Nama : Farhan Yusuf Akbar
NIM : 13519202
Kelas : 04

WebGL merupakan singkatan dari Web Graphics Library. WebGL adalah sebuah rasterization engine. WebGL menggambar komponen primitif seperti titik, garis dan segitiga berdasarkan kode yang dibuat. Hal-hal yang bisa dilakukan WebGL dilakukan dengan menggambar komponen-komponen primitif tersebut dan mengaturnya sedemikian rupa melalui kode program.

WebGL berjalan pada GPU komputer. Oleh karena itu dibutuhkan kode yang dapat berjalan pada GPU. Kode yang dibutuhkan berupa sebuah pasangan fungsi, yaitu fungsi vertex shader dan fragment shader. Fungsi ini ditulis dengan bahasa yang disebut GLSL (GL Shader Language. Pasangan fungsi ini disebut program.

I. How It Works

1. Triangle

   Pada vertex shader digunakan atribut a_position yang datanya diambil dari buffer, uniform u_matrix sebagai variabel global dan varying v_color yang akan diberikan ke fragment shader dari vertex shader.

```
<!-- vertex shader -->
<script  id="vertex-shader-2d" type="x-shader/x-vertex">
attribute vec2 a_position;

uniform mat3 u_matrix;

varying vec4 v_color;

void main() {
// Multiply the position by the matrix.
gl_Position = vec4((u_matrix * vec3(a_position, 1)).xy, 0, 1);

// Convert from clipspace to colorspace.
// Clipspace goes -1.0 to +1.0
// Colorspace goes from 0.0 to 1.0
v_color = gl_Position * 0.5 + 0.5;
}

</script>
```

Pada fragment shader, dilakukan declare varying yang sama dengan varying pada vertex shader.

```
<!-- fragment shader -->
<script  id="fragment-shader-2d" type="x-shader/x-fragment">
precision mediump float;

varying vec4 v_color;

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

Pada buffer diisi nilai-nilai yang mendefinisikan sebuah segitiga

```
function setGeometry(gl) {
  gl.bufferData(
      gl.ARRAY_BUFFER,
      new Float32Array([
              0, -100,
            150,  125,
           -175,  100]),
      gl.STATIC_DRAW);
}
```

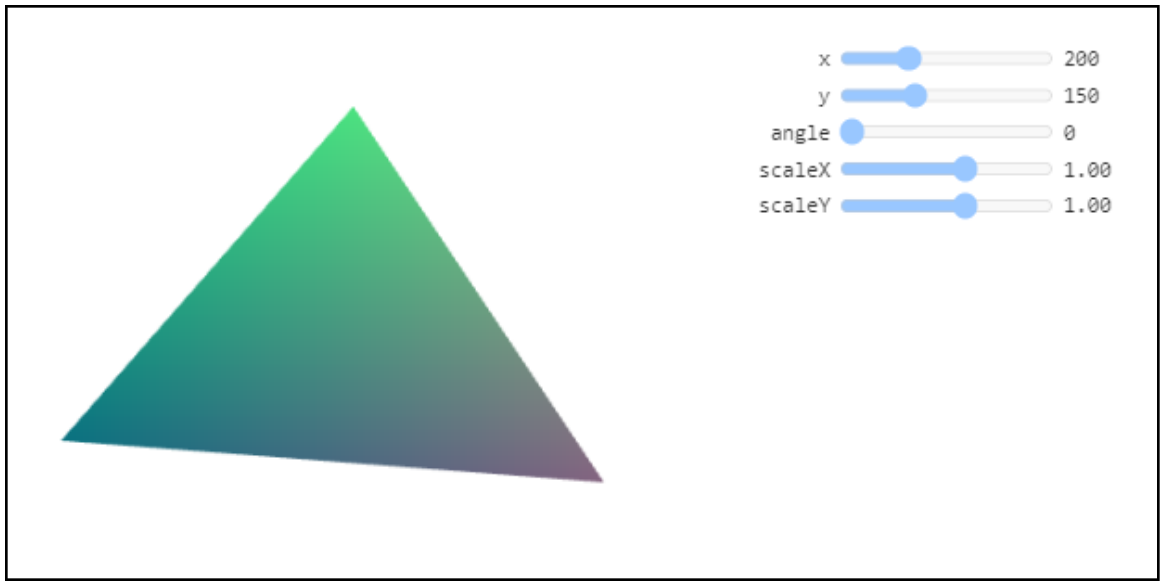
Dilakukan draw untuk 3 vertices

