CSE4204
LAB-1 : Intro to WebGL

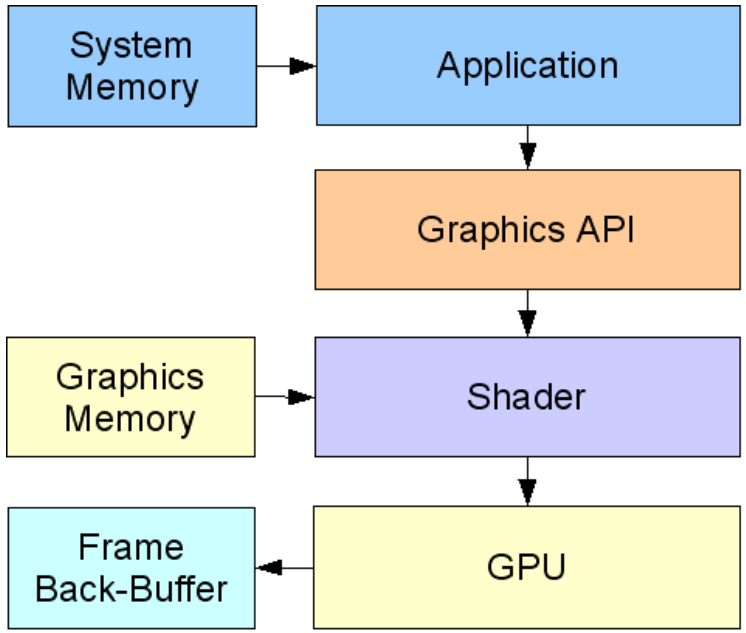
Mohammad Imrul Jubair

Graphics API

The content we see on screen is taken from Frame buffer. Frame buffer has the data. Application is the like java swing stuff. It calls an API. API has everything defined of how to draw things. Shader language is the part which runs in GPU. As GPU is itself an image which requires huge amount of computation, so we need a separate processing unit. The Graphics API connects the CPU and GPU.

- Example: OpenGL, WebGL, Direct3D, etc.

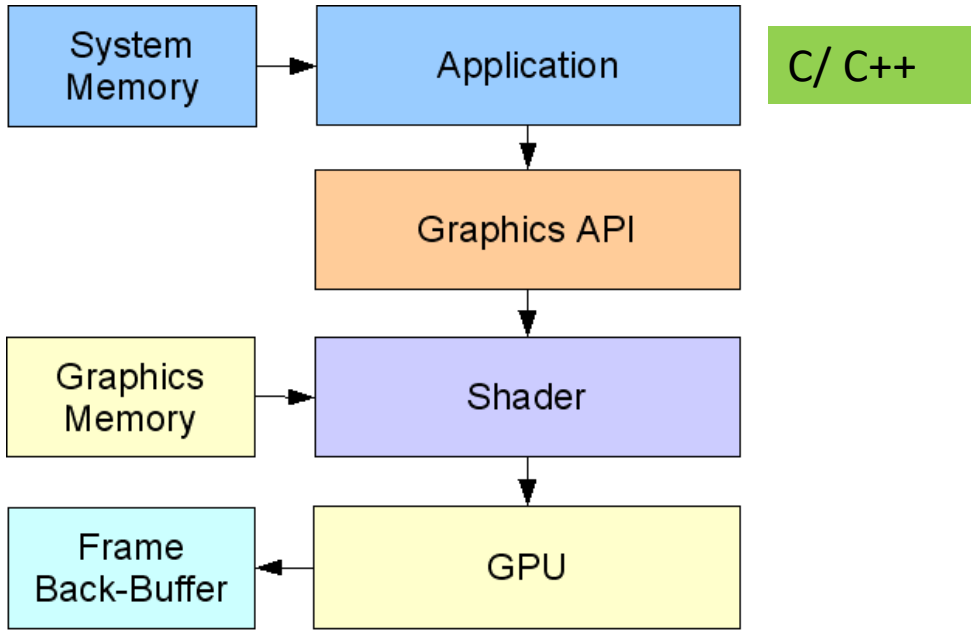
webgl is cross platform, it can run on any library.

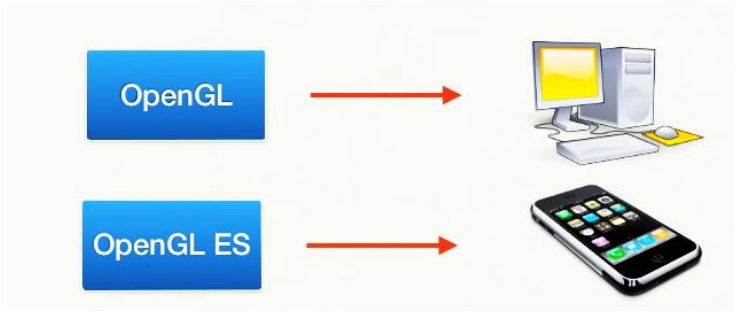


Source: https://ict.senecacollege.ca/~chris.szalwinski/archives/gam670.071/content/shadr_p.html

