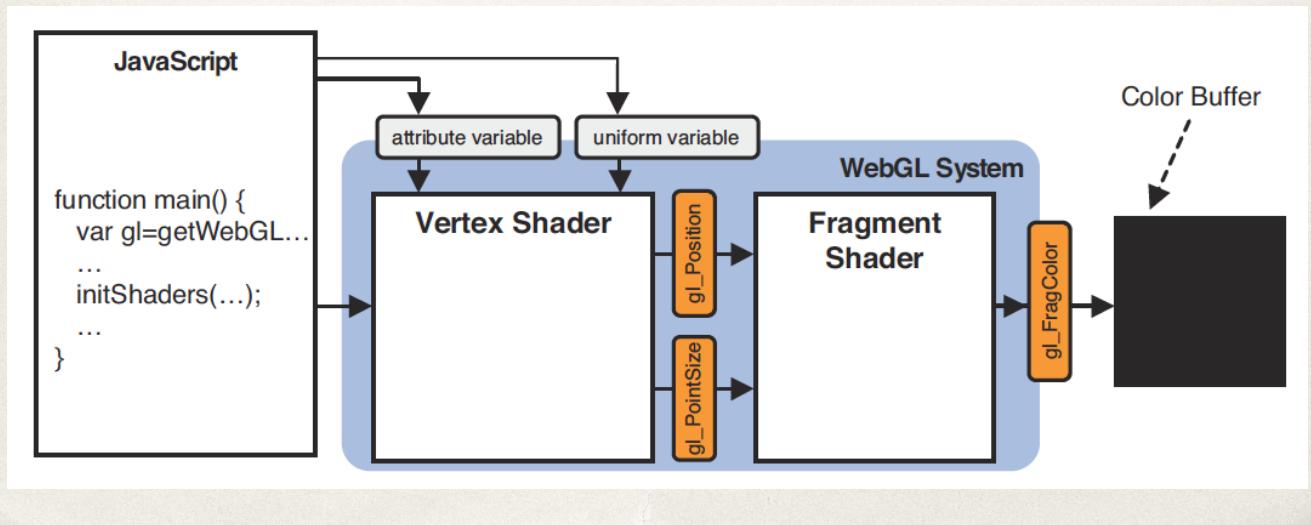
Renderização em WebGL

- Processo de desenho em WebGL:



Renderização em WebGL

- Passagem de parâmetros para os shaders:
 - Attributes
 - Uniforms



Renderização em WebGL

- Shaders mínimos:

- Vertex Shader

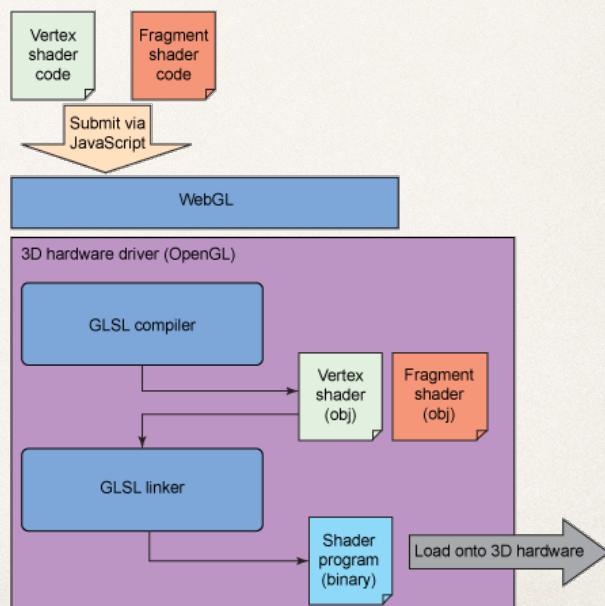
```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
    }
</script>
```

- Fragment Shader

```
<script id="shader-fs" type="x-shader/x-fragment">
    precision mediump float;
    void main(void) {
        gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
    }
</script>
```

Renderização em WebGL

- Para inicializar os *shaders*:
- Os *shaders* precisam ser:
 - carregados/lidos
 - compilados
 - linkados
 - Função **initShaders()**
- Carregados na GPU
 - **gl.useProgram(shader)**



Renderização em WebGL

- Como transferir dados para os *shaders* renderizarem?
 - *Vertex Arrays / Buffer Objects*
 - Descrevem objetos gráficos vetoriais
 - Vertices
 - Cores
 - Normais
 - etc...

