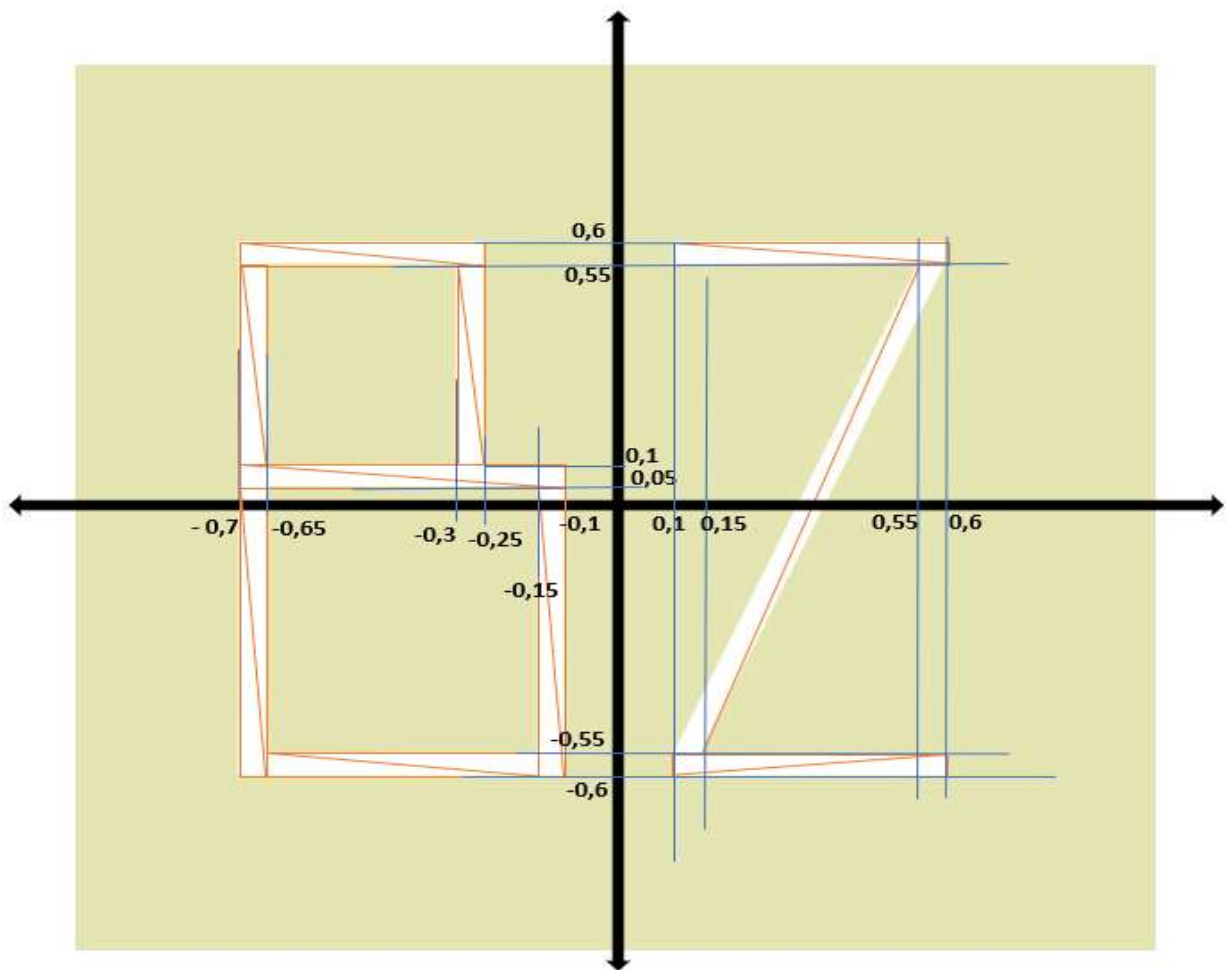


REPORT

Berfin Açıkgöz / <https://github.com/bberfin>

In this assignment , I wrote a WebGL application that displays the first letter of my first name (B) and the last letter of my last name (Z) . To do this , I drew 2D figures using geometric primitive that is triangle .



I used HTML to describe my web page and I wrote the Javascript code to draw the graphs

- Firstly, in the HTML file I created a canvas which is the area on the page that will display the output of my program . It provides an effective way to draw graphics using JavaScript

```
<canvas id="glcanvas" width="650" height="600"></canvas>
```

The attributes width and height determine the size of the canvas and id represents the identifier of the canvas element .

