**Fog Explanation**

onload="main()"

This **onload** event occurs when the **main()** object has been loaded. The **onload** is used within the **<body>** tag element to execute a webgl shaders, when web page has been completely loaded all content from the **main()** function.

var VSHADER\_SOURCE =

**VSHADER\_SOURCE** is a variable declared to store the vertex shader program.

'attribute vec4 a\_Position;\n' +

**Attribute** used **vec4** as type, so we can clip vertices into our buffer. The **a\_Position** is defined as the vertex position attribute. This is the first step of algorithm for creating object for vertex.This is also one of the key ideas of fog algorithm.

'attribute vec4 a\_Color;\n' +

The **a\_Color** is defined as the vertex color attribute. We will apply color to the object by assigning it in the vertex shader, along with the coordinates of the **attribute.** In the fog algorithm **attribute vec4 a\_Color** is a key idea and directly related to fog.

'uniform mat4 u\_MvpMatrix;\n' +

This is a constant which represent the combined projection matrix, view and model, which is known as the **Mvp**. To get the uniform variable for the model matrix, also this **uniform mat4 u\_MvpMatrix** is not directly related to fog because, it is not the key idea of the algorithm in the fog.

'uniform mat4 u\_ModelMatrix;\n' +

This is also a constant which represent the model matrix. This also directly related to the fog key idea of the algorithm.

'uniform vec4 u\_Eye;\n' +

This constant indicate the position of the eye point (world coordinates), which is also the key idea of my fog algorithm implementation.

'varying vec4 v\_Color;\n' +

The variable is a **varying**, which means that it is interpolated between the color.It basically demonstrate the basic idea behind the fog limitation.

'varying float v\_Dist;\n' +

The variable is a **varying**, which means that it is interpolated between the distances. It basically demonstrate the key idea behind the fog algorithm.

'void main() {\n' +

        '  gl\_Position = u\_MvpMatrix \* a\_Position;\n' +

        '  v\_Color = a\_Color;\n' +

        // Calculate the distance to each vertex from eye point

        '  v\_Dist = distance(u\_ModelMatrix \* a\_Position, u\_Eye);\n' +

        '}\n';

The main() is the routine which defines as the entry point for the program, after all variable has been declared at the beginning of the codes. Within this main() function, it calculate the distance to each vertex from the eye point(world coordinates).This **v\_Dist = distance(u\_ModelMatrix \* a\_Position, u\_Eye**) is also the key idea of this fog algorithm.

var FSHADER\_SOURCE =

**FSHADER\_SOURCE**  is a variable declared to store the fragment shader program.

'#ifdef GL\_ES\n' +

'precision mediump float;\n' +

'#endif\n' +

This means that, if features in **GLSL ES** are not supported in webgl, rather it support a subset of a core features needed for the webgl. Also the **precision mediump float** will determine how much the GPU uses when calculating floats, which means that the **medium** will bethe less intensive but not as **lowp**(low precision) than **highp** which will be coursemore intensive.This is not the key idea of fog algorithm.

'uniform vec3 u\_FogColor;\n' +

This shows a Shader Material with a texture for the fog. It is also the key idea of the fog algorithm implementation.

'uniform vec2 u\_FogDist;\n' +

This shows a Shader Material with a distance of the fog, which means the starting point and the end point for the fog. It is also the key idea of the fog algorithm implementation.

'varying vec4 v\_Color;\n' +

The variable is a **varying**, which means that it is interpolated between the color. It basically demonstrate the key idea in the fragment shader of the fog algorithm.

'varying float v\_Dist;\n' +

The variable is a **varying**, which means that it is interpolated between the distances. It basically demonstrate the key idea behind the fog algorithm, also in the fragment shader as in the vertex shader.

'void main() {\n' +

        // Calculation of fog factor (factor becomes smaller as it goes further away from eye point)

        '  float fogFactor = clamp((u\_FogDist.y - v\_Dist) / (u\_FogDist.y - u\_FogDist.x), 0.0, 1.0);\n' +

        // Stronger fog as it gets further: u\_FogColor \* (1 - fogFactor) + v\_Color \* fogFactor

        '  vec3 color = mix(u\_FogColor, vec3(v\_Color), fogFactor);\n' +

        '  gl\_FragColor = vec4(color, v\_Color.a);\n' +

        '}\n';