OpenGL

- OpenGL and OpenGL ES

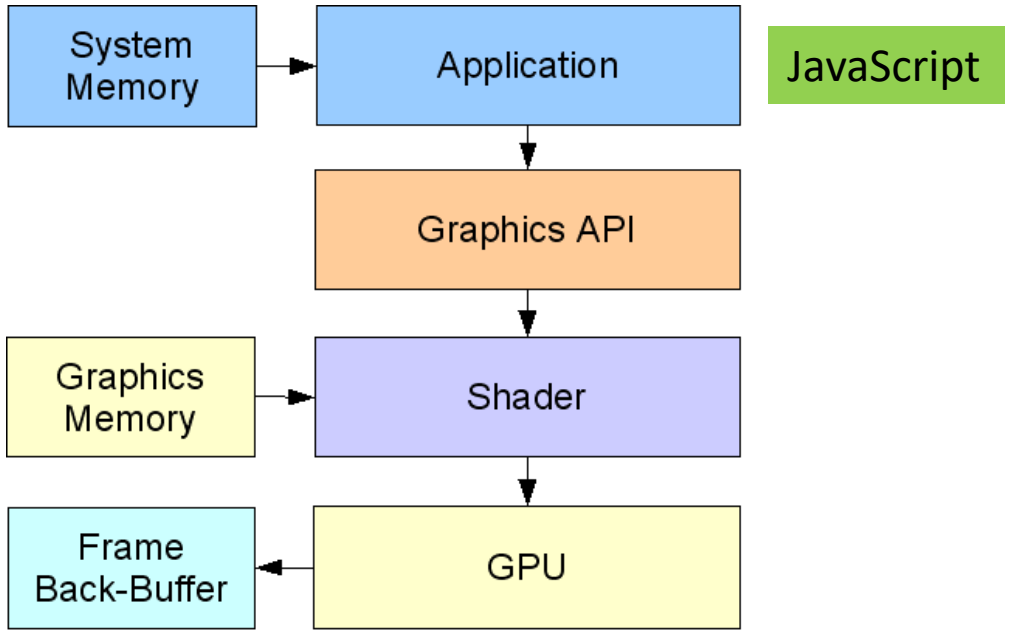




Source: https://ict.senecacollege.ca/~chris.szalwinski/archives/gam670.071/content/shadr_p.html

WebGL

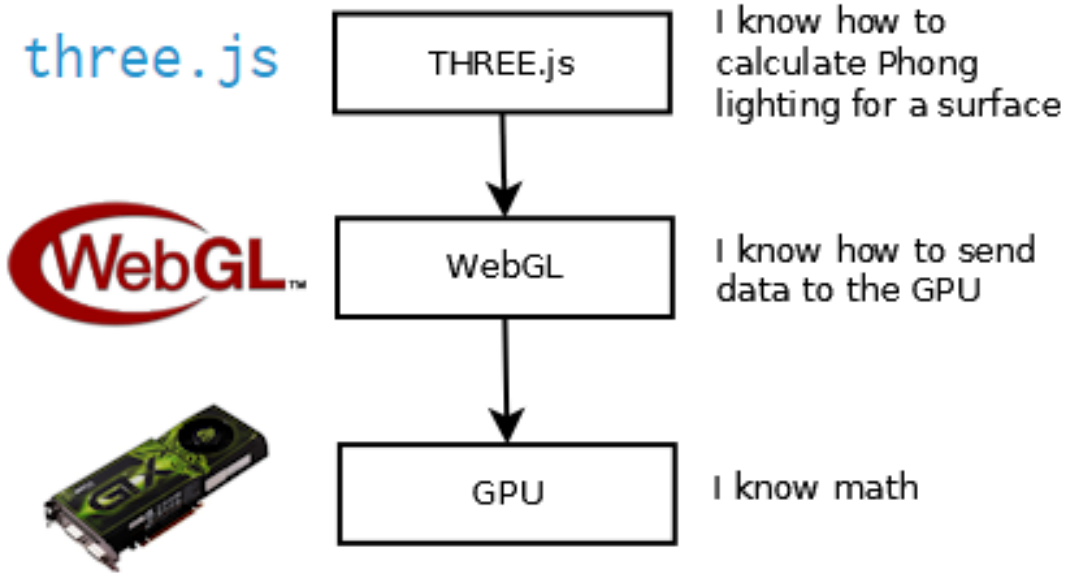
- a JavaScript interface for OpenGL-ES-2.x API, promoted by Khronos.



Source: https://ict.senecacollege.ca/~chris.szalwinski/archives/gam670.071/content/shadr_p.html

WebGL

Three js is easier.