```
    // Draw the geometry.
    var primitiveType = gl.TRIANGLES;
    var offset = 0;
    var count = 3;
    gl.drawArrays(primitiveType, offset, count);
  }
}
```

Bagian kode utility yang apabila pengguna melakukan slide pada slider, bentuk dari segitiga akan berubah

```
// Setup a ui.
webglLessonsUI.setupSlider("#x", {value: translation[0], slide: updatePosition(0), max: gl.canvas.width });
webglLessonsUI.setupSlider("#y", {value: translation[1], slide: updatePosition(1), max: gl.canvas.height});
webglLessonsUI.setupSlider("#angle", {slide: updateAngle, max: 360});
webglLessonsUI.setupSlider("#scaleX", {value: scale[0], slide: updateScale(0), min: -5, max: 5, step: 0.01, precision: 2}
webglLessonsUI.setupSlider("#scaleY", {value: scale[1], slide: updateScale(1), min: -5, max: 5, step: 0.01, precision: 2}
```

Hasil akhirnya akan menjadi seperti berikut.



Pada contoh segitiga di atas, hanya dilakukan komputasi untuk 3 vertices sehingga vertex shader hanya dipanggil 3 kali dan berakibat menghasilkan 3 warna juga. Namun pada kenyataannya segitiga menunjukkan banyak warna. Ini disebabkan oleh varying. WebGL mengambil 3 nilai yang dihitung untuk setiap vertex dan saat dilakukan rasterization, WebGL melakukan interpolasi nilai di antara nilai yang kita hitung untuk vertices. Untuk setiap pixel akan dipanggil fungsi fragment shader dengan nilai yang sudah diinterpolasi untuk pixel tersebut.

2. Rectangle (interpolated with same color)

   Untuk menggambar sebuah persegi panjang, dibutuhkan dua segitiga sehingga dibutuhkan 6 vertices.

```
  // Draw the geometry.
  var primitiveType = gl.TRIANGLES;
  var offset = 0;
  var count = 6;
  gl.drawArrays(primitiveType, offset, count);
}
```

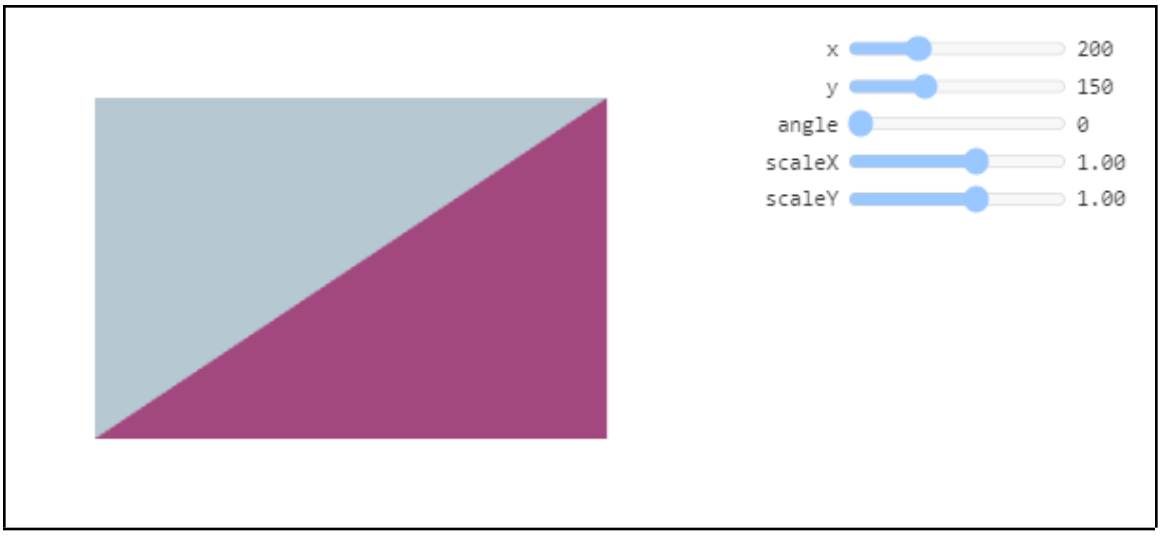
   Pada vertex shader ditambahkan atribut baru yaitu a_color sehingga dapat dilakukan lebih banyak pass data ke fragment shader melalui varying