- In the javascript file ,I initialized the GL context and I checked if WebGL is available and working or not .

```
// Initializing the GL context
const canvas = document.querySelector("#glcanvas");
gl = canvas.getContext("webgl");

// if WebGL is available and working , can continue
if (!gl) {
    alert("Unable to initialize WebGL. Your browser or machine may not support it.");
    return;
}
```

- These are the vertex shader and fragment shader (in the HTML file).

```
<script id="vertexShaderId" type="x-shader/x-vertex">

    attribute vec4 vPosition;
    uniform float theta;
    uniform vec4 relocate ;
    uniform float sizeC;
    uniform float sizeX;
    uniform float sizeY;

    void main(){
        gl_Position.x = (cos(theta) * vPosition.x - sin(theta) * vPosition.y)* sizeX ;
        gl_Position.y = (sin(theta) * vPosition.x + cos(theta) * vPosition.y)* sizeY;
        gl_Position.z = 0.0 ;
        gl_Position.w = 1.0 + sizeC ;

        gl_Position = gl_Position + relocate; }

</script>

<script id="fragmentShaderId" type="x-shader/x-fragment">
    precision mediump float;
    uniform vec4 fColor;

    void main(){
        gl_FragColor= fColor; }

</script>
```

- Vertex shader is the program code called on every vertex. It will be used to transform the triangles from one place to another. It will handle the data of each vertex such as vertex coordinates . It calculates the position of each vertex of a primitive polygon and stores it in the varying **gl_Position**.

In the vertex shader,

- The position of the object is changed according to the user's request (by adding "relocate").
- The object is rotating according to the user's request (using the "theta" angle in the operation).
- The object resizing according to the user's request (adding the "sizeC").
- The object scaling according to the user's request (multiplying the "sizeX or sizeY")

- Fragment shader is the code that runs on all pixels of every fragment. It is written to calculate and fill the color on individual pixels.

In fragment shader,

- The color of the object is white at first, but will then change continuously according to the user's preference.
- In the initShaders.js (javascript code), I loads its source codes and compiled , and I attached them to the program that I created and linked the program . The following is a call to be able to use the actions I performed for shaders in initShaders.js in the program I created.

```
var program = initShaders(gl,"vertexShaderId","fragmentShaderId");
gl.useProgram( program );
```

- I stored the vertices manually using arrays

```
//COORDINATES OF LETTERS

var vertices = [
  // letter Z :
  vec2(0.1,0.6),vec2(0.6,0.6),
  vec2(0.6,0.55),vec2(0.6,0.55),
  vec2(0.1, 0.6),vec2(0.1, 0.55),

  vec2(0.55,0.55),vec2(0.6,0.55),
  vec2(0.15,-0.55),vec2(0.15,-0.55),
  vec2(0.1, -0.55),vec2(0.55,0.55),

  vec2(0.1,-0.55),vec2(0.6,-0.55),
  vec2(0.1,-0.6),vec2(0.1,-0.6),
  vec2(0.6, -0.6),vec2(0.6,-0.55),

  // letter B :
  vec2(-0.7,0.6),vec2(-0.25,0.6),
  vec2(-0.25,0.55),vec2(-0.25,0.55),
  vec2(-0.7,0.6),vec2(-0.7,0.55),

  vec2(-0.3, 0.55),vec2(-0.25, 0.55),
  vec2(-0.25,0.1),vec2(-0.25,0.1),
  vec2(-0.3,0.1),vec2(-0.3,0.55),

  vec2(-0.7, 0.55),vec2(-0.65, 0.55),
  vec2(-0.65,0.1),vec2(-0.65,0.1),
  vec2(-0.7,0.1),vec2(-0.7,0.55),

  vec2(-0.7, 0.1),vec2(-0.1, 0.1),
  vec2(-0.1,0.05),vec2(-0.1,0.05),
  vec2(-0.7,0.05),vec2(-0.7,0.1),

  vec2(-0.7, 0.05),vec2(-0.65, 0.05),
  vec2(-0.65,-0.6),vec2(-0.65,-0.6),
  vec2(-0.7,-0.6),vec2(-0.7,0.05),

  vec2(-0.15, 0.05),vec2(-0.1, 0.05),
  vec2(-0.1,-0.6),vec2(-0.1,-0.6),
  vec2(-0.15,-0.6),vec2(-0.15,0.05),

  vec2(-0.65, -0.55),vec2(-0.15, -0.55),
  vec2(-0.15,-0.6),vec2(-0.15,-0.6),
  vec2(-0.65,-0.6),vec2(-0.65,-0.55)
];
```