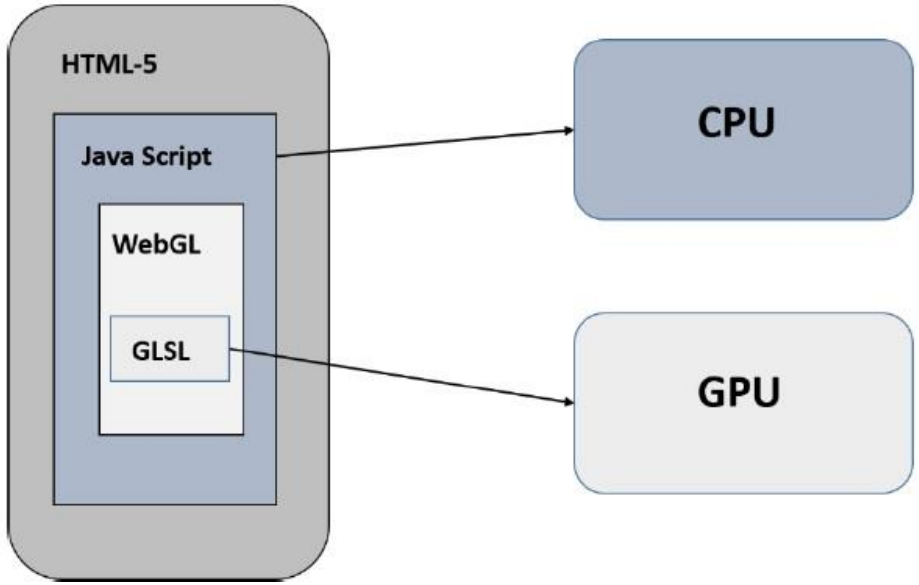
Source: https://cglearn.codelight.eu/pub/computer-graphics/computer-graphics

A WebGL Program

Webgl program has two parts.
1st - javascript, the html file is the canvas
2nd - inside the canvas, we do webgl stuffs.

java script part is handled by cpu, webgl connects javascript and GLSL, in which the shader program is defined. It does the graphics work.

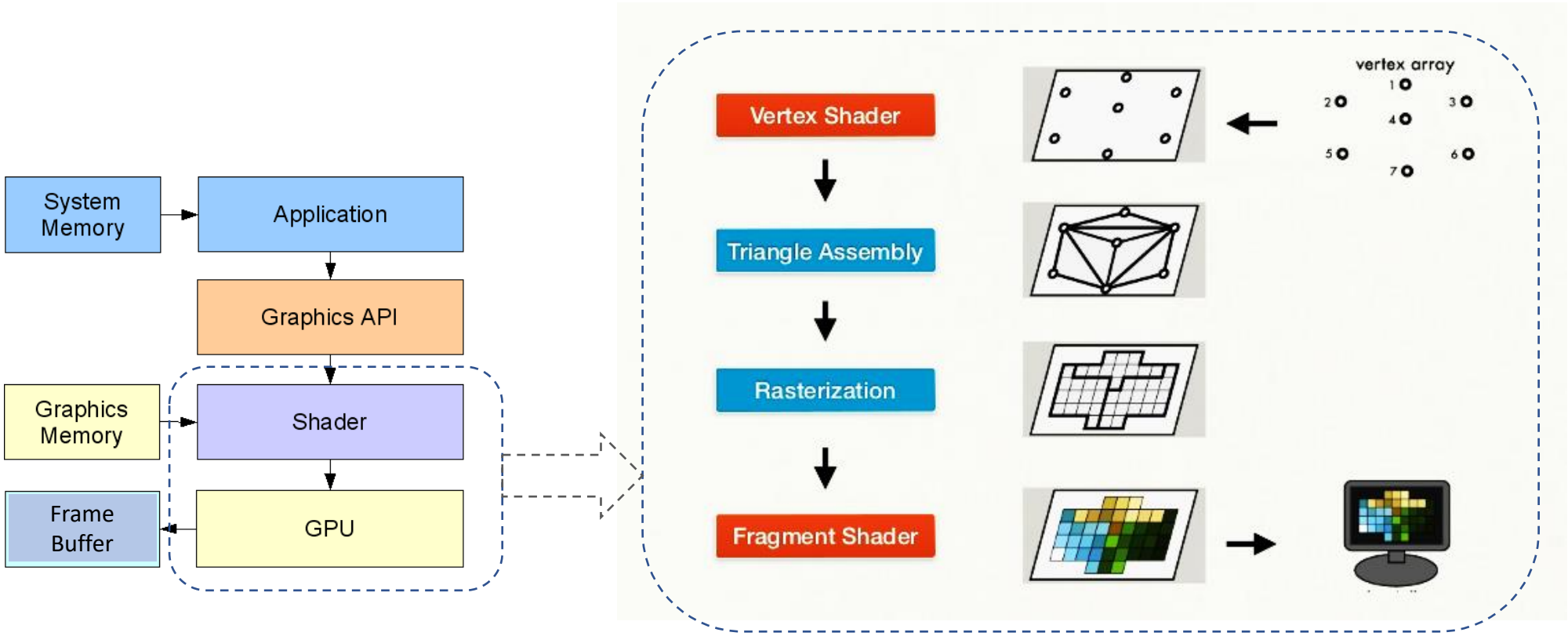
- There are two sides to any WebGL program:
  - Part – 1: written in JavaScript
  - Part – 2: written in GLSL, a language for writing "shader" programs that run on the GPU.



