

Assignment 1
MOHAMMAD SHABIB
19290116

This program Draws the letter M & B, and can do the following geometric transformations: rotation, scaling and translating, also it can change the letter & background color and the color transparency of the letters.

The GUI can be controlled Either using Mouse or Keyboard.

Function	Keyboard	Mouse
Start/Stop Rotation	Space	“Start/Stop Rotation” Button
Rotation Direction	Enter	“Change Rotation Direction” Button
Scaling	+ -	“Scale “ slider
Translating	→ D ← A ↑ W ↓ S	Move Up, Move Left, Move Down and Move Right Buttons
Font Color	R G	Font Color Picker
Transparency		“Transparency” Slider

Code Description:

```
window.onload = function main() {  
  const canvas = document.querySelector("#glcanvas");  
  // Initialize the GL context  
  gl = WebGLUtils.setupWebGL(canvas, {  
    transparent: true,  
  });  
  // Only continue if WebGL is available and working  
  if (!gl) {  
    alert(  
      "Unable to initialize WebGL. Your browser or machine may not support it."  
    );  
    return;  
  }  
  var program = initShaders(gl, "vertex-shader", "fragment-shader");  
  gl.useProgram(program);  
  
  gl.enable(gl.BLEND); //to enable transparency (to enable alpha in RGBA)  
  gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
```

The above code snippet initializes the vertex shaders and fragment shaders and creates a program and links it with the shader, also to enable the color transparency i added the last 2 lines.

```
var bufferId = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, bufferId);
gl.bufferData(gl.ARRAY_BUFFER, flatten(NAME), gl.STATIC_DRAW);

// Associate 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);
```

The above code creates buffer and we add the vertices to the buffer in order to display it And then associate our shader variables with our data buffer

```
<script id="vertex-shader" type="x-shader/xvertex">
    attribute vec4 vPosition;
    uniform float theta;
    uniform float scaler;
    uniform vec4 translate;
    void main()
    {
        //Rotation
        gl_Position.x = cos(theta) * vPosition.x -
        sin(theta) * vPosition.y;
        gl_Position.y = sin(theta) * vPosition.x +
        cos(theta) * vPosition.y;
        gl_Position.z = 0.0;
        gl_Position.w = 1.0;

        //Scaling
        gl_Position.x *= scaler;
        gl_Position.y *= scaler;

        //Translating
        gl_Position += translate;
    }
</script>
<script id="fragment-shader" type="x-shader/xfragment">
```

```
precision mediump float;
uniform vec4 u_color;
void main() {

    gl_FragColor = u_color;
}
</script>
```

The above code creates vertex shader and fragment shader
 The vertex shader is where we do all the geometric transformations

```
//Rotation
    gl_Position.x = cos(theta) * vPosition.x -
    sin(theta) * vPosition.y;
    gl_Position.y = sin(theta) * vPosition.x +
    cos(theta) * vPosition.y;    gl_Position.z = 0.0;
    gl_Position.w = 1.0;
```

The above code do the rotation by doing matrix multiplication

$$\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

```
//Scaling
    gl_Position.x *= scaler;
    gl_Position.y *= scaler;
```

The above code do the scaling by doing just doing scalar multiplication

```
//Translating
    gl_Position += translate;
```

The above code do the translation by adding 2 matrices

```

//CONNECTING THETA WITH THE SHADER
thetaLoc = gl.getUniformLocation(program, "theta");
theta = 0;
gl.uniform1f(thetaLoc, theta);

//CONNECTING SCALING WITH THE SHADER
scalerloc = gl.getUniformLocation(program, "scaler");
scaler = 1;
gl.uniform1f(scalerloc, scaler);

//CONNECTING COLOR WITH THE SHADER
colorLocation = gl.getUniformLocation(program, "u_color");
gl.uniform4fv(colorLocation, color);

//CONNECTING Translate WITH THE SHADER
translateloc = gl.getUniformLocation(program, "translate");
translate = [0, 0, 0, 0];
gl.uniform4f(
    translateloc,
    translate[0],
    translate[1],
    translate[2],
    translate[3]
);

```

The Above code connect the the theta, scaler, u_color and translate variable in the shader with the programm

```
thetaLoc = gl.getUniformLocation(program, "theta");
```

Then we create a way to access the shader variable to change the value and render it.

```
gl.uniform1f(thetaLoc, theta);
```

```
function render() {
  setTimeout(function () {
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);
    gl.uniform4fv(colorLocation, color); // letter color
    if (!stop) theta += isDirClockwise ? -0.1 : 0.1; //rotation direction
    gl.uniform1f(thetaLoc, theta); //rotation
    gl.uniform1f(scalerloc, scaler); //scalloing
    gl.uniform4f(
      //translating
      translateLoc,
      translate[0],
      translate[1],
      translate[2],
      translate[3]
    );
    gl.drawArrays(gl.TRIANGLES, 0, NAME_size); //drawing the letters
    requestAnimationFrame(render);
  }, delay);
}
```