The **main()** is the routine which defines as the entry point for the program, after all variable has been declared at the beginning of the codes in the fragment shader. Within this **main()** function, calculation of the fog factor , when factor decreases or becomes smaller as it moves away from the eye point, while it increase it get closer to the eye point. This means than as it increase it becomes **1.0** and as it decrease it becomes 0.0. Also the stronger fog as it then further, the color begins to appear from the hazy (atmospheric Effect), which make the color of the hazy different from the fog when mixed together.

function main()

This is a function called main() , to compile all codes in the <body> tag, function return a result at the end of the program.

var canvas = document.getElementById('webgl');

A variable declared as canvas to store a document **method()** **getElementById()** which returns an element object representing the element whose id property matches the specified string **'webgl'.** This ids are required to be unique, when specified. This is a useful approach to access specific element.

var gl = getWebGLContext(canvas);

if (!gl) {

            console.log('Failed to get the rendering context for WebGL');

            return;

        }

The **getWebGLContext** is an object that stores the current of the graphics library.The **WebGL** is a state machine, which means that , if I set **WebGL** variable to a specific value (canvas), that value in the canvas will not change until I change it.Example , is to set the background color used to clear the canvas once. Also it functionality is accessed through the context object, which means that the context object gl is the convention name. Moreover, the errors will be handled appropriately, if failed to get the rendering context for the **WebGL** based on the condition in the fog program but it is not any key idea of the algorithm.

if (!initShaders(gl, VSHADER\_SOURCE, FSHADER\_SOURCE)) {

            console.log('Failed to intialize shaders.');

            return;

        }

The shader will be linked and compiled together, so the condition is when the shader failed to initialize, it throws an error. This is because shaders has to be appropriately initilalize for the **WebGL** to light out the scene of the fog.

var n = initVertexBuffers(gl);

A variable **n** created to store, whiles creating buffer with the number of vertices. Also the surface number are set up in the **initVertexBuffers** which simply map vertices to a surface.

if (n < 1) {

            console.log('Failed to set the vertex information');

            return;

        }

This condition indicates that, if number of vertex is less than 1 then, it means it **Failed to set the vertex information,** This is because the number of vertices should be greater than 1 in order for the buffer to be complete.

var fogColor = new Float32Array([0.137, 0.231, 0.423]);

The variable **fogColor** is to store the color of the fog. This makes it the key idea of the fog algorithm.

var fogDist = new Float32Array([55, 80]);

The variable **fogDist** is to store the distance of the fog, this is because it indicate where the fog start, and when it completely covers object from the hazy. This also makes it the key idea of the fog algorithm.

var eye = new Float32Array([25, 65, 35, 1.0]);

The variable **eye** is to store the position of the eye point of the fog. This also makes it the key idea of the fog algorithm.

var u\_MvpMatrix = gl.getUniformLocation(gl.program, 'u\_MvpMatrix');

The variable **u\_MvpMatrix** stores the content of specific uniform variable within the program object.The gl.GetUniformLocation will return an integer of the specific variable.This is because after linking has already occurred, the command gl.GetUniformLocation can be set to obtain the specific location of the uniform variable. This specific variable string is the **u\_MvpMatrix** and therefore it make it not a key idea of the fog algorithm.

var u\_ModelMatrix = gl.getUniformLocation(gl.program, 'u\_ModelMatrix');

This also variable **u\_ModelMatrix** stores the content of specific uniform variable within the program object.The gl.GetUniformLocation will return an integer of the specific variable.This is because after linking has already occurred, the command gl.GetUniformLocation can be set to obtain the specific location of the uniform variable. This specific variable string is the **u\_ModelMatrix** and therefore it also not considered as the key idea of the fog algorithm

var u\_Eye = gl.getUniformLocation(gl.program, 'u\_Eye');

The variable **u\_Eye** stores the content of specific uniform variable within the program object. The gl.GetUniformLocation will return an interger of the specific variable. This is because after linking has already occurred, the command gl.GetUniformLocation can be set to obtain the specific location of the uniform variable. This specific variable string is the **u\_Eye** and therefore it make it is a key idea of the fog algorithm. This is because it set the position of eye point the fog in the algorithm.

var u\_FogColor = gl.getUniformLocation(gl.program, 'u\_FogColor');

The variable **u\_FogColor** stores the content of specific uniform variable within the program object.The gl.GetUniformLocation will return an interger of the specific variable.This is because after linking has already occurred, the command gl.GetUniformLocation can be set to obtain the specific location of the uniform variable. This specific variable string is the **u\_FogColor** and therefore it make it a key idea of the fog algorithm. This is because it set the color of the fog object.