Source: http://math.hws.edu/graphicsbook
Source: https://www.tutorialspoint.com/webgl/webgl_quick_guide.htm

A Graphics Pipeline

It is a pipe, o/p of one goes to i/p of another.
Say At first 7 vertex is defined in cpu, as we can't direclty talk to gpu. GPU sends this to the shader programs (vertex and fragment).
Vertex and Fragment Shader are programmable, but triangle assembly and rasterization is not programmable.
Rasterization creates the grid to be displayed on the screen. We donot need to code for that.
Fragment shader colors each of the pixels.

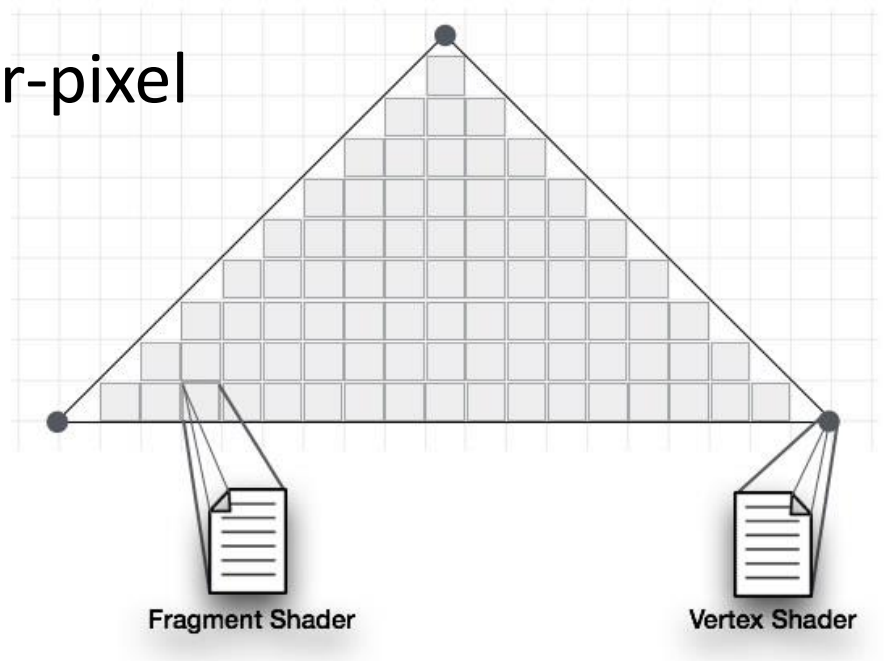


Source: https://pt.slideshare.net/senchainc/webgl-fundamentals-10013633/12

Shaders

Vertex shader is per vertex, there can be many codes, we will write code once, but it will run for all vertex points.
Similarly for fragment shader, we will write code for 1 pixel, but it will apply those for all pixels.

- Vertex Shader: Per-vertex
- Fragment Shader: Per-fragment/ per-pixel



Source: https://www.tutorialspoint.com/webgl/webgl_quick_guide.htm

A Shader Program

References:

- https://www.tutorialspoint.com/webgl/webgl_drawing_points.htm
- http://math.hws.edu/graphicsbook/index.html

Steps*

Get the code: https://rb.gy/cpf3uo

- Step 1 – Prepare the canvas and get WebGL rendering context
- Step 2 – Create and compile Shader programs
- Step 3 – Associate the shader programs with buffer objects
- Step 4 – Define the geometry and store it in buffer objects
- Step 5 – Drawing the required object

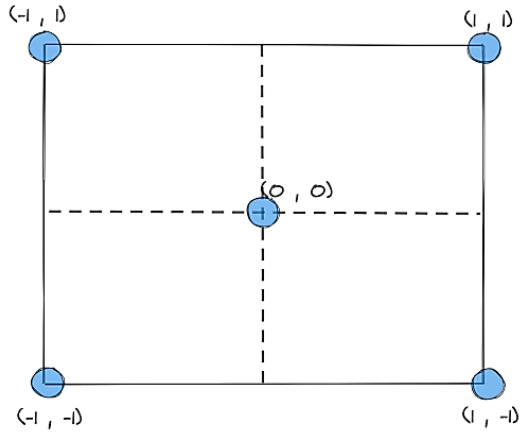
```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
   `attribute vec3 a_coords;
    void main() {
        gl_Position = vec4(a_coords, 1.0);
    }`;


var fragmentShaderSource =
   `void main() {
        gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
    }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Modified from the source: https://www.tutorialspoint.com/webgl/webgl_drawing_points.htm

*Can be altered

Step – 1: Canvas and WebGL rendering context

```
<canvas
id="webglcanvas" width="500" height="500">
</canvas>

var canvas = document.getElementById("webglcanvas");

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



Through gl, we can use many built in functions of webgl. It is like importing package through which we can call functions. The range of values from -1 to 1. If we want to draw somewhere 1.5,1.5, it is not possible.

```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
   `attribute vec3 a_coords;
    void main() {
        gl_Position = vec4(a_coords, 1.0);
    }`;


var fragmentShaderSource =
   `void main() {
        gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
    }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Step – 1: Background and reset buffer

```
gl.clearColor(0.75, 0.75, 0.75, 1.0);

gl.clear(gl.COLOR_BUFFER_BIT);
```

clearColor sets background color. The range is from 0 to 1. The last parameter is transparency.

gl.clear(..) -> it clears the buffer values. it refreshes the buffer values from which we will show content in the screen.