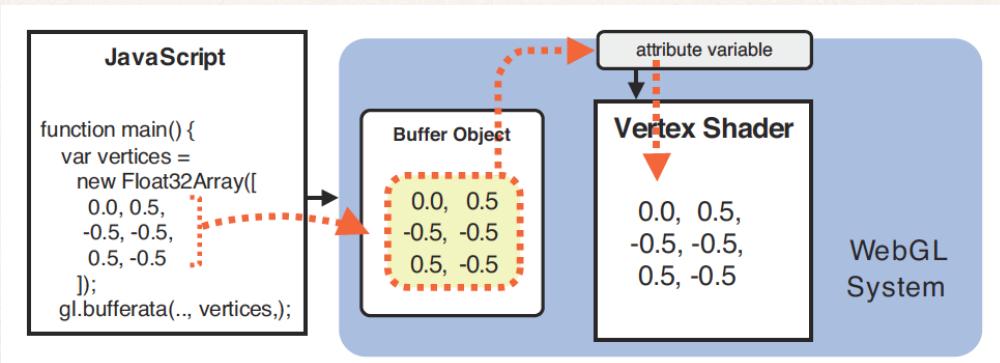
```
vertices
static const Vertex3D vertices[] = {
    {0, -0.525731, 0.850651},
    {0.850651, 0, 0.525731},
    {0.850651, 0, -0.525731},
    {-0.850651, 0, -0.525731},
    {-0.850651, 0, 0.525731},
    {-0.525731, 0.850651, 0},
    {0.525731, 0.850651, 0},
    {0.525731, -0.850651, 0},
    {-0.525731, -0.850651, 0},
    {0, -0.525731, -0.850651},
    {0, 0.525731, -0.850651},
    {0, 0.525731, 0.850651}
};
```

```
normals
static const Vector3D normals[] = {
    {0.0, 0.0, -0.417775, 0.675974},
    {0.675973, 0.0, 0.0, 0.417775},
    {0.675973, -0.0, 0.0, -0.417775},
    {-0.675973, 0.0, 0.0, -0.417775},
    {-0.675973, -0.0, 0.0, 0.417775},
    {-0.417775, 0.675974, 0.0, 0.0},
    {0.417775, 0.675974, -0.0, 0.0},
    {0.417775, -0.675974, 0.0, 0.0},
    {-0.417775, -0.675974, 0.0, 0.0},
    {0.0, 0.0, -0.417775, -0.675974},
    {0.0, 0.0, 0.417775, -0.675974},
    {0.0, 0.0, 0.417775, 0.675974}
};
```

```
colors
static const Color3D colors[] = {
    {1.0, 0.0, 0.0, 1.0},
    {1.0, 0.5, 0.0, 1.0},
    {1.0, 1.0, 0.0, 1.0},
    {0.5, 1.0, 0.0, 1.0},
    {0.0, 1.0, 0.0, 1.0},
    {0.0, 0.5, 1.0, 1.0},
    {0.0, 0.0, 1.0, 1.0},
    {0.5, 0.0, 1.0, 1.0},
    {1.0, 0.0, 1.0, 1.0},
    {1.0, 0.0, 0.5, 1.0}
};
```

Renderização em WebGL

- *Vertex Arrays / Buffer Objects*
 - Armazenados no espaço de endereçamento da GPU
 - Transferencia de dados da CPU para a GPU
 - Mapeados nos atributos do *vertex shader*



Renderização em WebGL

- No carregamento da página:

```
function webGLStart() {  
    var canvas = document.getElementById("triangulo");  
    var gl = initGL(canvas);  
    var shader = initShaders("shader", gl);  
    if (shader == null) {  
        alert("Erro na inicialização do shader!!");  
        return (null);  
    }  
    shader.vPosAttr      = gl.getAttribLocation(shaderProgram, "aVertexPosition");  
    if (shader.vPosAttr < 0) {  
        alert("Shader: atributo não localizado!");  
        return;  
    }  
    initBuffers(gl);  
    drawScene(gl, shader);  
}  
}
```