var u\_FogDist = gl.getUniformLocation(gl.program, 'u\_FogDist');

The variable **u\_FogDist** stores the content of specific uniform variable within the program object.**The gl.GetUniformLocation** will return an integer of the specific variable. This is because after linking has already occurred, the command **gl.GetUniformLocation** can be set to obtain the specific location of the uniform variable. This specific variable string is the **u\_FogDist** and therefore it makes it a key idea of the fog algorithm. This is also because it set the distance of the fog, that is the starting and the ending point.

if (!u\_MvpMatrix || !u\_ModelMatrix || !u\_Eye || !u\_FogColor || !u\_FogDist) {

            console.log('Failed to get the storage location');

            return;

        }

The condition means that, if **u\_MvpMatrix, u\_ModelMatrix, u\_Eye , u\_FogColor,** and the **u\_FogDist** storage location are not accessed, it will throw an error message indicating that,if either of the location was not accessed. If this condition is omitted the program might crush into errors. This is a check errors in accessing storage, therefore it makes it not a key idea of fog implementation algorithm.

gl.uniform3fv(u\_FogColor, fogColor);

The gl.uniform means that when using WebGL I must pass in an array of length equal to 3, if my sharder uniform is vec3. Therefore, I will pass fogColor into the uniform variable. This makes it a key idea of fog algorithm because it indicate gl. **uniform3fv**( uniformLocation, data ).

gl.uniform2fv(u\_FogDist, fogDist);

The gl.uniform means that when using WebGL I must pass in an array of length equal to 2, Therefore, it will pass distances into the uniform variable. This makes it a key idea of fog algorithm because it indicate gl. **Uniform2fv**( Location, value).

gl.uniform4fv(u\_Eye, eye);

The gl.uniform means that when using WebGL I must pass in an array of length equal to 4, Therefore, it will pass eye point into the uniform variable. This makes it a key idea of fog algorithm because it indicate gl. **Uniform4fv**( Location, value).

 // Set clear color and enable hidden surface removal

        gl.clearColor(fogColor[0], fogColor[1], fogColor[2], 1.0); // Color of Fog

This code must be implemented in the codes, because it set clear color fog, and therefore make it a key idea in the fog algorithm.

gl.enable(gl.DEPTH\_TEST);

This indicate that each fragment which has been processed by the fragment shader will carry an associated depth value. Even if, the fragment are two-dimensional as they are going to be displayed on screen, this is because the depth value keeps information of how distant the fragment is from the screen. Also the fragment has been calculated by the fragment shader, when it is eligible for the depth testing. This occurs if the depth test is enabled, if the **gl** variable contains WebGL context.This not the key idea of fog algorithm.

var modelMatrix = new Matrix4();

The variable **modelMatrix** is used to store the matrices of the object.

modelMatrix.setScale(10, 10, 10);

The **modelMatrix** which contains the matrices of the object will be set to an appropriate scale in order for the size and edges to show.

gl.uniformMatrix4fv(u\_ModelMatrix, false, modelMatrix.elements);

**This means that Uniform4fv**( Location, value), where **u\_ModelMatrix, the value is the modelMatrix.elements,** if the Boolean is false**.** This is basically not the key idea of fog algorithm.

var mvpMatrix = new Matrix4();

The variable **mvplMatrix** is used to store the matrices of the object in it view, model and project.

mvpMatrix.setPerspective(30, canvas.width/canvas.height, 1, 1000);

The **Perspective** divide is performed when a point is multiplied by a **projection matrix** in **mvpMatrix** of the fog.

mvpMatrix.lookAt(eye[0], eye[1], eye[2], 0, 2, 0, 0, 1, 0);

The mvpMatrx function calls the lookAt to create a camera transformation matrix of the eye point of the fog.

mvpMatrix.multiply(modelMatrix);

The **mvpMartrix** function multiply to **modelMatrix** by the column of the matrix and add up the result of the fog.

gl.uniformMatrix4fv(u\_MvpMatrix, false, mvpMatrix.elements);

**This means that Uniform4fv**( Location, value), where **u\_mvMatrix,** the value is the **mvpMatrix.elements,** if the Boolean is false**.** This is basically not the key idea of fog algorithm.

 document.onkeydown = function(ev){ keydown(ev, gl, n, u\_FogDist, fogDist); };