```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
    `attribute vec3 a_coords;
     void main() {
         gl_Position = vec4(a_coords, 1.0);
     }`;


var fragmentShaderSource =
    `void main() {
         gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
     }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

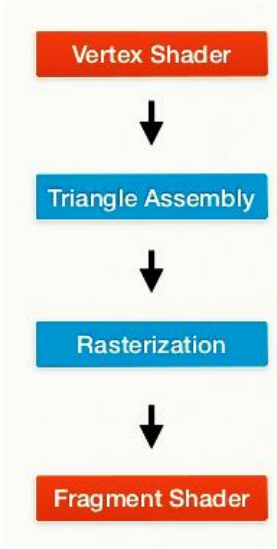
Step – 2: Vertex Shader

Vertex shadersource is a code which is a shader language, GLSL. This is a separate language. We will send this to graphics card. Everything in part of a tilde is a string. It is a C like code. It has main funciton.
attribute vec3 a_coords is a 3 value vector variable.
Vertex shader takes all vertex information that is the traingle we will draw. It is per vertex code. The last value is because of homogenous coordinate system. gl_Position is claled an output bucket. it shows o/p.

```
var vertexShaderSource =

`attribute vec3 a_coords;
 void main() {
     gl_Position = vec4(a_coords, 1.0);
         }`;
```

vertices

Clipping



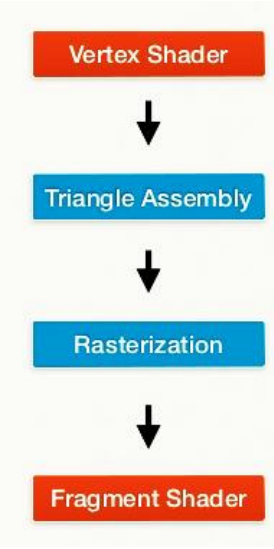

```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
    `attribute vec3 a_coords;
     void main() {
         gl_Position = vec4(a_coords, 1.0);
     }`;


var fragmentShaderSource =
    `void main() {
         gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
     }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Step – 2: Vertex Shader

attribute is Per vertex. For attribute each vertex coordinate and color is different.
For uniform, coordinate and color remains same. Trainalge assembly and Rasteriazation doesn't have to be computed by us.

```
var vertexShaderSource =

`attribute vec3 a_coords;
 void main() {
     gl_Position = vec4(a_coords, 1.0);
             }`;
```

Per-vertex



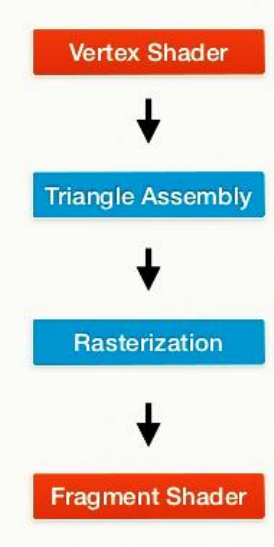

```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
   `attribute vec3 a_coords;
    void main() {
        gl_Position = vec4(a_coords, 1.0);
    }`;


var fragmentShaderSource =
   `void main() {
        gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
    }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Step – 2: Fragment Shader

what vertex we are providing, it will color the vertexes with this colors. parameter is rgb alpha.
gl_FragColor is case sensitive.

```
var fragmentShaderSource =

`void main() {
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
                }`;
```

Colored pixels




```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
   `attribute vec3 a_coords;
    void main() {
        gl_Position = vec4(a_coords, 1.0);
    }`;


var fragmentShaderSource =
   `void main() {
        gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
    }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Step – 2: Create and Compile Shaders

the shader codes are compiled. Cpu gives the compilation instruction.

```
var vsh = gl.createShader( gl.VERTEX_SHADER );

gl.shaderSource( vsh, vertexShaderSource );

gl.compileShader( vsh );
```

```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
    `attribute vec3 a_coords;
     void main() {
         gl_Position = vec4(a_coords, 1.0);
     }`;


var fragmentShaderSource =
    `void main() {
         gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
     }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Step – 2: Create and Compile Shaders

```
var fsh = gl.createShader( gl.FRAGMENT_SHADER );

gl.shaderSource( fsh, fragmentShaderSource );

gl.compileShader( fsh );
```

```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
    `attribute vec3 a_coords;
     void main() {
         gl_Position = vec4(a_coords, 1.0);
     }`;


var fragmentShaderSource =
    `void main() {
         gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
     }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Step – 2: Link shaders and use program

Link program is like telling gpu this program is runnable.
use program is similar, it tells gpu is programmable.