```
<script  id="vertex-shader-2d" type="x-shader/x-vertex">
attribute vec2 a_position;
attribute vec4 a_color;

uniform mat3 u_matrix;

varying vec4 v_color;

void main() {
  // Multiply the position by the matrix.
  gl_Position = vec4((u_matrix * vec3(a_position, 1)).xy, 0, 1);

  // Copy the color from the attribute to the varying.
  v_color = a_color;
}
</script>
```

Menentukan warna yang akan digunakan oleh WebGL pada buffer.

```
function setColors(gl) {
  // Pick 2 random colors.
  var r1 = Math.random();
  var b1 = Math.random();
  var g1 = Math.random();
  var r2 = Math.random();
  var b2 = Math.random();
  var g2 = Math.random();

  gl.bufferData(
      gl.ARRAY_BUFFER,
      new Float32Array(
        [ r1, b1, g1, 1,
          r1, b1, g1, 1,
          r1, b1, g1, 1,
          r2, b2, g2, 1,
          r2, b2, g2, 1,
          r2, b2, g2, 1]),
      gl.STATIC_DRAW);
}
```

Hasil akhirnya akan menjadi seperti berikut.



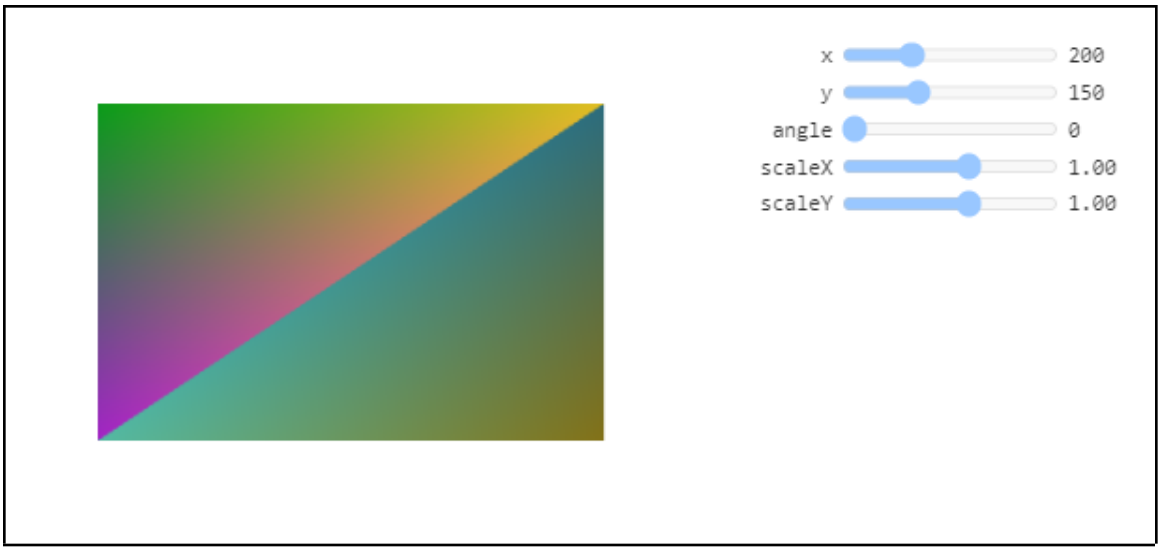
Persegi panjang yang dihasilkan memiliki 2 warna yang sama padahal digunakan varying untuk pass data ke fragment shader. Ini disebabkan warna dari setiap vertices pada segitiga dibuat sama sehingga interpolasi terjadi untuk 3 warna yang sama sehingga warna yang dihasilkan juga sama untuk satu segitiga

3. Rectangle (interpolated with different color)

   Persegi panjang ini sama dengan persegi panjang sebelumnya namun warna untuk setiap vertices berbeda sehingga terjadi interpolasi warna yang menghasilkan warna beragam pada satu segitiga.

