Lab 1 – Introduction to Graphics API and Shading Language

1.1. Objective:

Objectives of this topics are – (1) to learn about graphics APIs, such as OpenGL/ WebGL; (2) to write a simple program to basic primitives with coordinates and colors; (3) to understand the basic concepts of shading language and its important parts: vertex and fragment shader; (4) to understand the basic graphics pipeline.

1.2. The Graphics API – WebGL:

OpenGL is a family of computer graphics APIs that is implemented in many graphics hardware devices. There are several versions of the API, and there are implementations, or "bindings" for several different programming languages. Versions of OpenGL for embedded systems such as mobile phones are known as OpenGL ES. a graphics API that was introduced in 1992 and has gone through many versions and many changes since then. WebGL is a version for use on Web pages. OpenGL can be used for 2D as well as for 3D graphics, but it is most commonly associated with 3D.

WebGL is A 3D graphics API for use on web pages. WebGL programs are written in the JavaScript programming language and display their images in HTML canvas elements. WebGL is based on OpenGL ES, the version of OpenGL for embedded systems, with a few changes to adapt it to the JavaScript language and the Web environment.

1.3. The Graphics Pipeline:

OpenGL 1.1 used a fixed-function pipeline for graphics processing. Data is provided by a program and passes through a series of processing stages that ultimately produce the pixel colors seen in the final image. The program can enable and disable some of the steps in the process, such as the depth test and lighting calculations. But there is no way for it to change what happens at each stage. The functionality is fixed.

OpenGL 2.0 introduced a programmable pipeline. It is a processing pipeline in which some of the processing stages can or must be implemented by programs. Data for an image passes through a sequence of processing stages, with the image as the end product. The sequence is called a "pipeline." Programmable pipelines are used in modern GPUs to provide more flexibility and control to the programmer. The programs for a programmable pipeline are known as shaders and are written in a shader programming language such as GLSL. It

became possible for the programmer to replace certain stages in the pipeline with their own programs. This gives the programmer complete control over what happens at that stage. In OpenGL 2.0, the programmability was optional; the complete fixed-function pipeline was still available for programs that didn't need the flexibility of programmability. WebGL uses a programmable pipeline, and it is mandatory. There is no way to use WebGL without writing programs to implement part of the graphics processing pipeline.

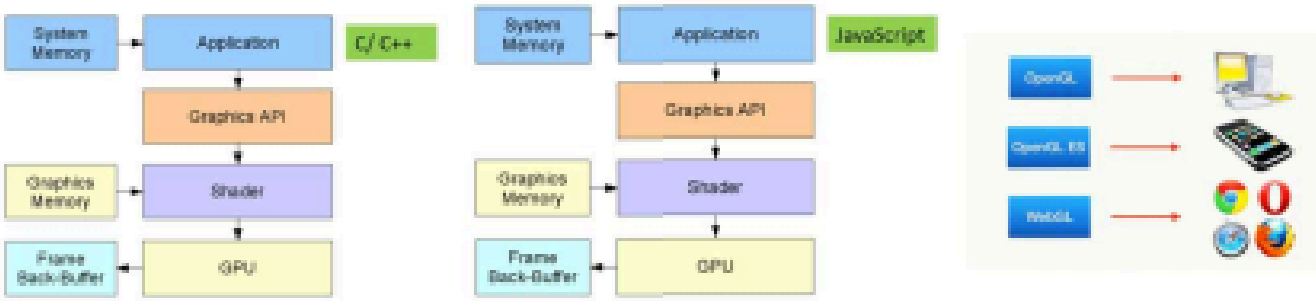


***Figure 1.1**: Source: Difference between OpenGL, OpenGL ES and WebGL. (source: https://ict.senecacollege.ca/~chris.szalwinski/archives/gam670.071/content/shadr_p.html)*

The programs that are written as part of the pipeline are called shaders. For WebGL, you need to write a vertex shader, which is called once for each vertex in a primitive, and a fragment shader, which is called once for each pixel in the primitive. Aside from these two programmable stages, the WebGL pipeline also contains several stages from the original fixed-function pipeline. For example, the depth test is still part of the fixed functionality, and it can be enabled or disabled in WebGL in the same way as in OpenGL 1.1.

1.3.1. Vertex Shader: A shader program that will be executed once for each vertex in a primitive. A vertex shader must compute the vertex coordinates in the clip coordinate system. It can also compute other properties, such as color.

1.3.2. Fragment Shader: A shader program that will be executed once for each pixel in a primitive. A fragment shader must compute a color for the pixel, or discard it. Fragment shaders are also called pixel shaders.

1.4. A Graphics Program:

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.

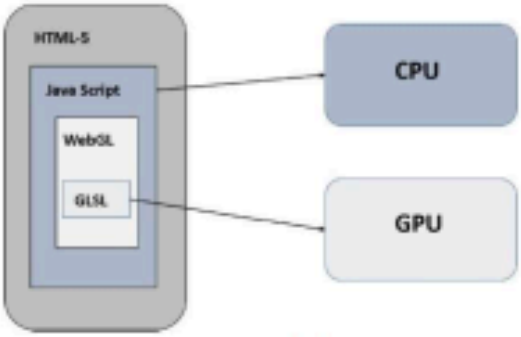


Figure 1.2: *Parts of a WebGL program. (source: https://www.tutorialspoint.com/webgl/webgl_quick_guide.htm)*

1.4.1. Standard steps for a WebGL program: A basic WebGL program follows the basic steps as listed below.

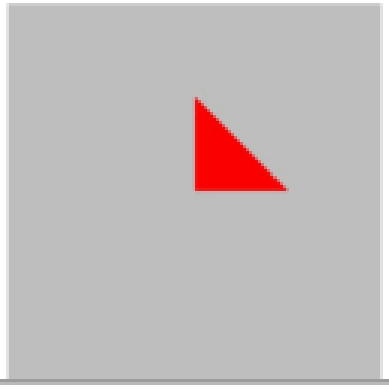
Step 1 – Prepare the canvas and get WebGL rendering context
Step 2 – Create and compile Shader programs
Step 3 – Associate the shader programs with buffer objects
Step 4 – Define the geometry and store it in buffer objects
Step 5 – Drawing the required object

Code: 1.1. Sample Code for drawing a triangle

```
<!-- saved from url=(0065)http://math.hws.edu/graphicsbook/source/webgl/simple-texture.html --
><!-- modified by Mohammad Imrul Jubair -->

<html>
<title>LAB-1: Intro</title>
<canvas id="webglcanvas" width="500" height="500"></canvas>

<script>

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

    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.clearColor(0.75, 0.75, 0.75, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);
    gl.drawArrays(gl.TRIANGLES, 0, 3);

</script></html>
```

Output:



1.5. Task:

a) Write a program to draw quad using two triangles.
b) Write a program that utilizes different drawing primitives to draw a multiple triangle.