```
var prog = gl.createProgram();

gl.attachShader( prog, vsh );

gl.attachShader( prog, fsh );

gl.linkProgram( prog );

gl.useProgram(prog);
```

```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
    `attribute vec3 a_coords;
     void main() {
         gl_Position = vec4(a_coords, 1.0);
     }`;


var fragmentShaderSource =
    `void main() {
         gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
     }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Step – 3: Associate Shaders

```
var a_coords_location =
        gl.getAttribLocation(prog, "a_coords");
```

```
attribute vec3 a_coords;
void main() {
        gl_Position = vec4(a_coords, 1.0);
        }
```

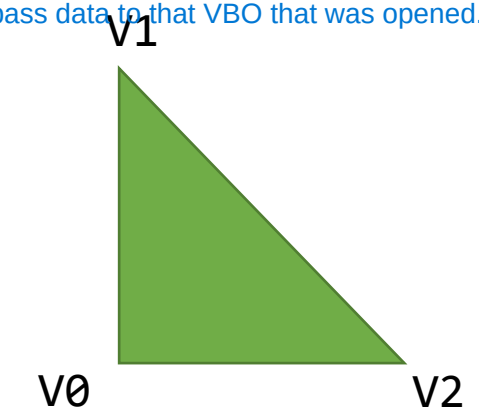
a_coords_location, here we are passing data to gpu, but cpu needs to be know this variable. For cpu to access this varialble, we use getAttribLocation. It gets the a_coords attribute and saves its location.
GPU -> a_coords
CPU -> a_coords_location
Now to pass values to the a_coords, vertex points.

```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
    `attribute vec3 a_coords;
     void main() {
         gl_Position = vec4(a_coords, 1.0);
     }`;


var fragmentShaderSource =
    `void main() {
         gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
     }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Step – 4: Define Geometry

```
var coords = new Float32Array( [0.0, 0.0, 0.0, \\V0
                                0.0, 0.5, 0.0, \\V1
                                0.5, 0.0, 0.0] \\V2
                             );
```

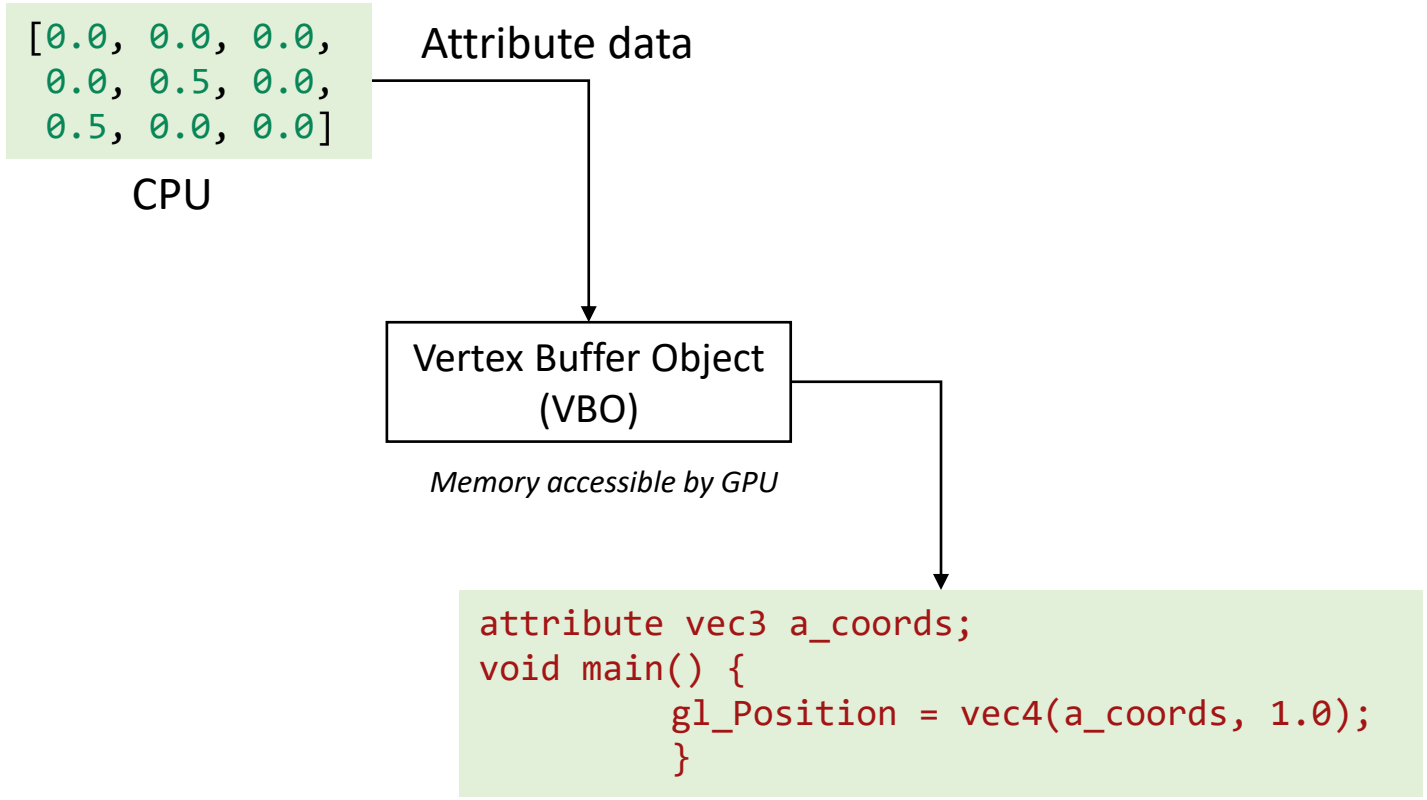
After defining the coordinate values in cpu, we need to send this data to gpu memory. GPU can take this data from that memory accessible by GPu. It is called vertex buffer object. VBO is memory accessible by gpu.
a_coords_buffer creates an empty VBO, this is done by createbuffer. THen we have to keep data that VBO, this is done by bind buffer. Now using bufferdata, we pass data to that VBO that was opened.