```
function setColors(gl) {
  // Make every vertex a different color.
  gl.bufferData(
      gl.ARRAY_BUFFER,
      new Float32Array(
        [ Math.random(), Math.random(), Math.random(), 1,
          Math.random(), Math.random(), Math.random(), 1,
          Math.random(), Math.random(), Math.random(), 1,
          Math.random(), Math.random(), Math.random(), 1,
          Math.random(), Math.random(), Math.random(), 1,
          Math.random(), Math.random(), Math.random(), 1]),
      gl.STATIC_DRAW);
}
```

   Hasil akhirnya adalah sebagai berikut.



II. Image Processing

Untuk menggambar image di WebGL diperlukan textures. Serupa dengan cara WebGL menggunakan koordinat clip space saat melakukan render alih alih menggunakan pixel, WebGL menggunakan koordinat textures saat membaca textures. Koordinat texture memiliki range dari 0,0 ke 1,0 terlepas dari dimensi texturenya.

Untuk melakukan pass koordinat texture dari vertex shader ke fragment shader, dibutuhkan deklarasi atribut texture pada vertex shader

```
<!-- vertex shader -->
<script  id="vertex-shader-2d" type="x-shader/x-vertex">
attribute vec2 a_position;
attribute vec2 a_texCoord;

uniform vec2 u_resolution;

varying vec2 v_texCoord;

void main() {
   .......
   v_texCoord = a_texCoord;
}
</script>
```

Pada fragment shader, pewarnaan dilakukan berdasarkan textures yang di-pass dari vertex shader.

```
<!-- fragment shader -->
<script  id="fragment-shader-2d" type="x-shader/x-fragment">
precision mediump float;

// our texture
uniform sampler2D u_image;

// the texCoords passed in from the vertex shader.
varying vec2 v_texCoord;

void main() {
   gl_FragColor = texture2D(u_image, v_texCoord);
}
</script><!--
```

Lalu, dilakukan upload image, membuat texture dan copy image ke texture tersebut.

```
function main() {
  var image = new Image();
  requestCORSIfNotSameOrigin(image, "https://webglfundamentals.org/webgl/resources/leaves.jpg")
  image.src = "https://webglfundamentals.org/webgl/resources/leaves.jpg";
  image.onload = function() {
    render(image);
  };
}

function render(image) {
  // Get A WebGL context
  /** @type {HTMLCanvasElement} */
  var canvas = document.querySelector("#canvas");
  var gl = canvas.getContext("webgl");
  if (!gl) {
    return;
  }
```

Hasilnya adalah sebagai berikut

1. Swapping red and blue image
   Kita bisa melakukan manipulasi image dengan menukar warna merah dan biru. Pada fragment shader :

```
void main() {
    gl_FragColor = texture2D(u_image, v_texCoord).bgra;
}
```

   Hasilnya akan menjadi seperti berikut.

2. Blurred image
   Manipulasi image juga dapat dilakukan dengan merata-ratakan pixel kiri dan kanan untuk setiap pixel pada fragment shader sehingga gambar tampak blurry.