The **onkeydown** event(**ev**) occurs when the user is press a key.What happens is that the number of times**(n)** the user press and hold, whether it the down arrow or right arrow, will trigger action in order to produce effect of the fog depending on the **u\_FogDist** and **fogDist**. This is a key idea of fog algorithm.

gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

The **COLOR\_BUFFER** which is a constant tells which buffer to clear of the fog and the **DEPTH\_BUFFER\_BIT** also clears the canvas, and both calculate the coordinate of the central point to be positioned.

gl.drawElements(gl.TRIANGLES, n, gl.UNSIGNED\_BYTE, 0);

This **drawElements** is a method used to render the primitives from the array of the fog data.

var modelViewMatrix = new Matrix4();

The variable **modelMatrix** is used to store the matrices of the object.

modelViewMatrix.setLookAt(eye[0], eye[1], eye[2], 0, 2, 0, 0, 1, 0);

In the modelMatrix , the setLookAt will return a view matrix which will then pass to the graphics of the fog.

modelViewMatrix.multiply(modelMatrix);

The **mvpMartrix** function multiply to **modelMatrix** by the column of the matrix and add up the result of the fog.

modelViewMatrix.multiplyVector4(new Vector4([1, 1, 1, 1]));

This multiply the modelViewMatrix by it model matrix of the fog.

 mvpMatrix.multiplyVector4(new Vector4([1, 1, 1, 1]));

This multiply the mvpMatrix by it view, model and projection matrix of the fog.

function keydown(ev, gl, n, u\_FogDist, fogDist) {

        switch (ev.keyCode) {

            case 38: // Up arrow key -> Increase the maximum distance of fog

                fogDist[1]  += 1;

                break;

This is when the user press the up key, it increases the maximum distance of the fog, by **fogDist[1]+=1**, which means that as the user press on the up key, **fogDist[1]** increase by one until it get to the point of object appears clearly from the hazy.

            case 40: // Down arrow key -> Decrease the maximum distance of fog

                if (fogDist[1] > fogDist[0]) fogDist[1] -= 1;

                break;

            default: return;

        }

This is when the user press the down key, it decreases to the minimum distance of the fog, which means that, if the **fogDist[1] > fogDist[0])**(fogDist[1] is greater than fogDist[0]) than **fogDist[1] -= 1( will decrease by one),** therefore object begun to disappear from the hazy, depending on the number of user press and hold.This basically a key idea of fog algorithm.

 gl.uniform2fv(u\_FogDist, fogDist);

The gl.uniform means that when using WebGL I must pass in an array of length equal to 2, Therefore, it will pass distances into the uniform variable. This makes it a key idea of fog algorithm because it indicate gl. **Uniform2fv**( Location, value).

gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

The **COLOR\_BUFFER** which is a constant tells which buffer to clear of the fog and the **DEPTH\_BUFFER\_BIT** also clears the canvas, and both calculate the coordinate of the central point to be positioned.

gl.drawElements(gl.TRIANGLES, n, gl.UNSIGNED\_BYTE, 0);

This drawElements is used to render the primitives from the array of the fog data.

function initVertexBuffers(gl) {

A function called initVertexBuffer to compile the content of the vertices of the fog.

var vertices = new Float32Array([   // Vertex coordinates

            1, 1, 1,  -1, 1, 1,  -1,-1, 1,   1,-1, 1,    // v0-v1-v2-v3 front

            1, 1, 1,   1,-1, 1,   1,-1,-1,   1, 1,-1,    // v0-v3-v4-v5 right

            1, 1, 1,   1, 1,-1,  -1, 1,-1,  -1, 1, 1,    // v0-v5-v6-v1 up

            -1, 1, 1,  -1, 1,-1,  -1,-1,-1,  -1,-1, 1,    // v1-v6-v7-v2 left

            -1,-1,-1,   1,-1,-1,   1,-1, 1,  -1,-1, 1,    // v7-v4-v3-v2 down

            1,-1,-1,  -1,-1,-1,  -1, 1,-1,   1, 1,-1     // v4-v7-v6-v5 back

        ]);

The variable color stores the 32-bit floating numbers in byte order to generate the vertices of the fog. This is basically not the key idea of fog Algorithm because, Also the v0,v1,v2,v3,v4,v5,v6 and v7 matches the vertices to indicate how the fog will be positioned.