The above code does rendering and displaying the letters to the UI.

```
gl.clearColor(0.0, 0.0, 0.0, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);
```

Set the background color and clears the color buffer

```
gl.uniform4fv(colorLocation, color); // letter color
```

Set the fragment color(The letters Color), `colorLocation` is the variable we use to access the color in the shader and the `color` variable is the value we want

```
if (!stop) theta += isDirClockwise
```

Controls the direction of the theta and also if the theta should increase or stay fixed

```

gl.uniform1f(thetaLoc, theta); //rotation
gl.uniform1f(scalerloc, scaler); //scalloing
gl.uniform4f(
    //translating
    translateLoc,
    translate[0],
    translate[1],
    translate[2],
    translate[3]
);

```

we to access the shader variable to change the values of theta and scaler and `translate` and render it

```

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

```

This tells the rendering pipe that the vertices will be triangles so each 3 vertices will resemble 1 triangle

```

requestAnimationFrame(render);

```

We do this in order to swap buffers and show the new details

```

setTimeout(function () , delay);

```

`Delay` allows to call the render function repeatedly with a specific interval in milliseconds

```

//letters
var l1_distance = 0.5;
var l1_width = [-0.7, -0.65];
//0,1 index for horizontal width, index 2 for vertical width
var l2_width = [0, 0.6, 0.05];
//index 0 for horizontal height
var l2_height = [0.05];

var floor = -0.5;
var top = 0.4;

```

These variables allow me to change the height and width of the rectangle instead of applying the change to each vertex manually.

```

var l1_distance = 0.5;
var l1_width = [-0.7, -0.65];
//0,1 index for horizontal width, index 2 for vertical width
var l2_width = [0, 0.6, 0.05];
//index 0 for horizontal height
var l2_height = [0.05];

var floor = -0.5;
var top = 0.4;
var NAME = [
  //L1
  vec2(l1_width[0], floor),
  vec2(l1_width[1], floor),
  vec2(l1_width[1], top),
  vec2(l1_width[0], top),
  vec2(l1_width[1], top),
  vec2(l1_width[0], floor),
  //
  vec2(l1_width[0], top),
  vec2(l1_width[1], top),
  vec2(l1_width[1] + l1_distance / 2, floor),

  vec2(l1_width[0] + l1_distance / 2, floor),
  vec2(l1_width[1] + l1_distance / 2, floor),
  vec2(l1_width[0], top),
  //
  vec2(l1_width[0] + l1_distance, top),
  vec2(l1_width[1] + l1_distance, top),
  vec2(l1_width[1] + l1_distance / 2, floor),
  vec2(l1_width[0] + l1_distance / 2, floor),
  vec2(l1_width[1] + l1_distance / 2, floor),
  vec2(l1_width[0] + l1_distance, top),

  //
  vec2(l1_width[0] + l1_distance, floor),
  vec2(l1_width[1] + l1_distance, floor),
  vec2(l1_width[1] + l1_distance, top),
  vec2(l1_width[0] + l1_distance, top),
  vec2(l1_width[1] + l1_distance, top),
  vec2(l1_width[0] + l1_distance, floor),

```

```
// L2

..... same as L1
];
```

The above code creates vertex for the first letter (M)

To draw a rectangle we use 2 triangles.

The letter M needs 4 rectangles so we used 8 triangles (24 vertices).

I drew the 2 letters in the same array, if we want to separate them we can do that just by creating another list.

HTML

```
<button id="StartRotationButton">Start/Stop Rotation</button>
```

Java

```
//Rotation Start/Stop Button
var StartButton = document.getElementById("StartRotationButton");
StartButton.addEventListener("click", function () {
    stop = !stop; });
```

This Button when clicked it change the value of stop to the opposite value

(The stop variable as shown above in the render function controls the increase or decrease in the value of theta if stop is 0 then theta will stay the same as non 0 then theta will continue increasing or decreasing.

HTML

```
<button id="DirectionButton">Change Rotation Direction</button>
```

Java

```
//Rotation Direction Button
var DirectionButton = document.getElementById("DirectionButton");
DirectionButton.addEventListener("click", function () {
    isDirClockwise = !isDirClockwise;
});
```

This Button when clicked The direction of the theta will change by changing the value of

`isDirClockwise`

(The `isDirClockwise` variable as shown above in the render functions choose if the theta will be increased or decreased)

HTML

```
<div>
  Rotation Speed 0
  <input
    id="RotationSpeedSlide"    type="range"    min="0"    max="500"    step="1"
value="50"    style="width: 500px"  /> 500
  </div>
```

Java

```
//slider, changing speed
document.getElementById("RotationSpeedSlide").onchange = function () {
  delay = this.value;
};
```

This slider will control the value of the rotation speed by changing the value of the delay variable(which controls the time interval of which the render function will be called).