- I created a vertex buffer object on the gpu and made this buffer the current buffer to use .
- I pass the vertices to the WebGL rendering pipeline using vertex buffer
- And then, I associated my shader variables with my data buffer .

```
//CREATING A VERTEX BUFFER OBJECT ON THE GPU

var bufferId = gl.createBuffer();

//making this buffer the current buffer to use
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );

gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );

// Associating our shader variables with our data buffer
var vPosition = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );
```

DRAWING LETTERS :

To draw my 2D objects , I used the method **drawArrays()** that is provided by the WebGL API . First parameter is the mode using which I selected the my objects I want to draw . I made each rectangle from triangles . So I chose **TRIANGLES** for this. I placed my letters in coordinates so that they consist of rectangles. I had to use 10 rectangles -> 20 triangles -> 60 vertices . So the drawArrays() function has parameters from 0 to 60 .

```
function render(){

    // first background color
    gl.clearColor(0.9, 0.9, 0.7, 1.0);

    // Clear the color buffer with specified clear color
    gl.clear(gl.COLOR_BUFFER_BIT);

    gl.drawArrays( gl.TRIANGLES, 0, 60 );

    gl.uniform1f(scale,sizeC);
    gl.viewport(Tx,Ty,glcanvas.width,glcanvas.height);


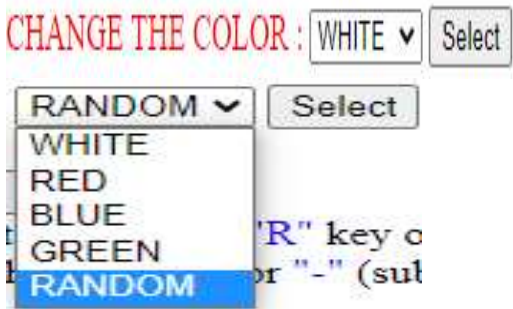
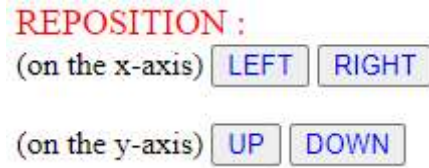



    if(rotation == true){
        setRotating();
    }

    setTimeout(function(){
        requestAnimationFrame(render);
    },delay);
}
```

!!!!

With **render()** function , first I set the background color (Clearing the canvas) , then Clear the color buffer bit and then I print my letters (triangles), this order is very important.

- Table of functions that can be used in my application and how to use them :

FUNCTIONS :	SYMBOLIC DISPLAY :	USAGE :
DRAWING LETTERS		
CHANGING THE COLOR		MOUSE <i>(There is a menu where you can choose 4 different colors or a random color)</i>
RELOCATING		MOUSE <i>(click the buttons)</i>
SET SPEED OF ROTATING		KEYBOARD <i>(press "+" or "-" keys on keyboard)</i>
START OR STOP ROTATING		MOUSE <i>(click the buttons)</i>
CHANGE THE DIRECTION OF ROTATING		KEYBOARD <i>press "R" key on keyboard</i>
SCALING		MOUSE <i>(using the slider, you can minimize or enlarge)</i>
SCALING BY X OR SCALING BY Y ALSO TRANSLATION		MOUSE <i>(using the slider, you can scale by x or by y) (The -x and -y ends translate with respect to the x and y axes.)</i>

CHANGING THE COLOR :

```
// CHANGING THE COLORS

var buttonColor = document.getElementById("ColorButton");
var colorLocation = gl.getUniformLocation(program,"fColor");

document.getElementById("color").onchange = function(){delay2 = this.value;};

var blue =[0.0,0.0,0.5,0.5];
var red =[1.0, 0.0, 0.0, 1.0];
var white =[1.0,1.0,1.0,1.0];
var green =[0.0, 0.5, 0.0,1.0];

buttonColor.addEventListener("click",
function(){
    if(delay2=="white"){color = white;gl.uniform4fv(colorLocation,color);}
    else if(delay2=="blue"){color = blue;gl.uniform4fv(colorLocation,color);}
    else if(delay2=="red"){color = red;gl.uniform4fv(colorLocation,color);}
    else if(delay2=="green"){color = green;gl.uniform4fv(colorLocation,color);}
    else if(delay2 == "random"){
        color = [Math.random(), Math.random(), Math.random(), Math.random()];
        gl.uniform4fv(colorLocation,color);
    }
});
```

With `addEventListener()` , when the user selects a color from the menu and clicks on "select" , the color change process takes place .

