Web GL Fundamentals - Manipulation and Image Processing

IF3260 Grafika Komputer

Ryo Richardo 13519193

1. How it works

Pada bagian ini, WebGL Fundamentals menjelaskan tentang sintaks dasar untuk membentuk objek 2 dimensi sederhana, yaitu segitiga. Untuk melakukan hal tersebut, dibuat 2 fungsi untuk membantu mendefinisikan sebuah segitiga sebagai berikut:

Fungsi setGeometry

```
// Fill the buffer with the values that define a triangle.
function setGeometry(gl) {
  gl.bufferData(
      gl.ARRAY_BUFFER,
      new Float32Array([
             0, -100,
           150,  125,
          -175,  100]),
      gl.STATIC_DRAW);
}
```

Fungsi drawScene

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

Perlu diperhatikan pada fungsi setGeometry, didefinisikan 3 buah koordinat dalam sebuah buffer yang mewakilkan posisi dari setiap titik segitiga yang akan digambar. Selain itu, didefinisikan juga pada fungsi drawScene bentuk segitiga dalam pemanggilan konstanta gl.TRIANGLES dan deklarasi jumlah sudut pada variabel count = 3.

Latihan berikutnya adalah membentuk lebih dari 1 segitiga dan memberikan warna yang konstan. Dilakukan perubahan untuk fungsi setGeometry, drawScene, dan penambahan fungsi baru setColors sebagai berikut:

Fungsi setGeometry

```
// Fill the buffer with the values that define a triangle.
function setGeometry(gl) {
  gl.bufferData(
      gl.ARRAY_BUFFER,
      new Float32Array([
          -150, -100,
           150, -100,
          -150,  100,
           150, -100,
          -150,  100,
           150,  100]),
      gl.STATIC_DRAW);
}
```

Fungsi drawScene

```
// Draw the scene.
function drawScene() {
  ...
```

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

Fungsi setColors

```
// Fill the buffer with colors for the 2 triangles
// that make the rectangle.
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);
}
```

Perhatikan bahwa pada kasus ini, terdapat 6 koordinat titik pada buffer untuk mewakilkan keenam titik dari 2 segitiga yang dibentuk. Jumlah sudut segitiga juga berubah menjadi 6 yang didefinisikan pada variabel count = 6 pada fungsi drawScene. Untuk menambahkan warna, ditambahkan pula buffer yang berisi warna pada setiap sudut segitiga dalam format RGBA pada fungsi setColors. Semua fungsi tersebut nantinya akan dipanggil dalam program utama dan kemudian di-bind dengan objek canvas.

Latihan yang terakhir adalah membuat gradasi warna pada objek. Caranya adalah dengan menambahkan nilai yang berbeda di setiap titik dari sudut-sudut objek 2 dimensi yang didefinisikan sebelumnya. Untuk merealisasikannya, dilakukan perubahan terhadap fungsi setColors sebagai berikut:

Fungsi setColors

```
// Fill the buffer with colors for the 2 triangles
// that make the rectangle.
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);
}
```

Perlu diperhatikan bahwa pada kasus ini, setiap sudut diberikan nilai RGBA yang unik secara random, sehingga terjadi perbedaan warna di sudut-sudut segitiga. Hal ini membuat gradasi warna secara otomatis pada area dalam segitiga. Jika sudah mencapai tahap ini, maka hasil program akan tampak seperti gambar berikut:

2. Image Processing

Berikutnya, WebGL Fundamentals menjelaskan tentang image processing, mulai dari cara menambahkan sebuah gambar hingga memanipulasi sebuah gambar. Untuk menambahkan sebuah gambar ke dalam canvas, cukup memanggil link reference gambar tersebut dan memasukkannya ke dalam fungsi render yang berisi:

Fungsi render

```
function render(image) {
  ...
  // all the code we had before.
  ...
  // look up where the texture coordinates need to go.
  var texCoordLocation = gl.getAttribLocation(program, "a_texCoord");

  // provide texture coordinates for the rectangle.
  var texCoordBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, texCoordBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([
      0.0,  0.0,
      1.0,  0.0,
      0.0,  1.0,
      0.0,  1.0,
      1.0,  0.0,
      1.0,  1.0]), gl.STATIC_DRAW);
  gl.enableVertexAttribArray(texCoordLocation);
  gl.vertexAttribPointer(texCoordLocation, 2, gl.FLOAT, false, 0, 0);

  // Create a texture.
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);

  // Set the parameters so we can render any size image.
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);

  // Upload the image into the texture.
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
  ...
}
```

Perhatikan bahwa untuk menambahkan sebuah gambar, kita perlu membuat sebuah texture dengan method createTexture dan meletakkan gambar pada texture tersebut dengan method texImage2D.

Catatan penting jika menjalankan program secara local, maka program harus dijalankan dalam web server agar WebGL dapat memuat gambar.

Kemudian WebGL mencoba memanipulasi gambar dengan cara menukar warna merah dengan biru dan melakukan blur pada gambar. Ditambahkan potongan kode berikut ke file html:

Variabel gl_FragColor pada file html

```
...
gl_FragColor = texture2D(u_image, v_texCoord).bgra;
...
```

Fungsi main pada file html

```
void main() {
   // compute 1 pixel in texture coordinates.
   vec2 onePixel = vec2(1.0, 1.0) / u_textureSize;

   // average the left, middle, and right pixels.
   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;
}
```

Untuk menukar warna merah dengan biru, cukup mengubah format warna dari rgba menjadi bgra, sedangkan untuk melakukan blur, diambil nilai warna rata-rata dari setiap pixel pada gambar dengan nilai warna pixel di sebelah kanan dan kirinya. Semakin kecil jarak nilai warna pada setiap pixel, maka gambar akan semakin blur.

Sebagai latihan terakhir, WebGL melakukan manipulasi dengan berbagai metode, seperti blur, sharpen, dan edgeDetect. Manipulasi ini dilakukan dengan cara mengubah nilai warna dari setiap pixel dengan 8 pixel tetangganya dalam rentang matriks 3x3. Terdapat beberapa preset nilai matriks untuk setiap metode manipulasi. Untuk merealisasikannya, fungsi main pada file html diubah agar dapat menerima matriks-matriks preset yang didefiniskan pada file js yang kemudian dikalkulasi untuk setiap pixel dalam rentang 3x3 sebagai berikut:

Fungsi main pada file html

```
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] ;

   // Divide the sum by the weight but just use rgb
   // we'll set alpha to 1.0
   gl_FragColor = vec4((colorSum / u_kernelWeight).rgb, 1.0);
}
```