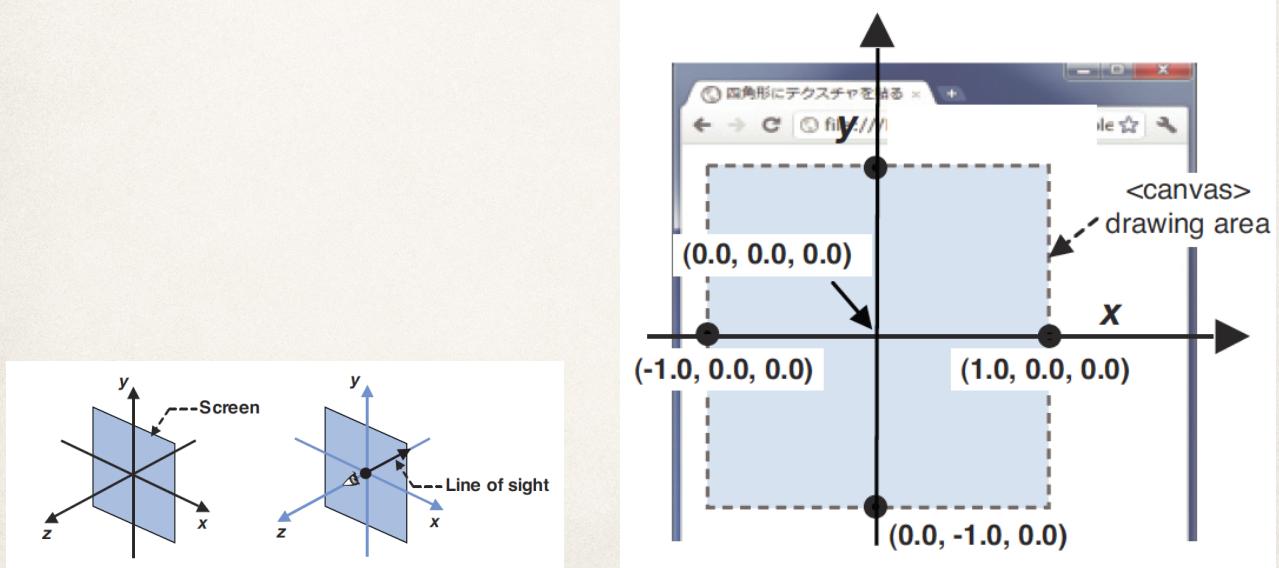
Primeiro desenho em WebGL

- Para inicializar os *buffers objects*:

```
var vPosBuf;  
// *****  
// *****  
function initBuffers(gl) {  
var vPos = [ -0.5, -0.5, 0.0,  
            0.5, -0.5, 0.0,  
            0.0, 0.5, 0.0  
        ];  
  
vPosBuf = gl.createBuffer();  
gl.bindBuffer(gl.ARRAY_BUFFER, vPosBuf);  
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vPos), gl.STATIC_DRAW);  
vPosBuf.itemSize = 3;  
vPosBuf.numItems = 3;  
}
```

Renderização em WebGL

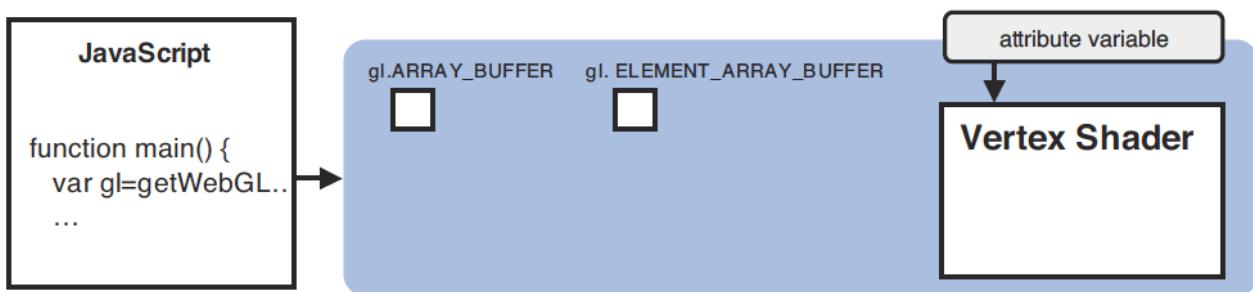
- Sistema de coordenadas *default* do contexto WebGL



Construindo um *Buffer Object*

- Obtendo o contexto WebGL

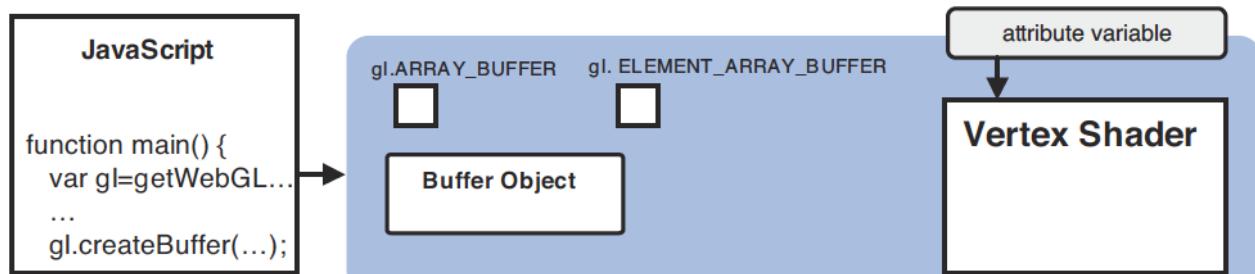
```
gl = canvas.getContext("webgl");
```