I found the vectorial equivalents for 4 GLSL colors . The `math.Random` function returns a value between 0 and 1. In this way, we can generate a random GLSL color value. The color of the object will change with the **`uniform4fv()`** function whichever color the user wants. It specifies the value of a uniform variable for the current program object.

- In the fragment shader (in the html), the varying (`fColor`) that holds the color value is assigned to `gl_FragColor`, which holds the final color of the object.

```
<script id="fragmentShaderId" type="x-shader/x-fragment">
    precision mediump float;
    uniform vec4 fColor;

    void main(){
        gl_FragColor= fColor; }
</script>
```

RELOCATING :

I was able to replace the letters with the parameters I placed in the **`viewport()`** function. I called this viewport function inside the **`render`** function.

```
gl.viewport(Tx,Ty,glcanvas.width,glcanvas.height);
```

When the user press the right button and wanted to move the letters to the right , I added positive 10 to the x coordinate . When the user press the left button and wanted to move the letters to the left ,I added negative 10 to the x coordinate .

When the user press the up button and wanted to move the letters to the up , I added positive 10 to the y coordinate .When the user press the down button and wanted to move the letters to the down ,I added negative 10 to the coordinate .

I tracked if the button was pressed with **addEventListener("click",...)** .

```
//RELOCATING

var right_x = document.getElementById("right_x");
right_x.addEventListener("click",
    function(){ Tx += 10;
    });
var left_x = document.getElementById("left_x");
left_x.addEventListener("click",
    function(){ Tx -= 10;
    });

var up_y = document.getElementById("up_y");
up_y.addEventListener("click",
    function(){ Ty += 10;
    });
var down_y = document.getElementById("down_y");
down_y.addEventListener("click",
    function(){ Ty -= 10;
    });
```

SET SPEED OF ROTATING AND CHANGE THE DIRECTION OF ROTATING :

```
//SET SPEED OF ROTATING
window.addEventListener("keydown",
    function(){
        switch(event.keyCode){
            case 82 :
                inverse = !inverse;
                break;
            case 107 :
                delay /=1.25;
                break;
            case 109 :
                if(delay<2000)
                    delay *= 1.25;
                break;
        }
    });
```

82 : “ R (r) ” key and is for change the direction

107 : “ + ” key and is for speed up

109 : “ - ” key and is for slow down (I set a limit on the delay value so it doesn't get too slow)

As the delay decreases, the waiting time will decrease and the rotating process will accelerate.

I tracked if the any of these keys was pressed on the keyboard with **window.addEventListener(“keydown”,...)** .

```
setTimeout(function(){  
    requestAnimationFrame(render);  
  
    },delay);
```

setTimeout() method is for calling the **render** function after a specified number of milliseconds . (Here it is “delay” and increases or decreases according to the user's request.)

requestAnimationFrame() method is for requesting the browser calling a specified function to update an animation before the next repaint . The method takes a “render” callback as an argument to be invoked before the repaint.

```
//LOCATION FOR ROTATING  
  
thetaLoc = gl.getUniformLocation(program,"theta");
```

```
function setRotating(){  
    theta += (inverse ? -0.1 : 0.1);  
    gl.uniform1f(thetaLoc,theta);  
}
```

Theta angle rotates clockwise when it is positive and counterclockwise when it is negative. We calculate whether it will be negative or positive when the user presses the R key on the keyboard.

Thanks to the **uniform1f()**, the theta angle plays a role in determining the direction of rotation.

STOP OR START ROTATING:

```
//STOP OR START ROTATING

var startButton = document.getElementById("startButton");
startButton.addEventListener("click",
    function(){
        rotation=true;
    });
var stopButton = document.getElementById("stopButton");
stopButton.addEventListener("click",
    function(){
        rotation=false;
    });
```

```
if(rotation == true){
    setRotating();
}
```

With **getElementById()**, it detects whether the start button or stop button is clicked or not, and if they are clicked, their operations are performed.