```
<!-- fragment shader -->
<script  id="fragment-shader-2d" type="x-shader/x-fragment">
precision mediump float;

// our texture
uniform sampler2D u_image;
uniform vec2 u_textureSize;

// the texCoords passed in from the vertex shader.
varying vec2 v_texCoord;

void main() {
   vec2 onePixel = vec2(1.0, 1.0) / u_textureSize;
   gl_FragColor = (
       texture2D(u_image, v_texCoord) +
       texture2D(u_image, v_texCoord + vec2(onePixel.x, 0.0)) +
       texture2D(u_image, v_texCoord + vec2(-onePixel.x, 0.0))) / 3.0;
}
</script><!--
```

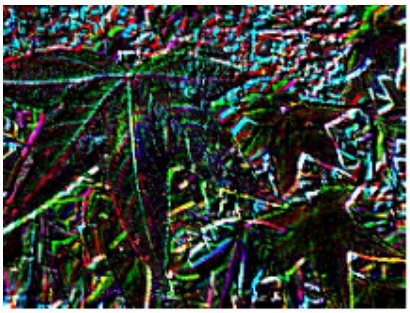
Hasilnya akan menjadi seperti berikut.

3. Another image processing

   Kita bisa melakukan berbagai image processing dengan melakukan berbagai modifikasi pada pixel disekitar pixel yang sedang diproses oleh fragment shader. Contohnya adalah 3x3 kernel. 3x3 kernel merupakan sebuah matrix yang mempresentasikan seberapa besar perkalian yang dilakukan pada 8 pixel disekitar pixel yang sedang di render. Dengan 3x3 pixel ini dapat dihasilkan blur image, sharpened image, emboss, dan lainnya.

```
</script>
<!-- fragment shader -->
<script  id="fragment-shader-2d" type="x-shader/x-fragment">
precision mediump float;

// our texture
uniform sampler2D u_image;
uniform vec2 u_textureSize;
uniform float u_kernel[9];
uniform float u_kernelWeight;

// the texCoords passed in from the vertex shader.
varying vec2 v_texCoord;

void main() {
   vec2 onePixel = vec2(1.0, 1.0) / u_textureSize;
   vec4 colorSum =
       texture2D(u_image, v_texCoord + onePixel * vec2(-1, -1)) * u_kernel[0] +
       texture2D(u_image, v_texCoord + onePixel * vec2( 0, -1)) * u_kernel[1] +
       texture2D(u_image, v_texCoord + onePixel * vec2( 1, -1)) * u_kernel[2] +
       texture2D(u_image, v_texCoord + onePixel * vec2(-1,  0)) * u_kernel[3] +
       texture2D(u_image, v_texCoord + onePixel * vec2( 0,  0)) * u_kernel[4] +
       texture2D(u_image, v_texCoord + onePixel * vec2( 1,  0)) * u_kernel[5] +
       texture2D(u_image, v_texCoord + onePixel * vec2(-1,  1)) * u_kernel[6] +
       texture2D(u_image, v_texCoord + onePixel * vec2( 0,  1)) * u_kernel[7] +
       texture2D(u_image, v_texCoord + onePixel * vec2( 1,  1)) * u_kernel[8] ;

   gl_FragColor = vec4((colorSum / u_kernelWeight).rgb, 1);
}
</script><!--
```

Salah satu contoh hasil image processing dari kernel adalah sebagai berikut.



edgeDetect3

Referensi

https://webglfundamentals.org/webgl/lessons/webgl-fundamentals.html
https://webglfundamentals.org/webgl/lessons/webgl-how-it-works.html
https://webglfundamentals.org/webgl/lessons/webgl-image-processing.html

Link github : https://github.com/farhanyusuf/Tugas-Grafkom
Link youtube : https://youtu.be/YPJagCC8HgA