var colors = new Float32Array([     // Colors

            0.4, 0.4, 1.0,  0.4, 0.4, 1.0,  0.4, 0.4, 1.0,  0.4, 0.4, 1.0,  // v0-v1-v2-v3 front

            0.4, 1.0, 0.4,  0.4, 1.0, 0.4,  0.4, 1.0, 0.4,  0.4, 1.0, 0.4,  // v0-v3-v4-v5 right

            1.0, 0.4, 0.4,  1.0, 0.4, 0.4,  1.0, 0.4, 0.4,  1.0, 0.4, 0.4,  // v0-v5-v6-v1 up

            1.0, 1.0, 0.4,  1.0, 1.0, 0.4,  1.0, 1.0, 0.4,  1.0, 1.0, 0.4,  // v1-v6-v7-v2 left

            1.0, 1.0, 1.0,  1.0, 1.0, 1.0,  1.0, 1.0, 1.0,  1.0, 1.0, 1.0,  // v7-v4-v3-v2 down

            0.4, 1.0, 1.0,  0.4, 1.0, 1.0,  0.4, 1.0, 1.0,  0.4, 1.0, 1.0   // v4-v7-v6-v5 back

        ]);

The variable color stores the 32-bit floating numbers in byte order to generate the color of the fog. This is basically not the key idea of fog Algorithm because, v0,v1,v2,v3,v4,v5,v6 and v7 matches the vertices to indicate the color of that particular position of the fog.

var indices = new Uint8Array([       // Indices of the vertices

            0, 1, 2,   0, 2, 3,    // front

            4, 5, 6,   4, 6, 7,    // right

            8, 9,10,   8,10,11,    // up

            12,13,14,  12,14,15,    // left

            16,17,18,  16,18,19,    // down

            20,21,22,  20,22,23     // back

        ]);

The variable indices store the indicies of the vertice, which means that the **Uint8Array** typed array represent an array of 8-bit unsigned integers. Firstly the content is initialized to zero 0.This is not the key idea of fog algorithm.

var indexBuffer = gl.createBuffer();

The variable **indexBuffer** stores the created buffer, this **createBuffer**() method create and initializes a **WebGLBuffe**r for storing data such as vertices and colors of the fog.

if (!indexBuffer)

            return -1;

The condition checks if the buffer is not empty.

if (!initArrayBuffer(gl, vertices, 3, gl.FLOAT, 'a\_Position')) return -1;

This condition check if the position vertices are not empty.

if (!initArrayBuffer(gl, colors, 3, gl.FLOAT, 'a\_Color')) return -1;

This condition check if the color of the vertices are not empty.

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, indexBuffer);

This binds a given **WebGLBuffer** to a target of the fog.

gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, indices, gl.STATIC\_DRAW);

This initializes and creates buffer object of the fog to store data.

   return indices.length;

    }

function initArrayBuffer (gl, data, num, type, attribute) {

function initArrayBuffer (gl, data, num, type, attribute) {

var buffer = gl.createBuffer();

The variable buffer to store the created buffer.

if (!buffer) {

            console.log('Failed to create the buffer object');

            return false;

        }

This condition check, if the buffer can be created, else return an error message.

gl.bindBuffer(gl.ARRAY\_BUFFER, buffer);

This binds a given **WebGLBuffer** to a target of the fog.

gl.bufferData(gl.ARRAY\_BUFFER, data, gl.STATIC\_DRAW);

This initializes and creates buffer object of the fog to store data.

var a\_attribute = gl.getAttribLocation(gl.program, attribute);

The variable a\_attribute stores the content of location of the attribute variable.

if (a\_attribute < 0) {

            console.log('Failed to get the storage location of ' + attribute);

            return false;

        }

This condition indicates that, if a\_attribute of vertex is less than 1 then, it means it **Failed to get storage location information,** This is because the a\_attribute should be greater than 1 in order for the position to be complete and return location of the attribute variable

gl.vertexAttribPointer(a\_attribute, num, type, false, 0, 0);

This vertexAttribPointer binds the buffer to **gl.ARRAY\_BUFFER** to a generic vertice a\_attribute of the current vertex object and give it layout.

 gl.enableVertexAttribArray(a\_attribute);

This turns on the a\_attribute array at the specific index into the list of all attribute arrays.

 gl.bindBuffer(gl.ARRAY\_BUFFER, null);

        return true;

    }

This binds a given **WebGLBuffer** to a target of the fog, depending on the condition of the a\_attribute.