Construindo um *Buffer Object*

- Criando um *Buffer Object*

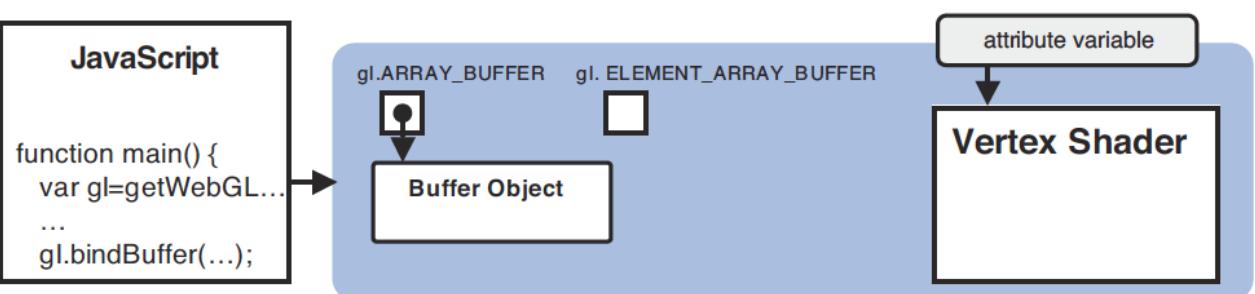
```
vPosBuf = gl.createBuffer();
```



Construindo um *Buffer Object*

- Associando um *Buffer Object* ao contexto WebGL

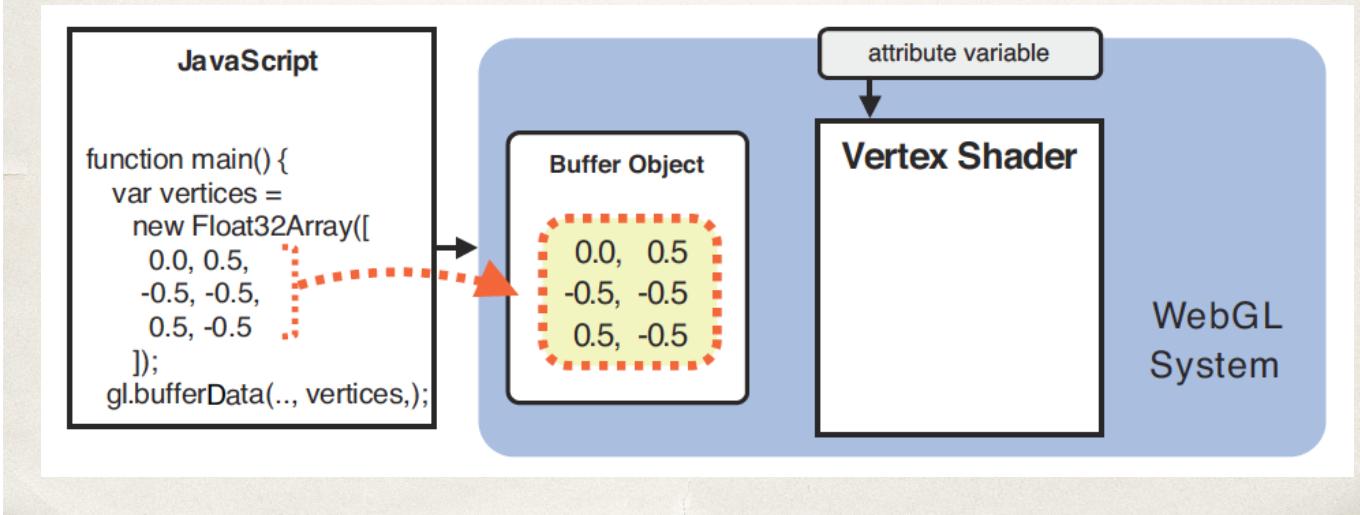
```
gl.bindBuffer(gl.ARRAY_BUFFER, vPosBuf);
```



Construindo um *Buffer Object*

- Transferindo os dados para o *Buffer Object*

```
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vPos), gl.STATIC_DRAW);
```



Renderizando o conteúdo do *Buffer Object*

- Para desenhar utilizando os *buffers objects*:

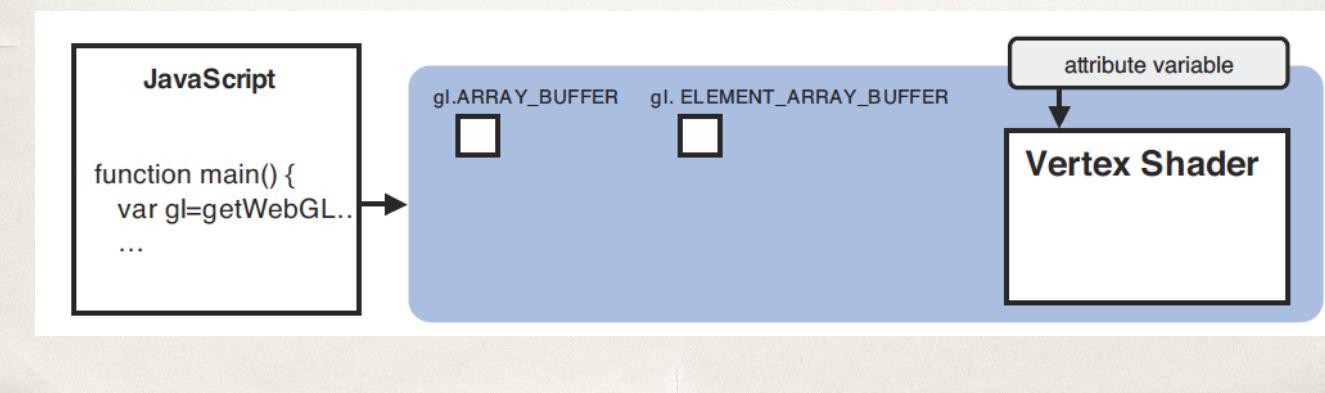
```
// ****
// ****
function drawScene(gl, shader) {
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT);

    gl.useProgram(shader);
    gl.bindBuffer(gl.ARRAY_BUFFER, vPosBuf);
    gl.enableVertexAttribArray(shader.vPosAttr);
    gl.vertexAttribPointer(shader.vPosAttr, vPosBuf.itemSize, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, vPosBuf.numItems);
    gl.disableVertexAttribArray(shader.vPosAttr);
    gl.useProgram(null);
}
```