```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
    `attribute vec3 a_coords;
     void main() {
         gl_Position = vec4(a_coords, 1.0);
     }`;


var fragmentShaderSource =
    `void main() {
         gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
     }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Step – 4: Vertex Buffer Objects

```
var a_coords_buffer = gl.createBuffer();
```




```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
    `attribute vec3 a_coords;
     void main() {
         gl_Position = vec4(a_coords, 1.0);
     }`;


var fragmentShaderSource =
    `void main() {
         gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
     }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Step – 4: Vertex Buffer Objects (VBOs)

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

will be used for attribute

It specifies how the VBO will be used.

```
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
```

same data will be used

through this 3rd line, we will have to let the GPU know about how we plan to use the data that we are providing, we are considering it to be a 3d data, so 3 is passed. a_coords_location is cpu version of a_coords.

```
gl.vertexAttribPointer(a_coords_location, 3,
                       gl.FLOAT, false, 0, 0);
```

For 3D data (x,y,z)

```
gl.enableVertexAttribArray(a_coords_location);
```

Enable it means gpu can use it now.

```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
    `attribute vec3 a_coords;
     void main() {
         gl_Position = vec4(a_coords, 1.0);
     }`;


var fragmentShaderSource =
    `void main() {
         gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
     }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Step – 5: Draw Required Objects