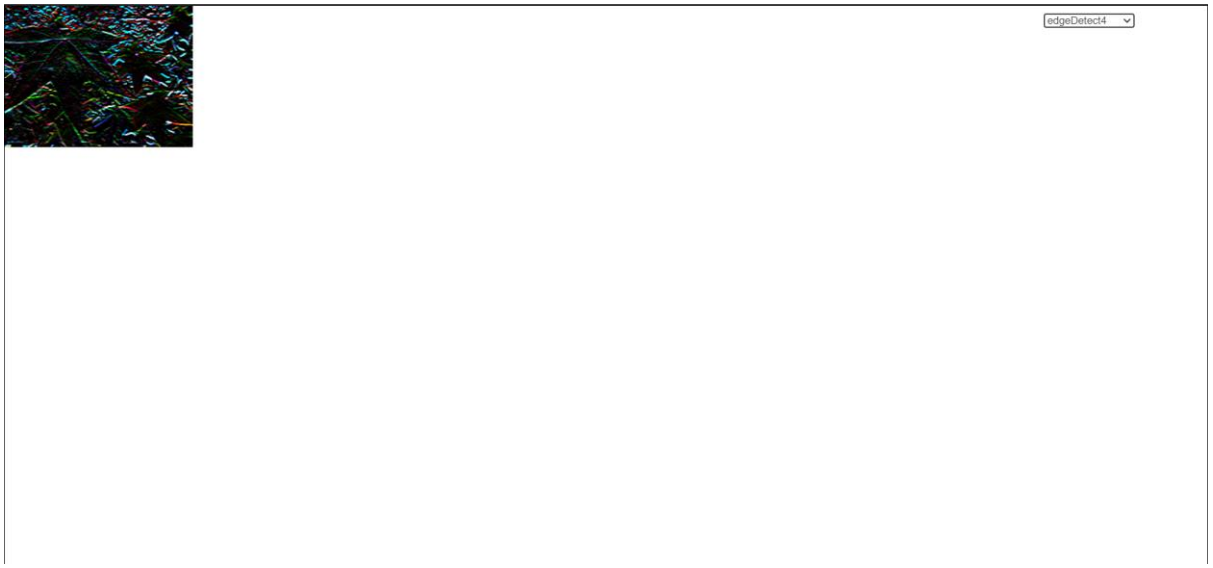
Variabel kernels pada file js

```
  // Define several convolution kernels
  var kernels = {
    normal: [
      0, 0, 0,
      0, 1, 0,
      0, 0, 0
    ],
    gaussianBlur: [
      0.045, 0.122, 0.045,
      0.122, 0.332, 0.122,
```

```
      0.045, 0.122, 0.045
    ],
    gaussianBlur2: [
      1, 2, 1,
      2, 4, 2,
      1, 2, 1
    ],
    gaussianBlur3: [
      0, 1, 0,
      1, 1, 1,
      0, 1, 0
    ],
    unsharpen: [
      -1, -1, -1,
      -1,  9, -1,
      -1, -1, -1
    ],
    sharpness: [
       0,-1, 0,
      -1, 5,-1,
       0,-1, 0
    ],
    sharpen: [
       -1, -1, -1,
       -1, 16, -1,
       -1, -1, -1
    ],
    edgeDetect: [
       -0.125, -0.125, -0.125,
       -0.125,  1,     -0.125,
       -0.125, -0.125, -0.125
    ],
    edgeDetect2: [
       -1, -1, -1,
       -1,  8, -1,
       -1, -1, -1
    ],
    edgeDetect3: [
       -5, 0, 0,
        0, 0, 0,
        0, 0, 5
    ],
    edgeDetect4: [
       -1, -1, -1,
        0,  0,  0,
        1,  1,  1
    ],
    edgeDetect5: [
       -1, -1, -1,
        2,  2,  2,
       -1, -1, -1
    ],
    edgeDetect6: [
       -5, -5, -5,
       -5, 39, -5,
       -5, -5, -5
    ],
    sobelHorizontal: [
        1,  2,  1,
        0,  0,  0,
       -1, -2, -1
    ],
    sobelVertical: [
        1,  0, -1,
        2,  0, -2,
        1,  0, -1
    ],
    previtHorizontal: [
```

```
        1,  1,  1,
        0,  0,  0,
       -1, -1, -1
    ],
    previtVertical: [
        1,  0, -1,
        1,  0, -1,
        1,  0, -1
    ],
    boxBlur: [
        0.111, 0.111, 0.111,
        0.111, 0.111, 0.111,
        0.111, 0.111, 0.111
    ],
    triangleBlur: [
        0.0625, 0.125, 0.0625,
        0.125,  0.25,  0.125,
        0.0625, 0.125, 0.0625
    ],
    emboss: [
       -2, -1,  0,
       -1,  1,  1,
        0,  1,  2
    ]
};
```

Pada fungsi main, dilakukan manipulasi muali dari pixel di posisi kiri bawah hingga posisi kanan atas dengan cara dikalikan dengan nilai matriks yang dipilih untuk setiap posisi yang bersesuaian. Terdapat 20 nilai preset matriks berbeda yang didefinisikan pada file js. Hasil akhir dari latihan ini dapat dilihat pada gambar berikut:



3. Link Repository dan Video

https://github.com/ryorichardo/Tugas-Grafkom

https://www.youtube.com/watch?v=WkxCdMPeOrI