Renderizando o conteúdo do *Buffer Object*

- "Instalando" o *shader* para renderização

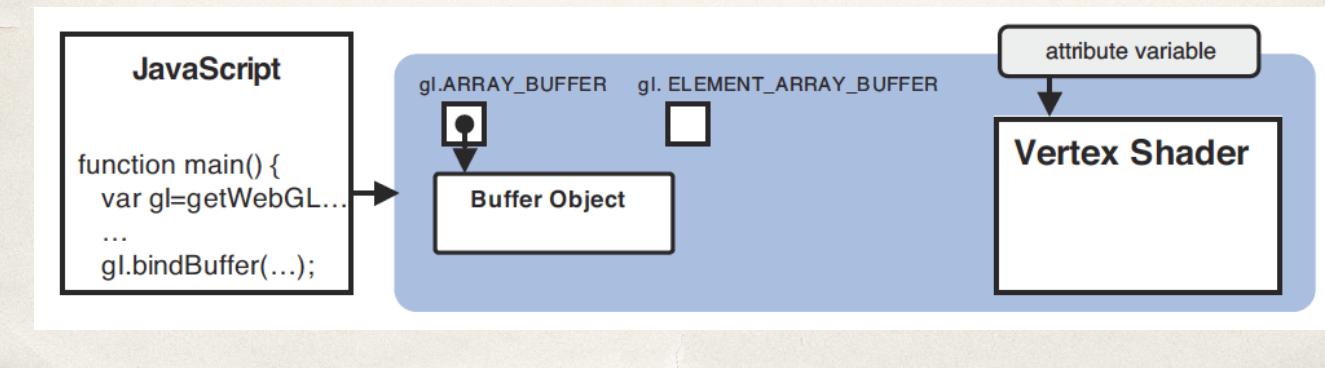
```
gl.useProgram(shader);
```



Renderizando o conteúdo do *Buffer Object*

- Associando o *Buffer Object* para renderização

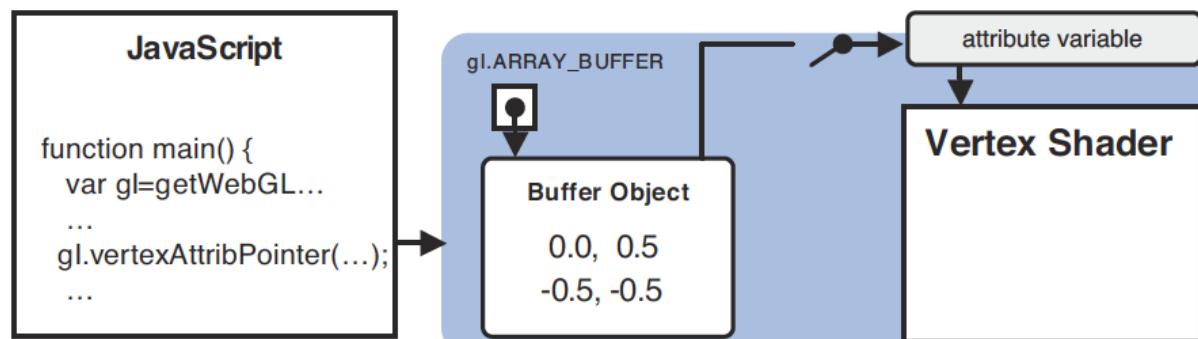
```
gl.bindBuffer(gl.ARRAY_BUFFER, vPosBuf);
```



Renderizando o conteúdo do *Buffer Object*

- Associando o *Buffer Object* corrente ao atributo do *shader*

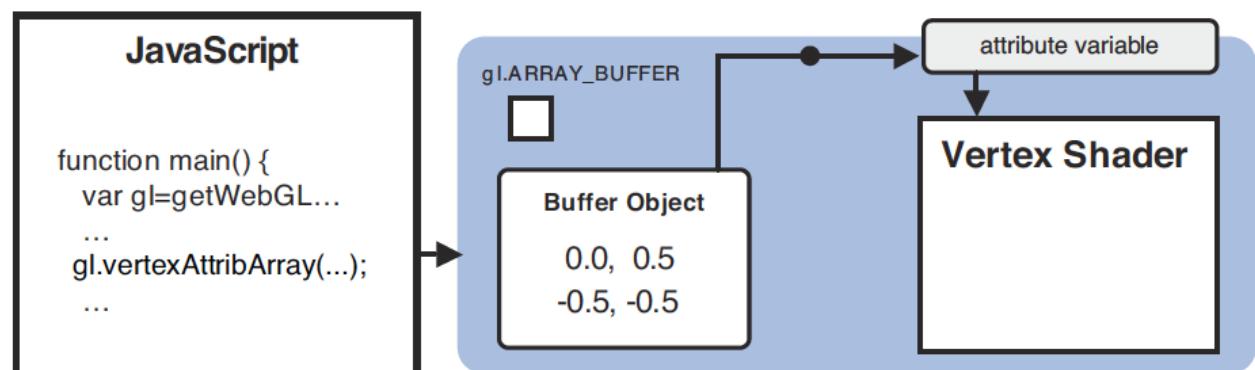
```
gl.vertexAttribPointer(shader.vPosAttr, vPosBuf.itemSize,  
                      gl.FLOAT, false, 0, 0);
```