HTML

```
<div>  Scale 0.1  <input    id="ScallingSlide"    type="range"    min="0.1"
    max="2"    step="0.02"    value="1"    style="width: 500px"  />
2  </div>
```

Java

```
//slider, Scaling
document.getElementById("ScallingSlide").onchange = function () {
  scaler = this.value;
};
```

This slider will control the magnitude of scaling by changing the value of the scaler variable(then multiplied by X and Y).

HTML

```
<label id="label">Font Color </label><input type="color" id="fontColor" />
```

JAVA

```
//Color Button
var ColorButton = document.getElementById("fontColor");
ColorButton.addEventListener("input", (e) => {
  let font_color = document.getElementById("labelxx");
  color = e.target.value;
  //console.log(color);
  color = color.slice(1, color.length);
  //CONVERTING HEX TO RGBA SINCE WEBGL ONLY USES RGBA
  var aRgbHex = color.match(/.{1,2}/g);
  color = [
    parseInt(aRgbHex[0], 16) / 255,
```

```

    parseInt(aRgbHex[1], 16) / 255,
    parseInt(aRgbHex[2], 16) / 255,
];
color.push(1);
console.log(color);
});

```

This color picker will change the color of the letters since HTML color input get the value in Hex we needed to convert it to the RGBA Format because **gl_FragColor** only support RGBA format (The code for transformation was obtained from the internet, it uses Regular Expression to convert.

The Color variable is the variable associated with the shader

HTML

```

<div>
  Transparency 0
  <input
    id="transparencySlide"
    type="range"
    min="0"
    max="1"
    step="0.001"
    value="1"
    style="width: 500px"
  />
  1
</div>

```

JAVA

```

//slider, Color Transparency
document.getElementById("transparencySlide").onchange = function () {
  color[3] = this.value;
};

```

The slider controls the value of alpha in RGBA (R, G, B, ALPHA)

HTML

```

<!--Empty Charecters-->
<label>          </label>
<button id="Up">Move Up</button>
<br />
<button id="left">Move Left</button>
<button id="Down">Move Down</button>

```

```
<button id="Right">Move Right</button>
<br />
<pre>
```

JAVA

```
//translate
var DirectionButton = document.getElementById("Right");
DirectionButton.addEventListener("click", function () {
    translate[0] += 0.05;
});
var DirectionButton = document.getElementById("left");
DirectionButton.addEventListener("click", function () {
    translate[0] -= 0.05;
});
var DirectionButton = document.getElementById("Up");
DirectionButton.addEventListener("click", function () {
    translate[1] += 0.05;
});
var DirectionButton = document.getElementById("Down");
DirectionButton.addEventListener("click", function () {
    translate[1] -= 0.05;
});
```

These buttons does the translations by change the value of the vec4 `translate` variable which is then passed to the shader and change the value

```
window.addEventListener("keydown", function () {
    switch (event.keyCode) {
        case 37: //arrow left
        case 65: // 'A'
            translate[0] -= 0.05;
            break;
        case 38: //arrow up
        case 87: // 'W'
            translate[1] += 0.05;
            break;
        case 39: //arrow right
        case 68: // 'D'
            translate[0] += 0.05;
            break;
        case 40: //arrow down
        case 83: // 'S'
            translate[1] -= 0.05;
```

```

    break;
case 32: // space
    stop = !stop;
    break;
case 13: // ENTER
    isDirClockwise = !isDirClockwise;
    break;
case 71: //'G'
    color = [0, 1, 0, 1];
    break;
case 82: //'R'
    color = [1, 0, 0, 1];
    Break;
case 187: //'+'
    scaler += 0.01;
    break;
case 189: //'-'
    scaler -= 0.01;
    break;

} });

```

This added a keyboard listener which associated the above keys to a specific instruction





19290116
MOHAMMAD SHABIB

Start/Stop Rotation Change Rotation Direction

Rotation Speed 0 500

Scale 0.1 2

Background Color 

Font Color 

Transparency 0 1

Move Up
Move Left Move Down Move Right

Keyboard Shortcuts:

R	:	Red Color
G	:	Green Color
Space	:	Start/Stop Rotating
Enter	:	Change Rotation Direction
Right arrow +		D : Translate Right
Left arrow +		A : Translate Left
Up arrow +		W : Translate Up
Down arrow +		S : Translate Rldownght
+ -	:	Scaling