We draw the letters inside the render function. Therefore, in the render the new position must change after each rotation . If rotation is requested, the user will click "start" and this will be detected with `addEventListener("click",...)` and the value of "rotation" will be true . In this way, the rotation process will be run in the render. Otherwise, "rotation" be false and will not work in render.

SCALING , SCALING BY X -axes OR SCALING BY Y-axes , ALSO TRANSLATION

When the user move the slide to the right or left and wanted to minimize or enlarge the letters , the value of the slide is added to the position of the object (`position.w`) (in the html) . So the letter gets bigger or smaller .

```
<script id="vertexShaderId" type="x-shader/x-vertex">

    attribute vec4 vPosition;
    uniform float theta;
    uniform vec4 relocate ;
    uniform float sizeC;
    uniform float sizeX;
    uniform float sizeY;

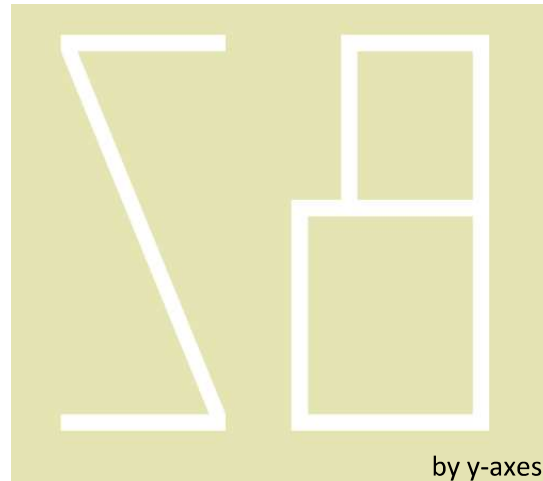
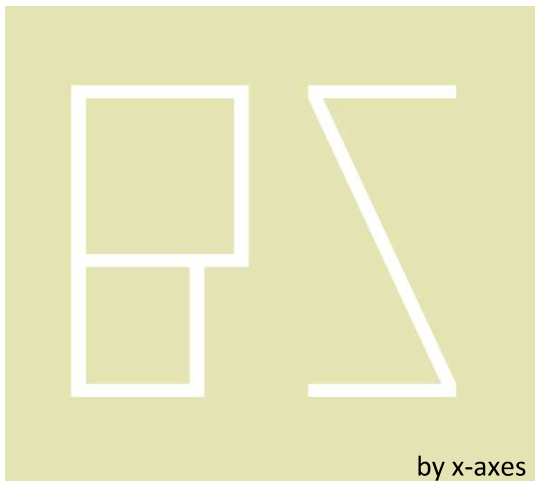
    void main(){
        gl_Position.x = (cos(theta) * vPosition.x - sin(theta) * vPosition.y)* sizeX ;
        gl_Position.y = (sin(theta) * vPosition.x + cos(theta) * vPosition.y)* sizeY;
        gl_Position.z = 0.0 ;
        gl_Position.w = 1.0 + sizeC ;

        gl_Position = gl_Position + relocate; }

</script>
```

When the user move the slide to the right or left and wanted to scaling the letters by x-axes or y-axes , the value of the slide (sizeX ,or sizeY) is multiplied to the position of the object (position.y or position_x) (in the html) . So the letter gets scaled .

- Multiplication with negative values translates letters :



I followed these requests of the user with "onchange" when using the slider on the screen. With getElementById(), it detects whether the slider is clicked or not, and if it is clicked, its operations are performed.

```
//SCALING
```

```
document.getElementById("slide").onchange = function(){sizeC = this.value;};  
scale = gl.getUniformLocation(program,"sizeC");
```

```
document.getElementById("scale_x").onchange= function(){sizeX=this.value;};  
document.getElementById("scale_y").onchange= function(){ sizeY=this.value;};  
  
scaleX = gl.getUniformLocation(program,"sizeX");  
scaleY = gl.getUniformLocation(program,"sizeY");
```

```
gl.uniform1f(scale,sizeC);
```

```
gl.uniform1f(scaleX,sizeX);  
gl.uniform1f(scaleY,sizeY);
```

I reflected this change to the objects with the **uniform1f()** function that I called in the render function.