Renderizando o conteúdo do *Buffer Object*

- Habilitando o atributo do *shader* e o *buffer object*

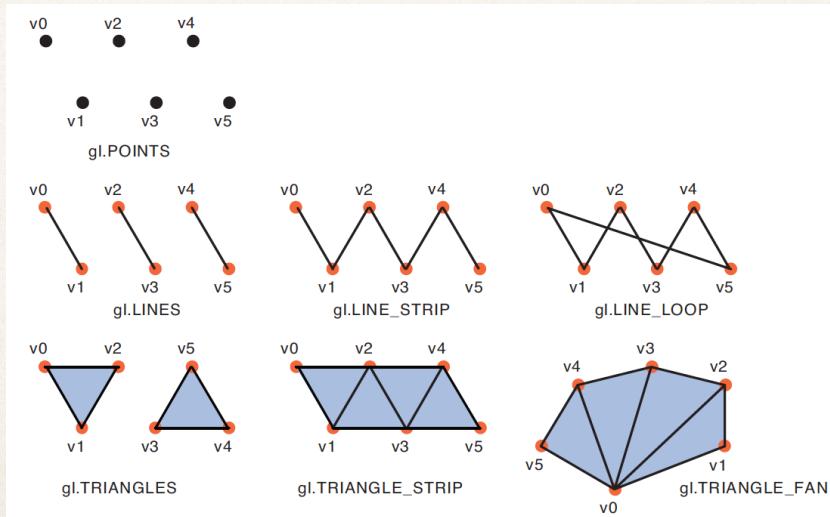
```
gl.enableVertexAttribArray(shader.vPosAttr);
```



Renderizando o conteúdo do Buffer Object

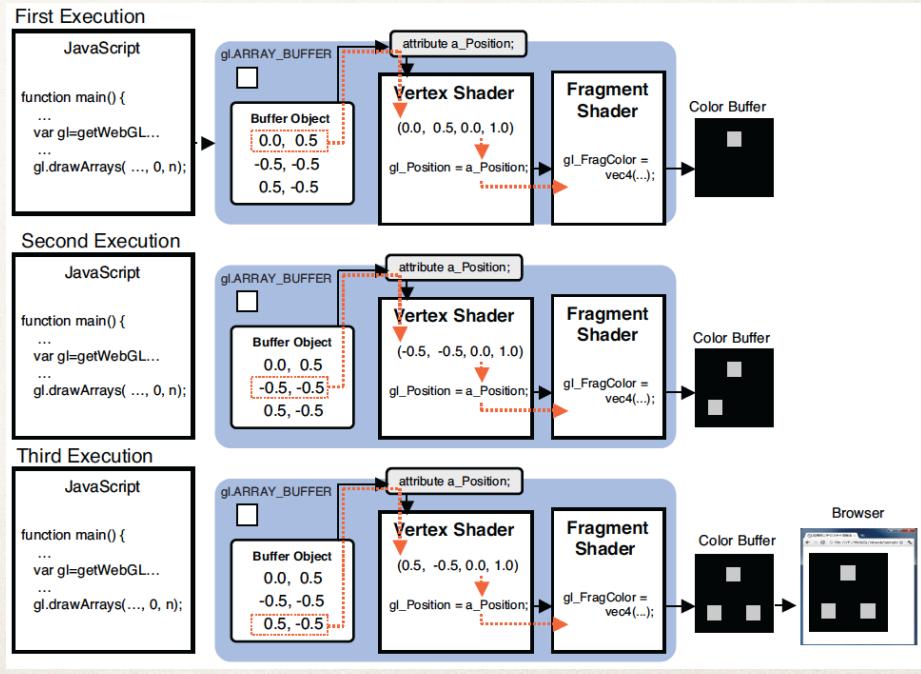
- Desenhandando o conteúdo do *Buffer Object* corrente

```
gl.drawArrays(gl.TRIANGLES, 0, vPosBuf.numItems);
```



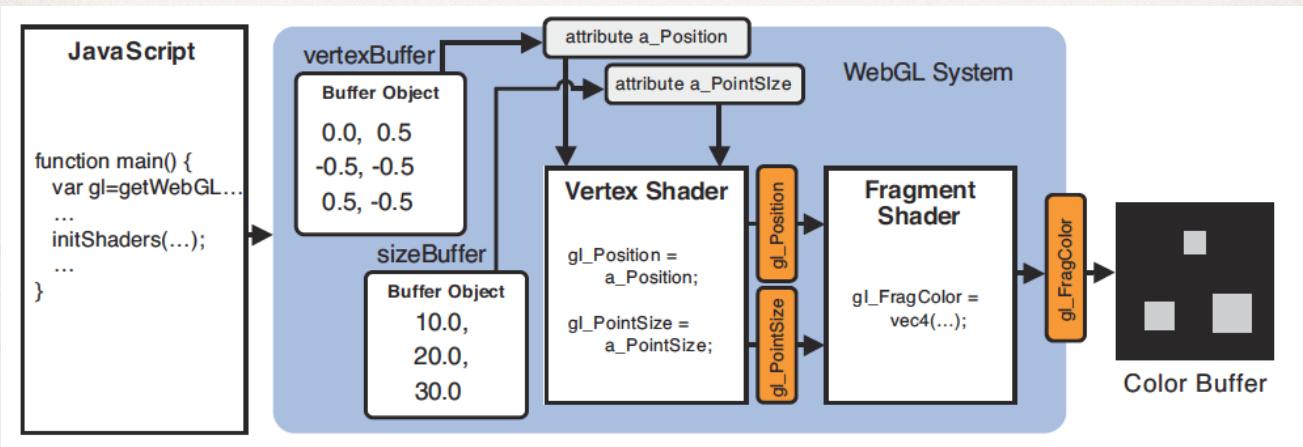
Renderizando o conteúdo do Buffer Object

- O processo geral:



Mais de um *Buffer Object*

- Adicionar cor aos vértices:
 - 2 *Buffer Objects*
 - 2 *attributes* no *shader*