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

primitives

Start vertex

Number of vertices

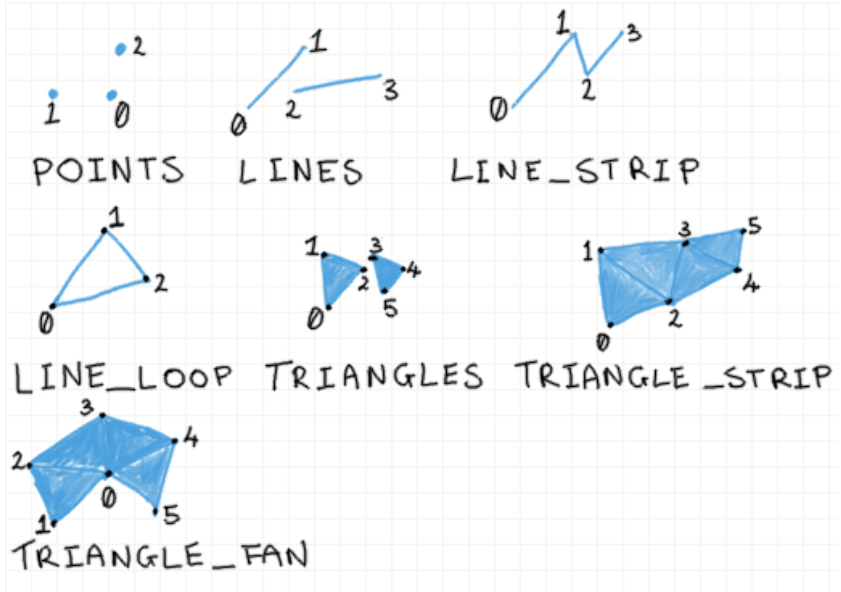
Gl_triangle is primitiive, how do we want to show the points. From there, we will start from drawing 0 vertexes, and then draw number of vertexes.

```
[0.0, 0.0, 0.0, → 0
 0.0, 0.5, 0.0, → 1
 0.5, 0.0, 0.0] → 3
```

```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
    `attribute vec3 a_coords;
     void main() {
         gl_Position = vec4(a_coords, 1.0);
     }`;


var fragmentShaderSource =
    `void main() {
         gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
     }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Primitives

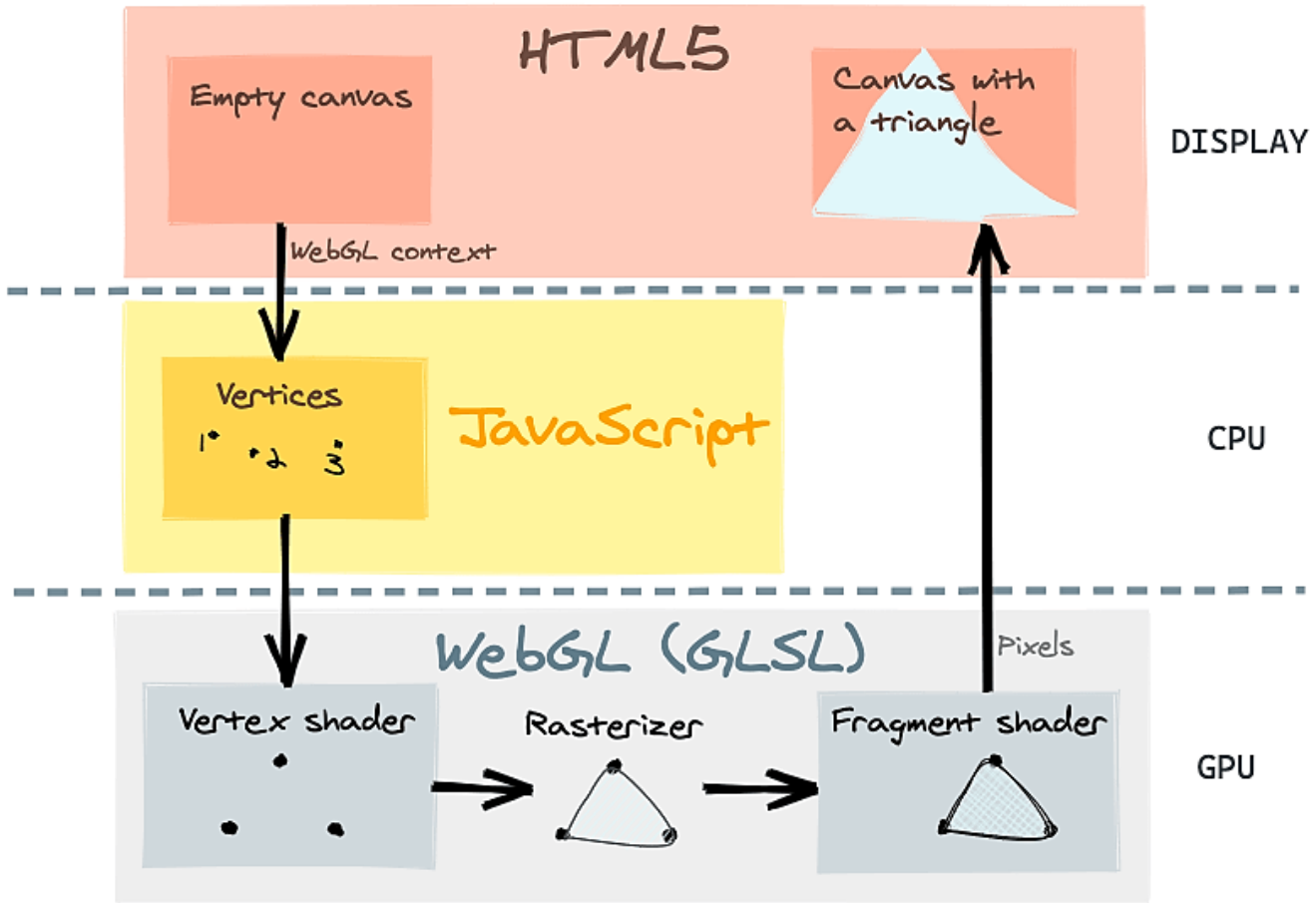
```
gl.drawArrays(gl.TRIANGLES, 0, 3);
```



Source: https://antongerdelan.net/opengl/vertexbuffers.html

```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
    `attribute vec3 a_coords;
     void main() {
         gl_Position = vec4(a_coords, 1.0);
     }`;


var fragmentShaderSource =
    `void main() {
         gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
     }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Full Code




```
var canvas = document.getElementById("webglcanvas");
var  gl = canvas.getContext("webgl");

gl.clearColor(0.75, 0.75, 0.75, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

var vertexShaderSource =
    `attribute vec3 a_coords;
     void main() {
         gl_Position = vec4(a_coords, 1.0);
     }`;


var fragmentShaderSource =
    `void main() {
         gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
     }`;

var vsh = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource( vsh, vertexShaderSource );
gl.compileShader( vsh );

var fsh = gl.createShader( gl.FRAGMENT_SHADER );
gl.shaderSource( fsh, fragmentShaderSource );
gl.compileShader( fsh );

var prog = gl.createProgram();

gl.attachShader( prog, vsh );
gl.attachShader( prog, fsh );
gl.linkProgram( prog );
gl.useProgram(prog);

var a_coords_location = gl.getAttribLocation(prog, "a_coords");

var coords = new Float32Array( [0.0, 0.0, 0.0,
                                0.0, 0.5, 0.0,
                                0.5, 0.0, 0.0] );

var a_coords_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, coords, gl.STATIC_DRAW);
gl.vertexAttribPointer(a_coords_location, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(a_coords_location);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```