Utilizando cores no desenho

- Criando os *buffers*:

```
var vPosBuf;  
var vColorBuf;  
  
function initBuffers(gl) {  
    var vPos = [-0.5, -0.5, 0.0,  
               0.5, -0.5, 0.0,  
               0.5, 0.5, 0.0,  
              -0.5, -0.5, 0.0,  
               0.5, 0.5, 0.0,  
              -0.5, 0.5, 0.0];  
  
    var vColor = [1.0, 0.0, 0.0,  
                 0.0, 1.0, 0.0,  
                 0.0, 0.0, 1.0,  
                 1.0, 0.0, 0.0,  
                 0.0, 0.0, 1.0,  
                 1.0, 1.0, 1.0];
```

```
vPosBuf = gl.createBuffer();  
gl.bindBuffer(gl.ARRAY_BUFFER, vPosBuf);  
gl.bufferData(gl.ARRAY_BUFFER,  
             new Float32Array(vPos),  
             gl.STATIC_DRAW);  
vPosBuf.itemSize = 3;  
vPosBuf.numItems = 6;  
  
vColorBuf = gl.createBuffer();  
gl.bindBuffer(gl.ARRAY_BUFFER, vColorBuf);  
gl.bufferData(gl.ARRAY_BUFFER,  
             new Float32Array(vColor),  
             gl.STATIC_DRAW);  
vColorBuf.itemSize = 3;  
vColorBuf.numItems = 6;
```

Utilizando cores no desenho

- ❖ Desenhandoo:

```
function drawScene(gl, shader) {  
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);  
    gl.clear(gl.COLOR_BUFFER_BIT);  
  
    gl.useProgram(shader);  
  
    gl.bindBuffer(gl.ARRAY_BUFFER, vPosBuf);  
    gl.enableVertexAttribArray(shader.vPosAttr);  
    gl.vertexAttribPointer(shader.vPosAttr, vPosBuf.itemSize, gl.FLOAT, false, 0, 0);  
  
    gl.bindBuffer(gl.ARRAY_BUFFER, vColorBuf);  
    gl.enableVertexAttribArray(shader.vColorAttr);  
    gl.vertexAttribPointer(shader.vColorAttr, vColorBuf.itemSize, gl.FLOAT, false, 0, 0);  
  
    gl.drawArrays(gl.TRIANGLES, 0, vPosBuf.numItems);  
  
    gl.disableVertexAttribArray(shader.vPosAttr);  
    gl.disableVertexAttribArray(shader.vColorAttr);  
    gl.useProgram(null);  
}
```

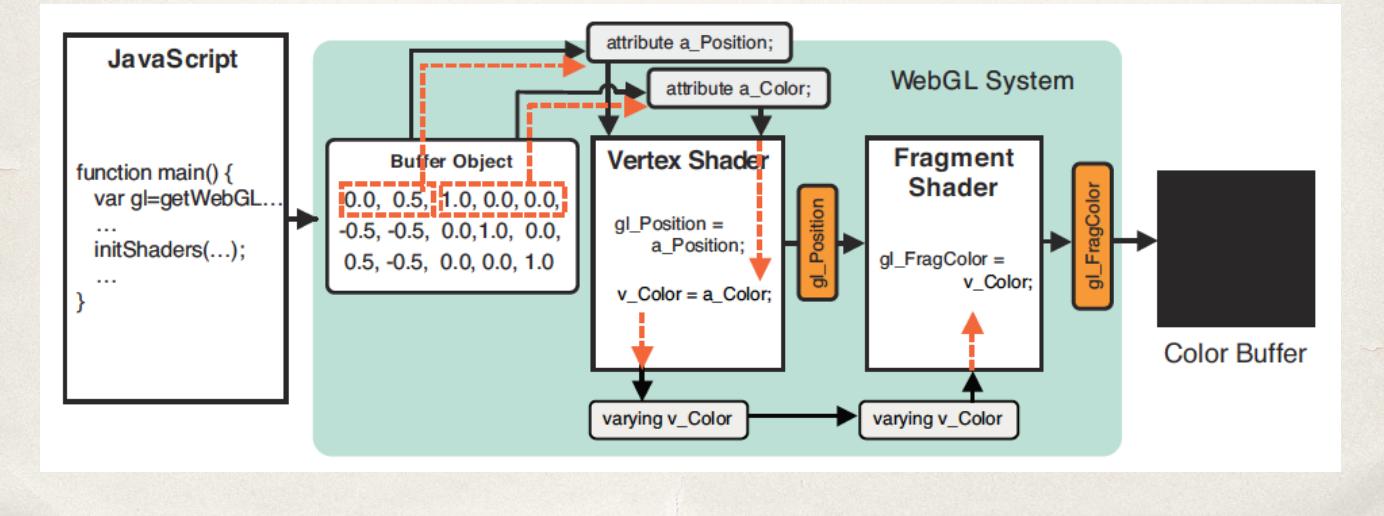
Passando parâmetros entre *Shaders*

- ❖ *Shaders*:

```
<script id="shader-vs" type="x-shader/x-vertex">  
    attribute vec3 aVertexPosition;  
    attribute vec3 aVertexColor;  
    varying vec3 vColor;  
    void main(void) {  
        gl_Position = vec4(aVertexPosition, 1.0);  
        vColor = aVertexColor;  
    }  
</script>  
<script id="shader-fs" type="x-shader/x-fragment">  
    precision mediump float;  
    varying vec3 vColor;  
    void main(void) {  
        gl_FragColor = vec4(vColor, 1.0);  
    }  
</script>
```

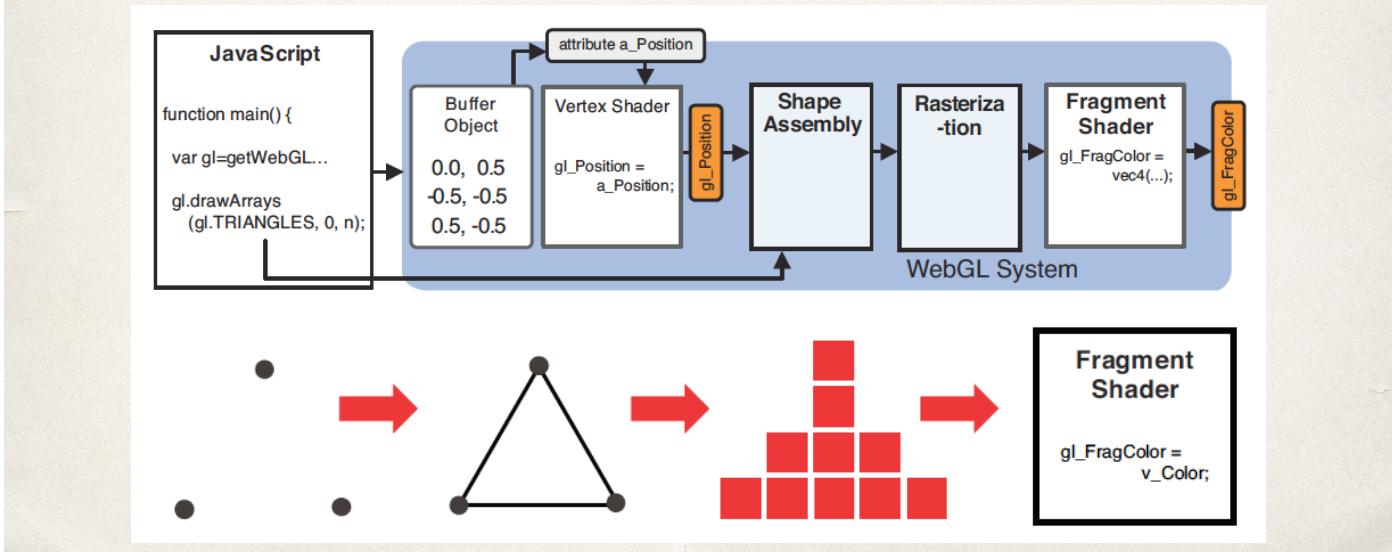
Passando parâmetros entre Shaders

- Cores dos vértices devem ser transmitidas para os fragmentos
 - variáveis *varying*
 - Mesmo nome



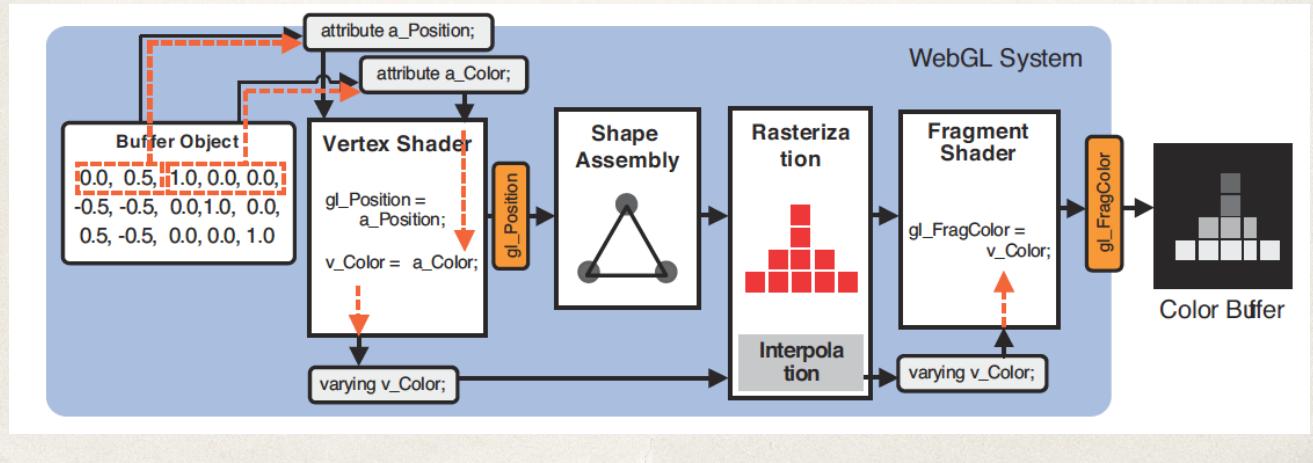
Passando parâmetros entre Shaders

- variáveis *varying*
 - Valores enviados pelo *vertex shader* são interpolados para o *fragment shader*



Passando parâmetros entre Shaders

- variáveis *varying*
- Valores enviados pelo *vertex shader* são interpolados para o *fragment shader*



A Seguir...

LAB2: Processamento de Imagem