Laporan Tugas Grafika Komputer – webGL Image Processing

Clarisa Natalia Edelin

13519213

Modul WebGL Image Processing merupakan modul yang menjelaskan mengenai cara sebuah program untuk memproses sebuah gambar, dimana program akan mendeteksi beberapa hal dari gambar yang kita berikan dan program akan menggambarkan ulang sesuai dengan ketentuan yang kita berikan. Gambar yang diproses merupakan gambar yang kita input ke dalam program Javascript. Untuk dapat menghasilkan sebuah gambar, WebGL perlu menggunakan tekstur, dimana WebGL akan membutuhkan koordinat tekstur saat membaca tekstur sebuah gambar.

Pada program HTML, kita akan mendeklarasikan vertex shader dan fragment shader. Namun kita perlu menambahkan atribut baru untuk menyampaikan informasi koordinat tekstur dan menyampaikannya ke fragment shader. Atribut baru pada verxtex shader adalah v_texCoord.

```
<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() {
      vec2 zeroToOne = a_position / u_resolution;

      vec2 zeroToTwo = zeroToOne * 2.0;

      vec2 clipSpace = zeroToTwo - 1.0;

      gl_Position = vec4(clipSpace * vec2(1, -1), 0, 1);

      v_texCoord = a_texCoord;
   }
   </script>
```

Pada fragment shader kita juga perlu menambahkan atribut terkait dengan tekstur agar dapat mencari warna dari tekstur gambar. Variabel – variabel yang berhubungn dengan tekstur gambar adalah uniform sampler2D u_image, varying vec2 v_texCoord, dan gl_FragColor.

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

```
   uniform sampler2D u_image;
   uniform vec2 u_textureSize;
   uniform float u_kernel[9];
   uniform float u_kernelWeight;

   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>
```

Fungsi main pada file Javascript perlu memuat sebuah gambar, yaitu pada variabel image, untuk diproses oleh program. Sumber gambar yang diperlukan didapat dari image.src. Lalu gambar akan diproses dengan image.onload.

```
function main() {
  var image = new Image();
  image.src = "image2.jpg";
  image.onload = function() {
    render(image);
  };
}
```

Proses render gambar akan dilakukan pada fungsi render dengan parameter image yang tadi kita muat di fungsi main. Kita perlu menghubungkan antara file Javascript dengan canvas pada file HTML dengan menggunakan variabel canvas. Selain itu, kita juga membuat sebuah program untuk menggambarkan bentuk dari vertex dan fragment shader. Posisi dan tekstur disimpan dalam variabel positionLocation dan texcoordLocation.

```
var canvas = document.querySelector("#canvas");
```

```
var program = webglUtils.createProgramFromScripts(gl, ["vertex-
shader-2d", "fragment-shader-2d"]);

var positionLocation = gl.getAttribLocation(program,
"a_position");
  var texcoordLocation = gl.getAttribLocation(program,
"a_texCoord");
```

Selanjutnya kita akan membuat buffer dari posisi dari gambar yang kita muat ke dalam program. Kita juga mengikat antara positionBuffer dengan array buffer dan digunakan untuk membuat sebuah persegi panjang dengan ketentuan tertentu. Fungsi setRectangle digunakan untuk membuat sebuah persegi panjang tempat gambar yang kita muat akan digambar dan memiliki ukuran yang sama dengan gambar kita

```
  var positionBuffer = gl.createBuffer();

  gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);

  setRectangle( gl, 0, 0, image.width, image.height);
```

Langkah berikutnya akan berfokus pada tekstur sebuah gambar. Pertama kita akan mencari dimana koordinat tekstur diperlukan, yaitu di dalam variabel texCoordLocation. Kemudian kita akan memberikan informasi koordinat tekstur untuk sebuah persegi panjang pada variabel texCoordBuffer, dimana data didapatkan dari bufferData. Lalu kita akan membuat variabel texture untuk membuat sebuah tekstur dengan memanggil createTexture dan melakukan bindTexture antara texture 2D dengan texture yang telah kita buat. Fungsi texParameter merupakan fungsi yang digunakan untuk merender berbagai jenis ukuran gambar dan fungsi texImage2D digunakan untuk memuat gambar yang telah kita pilih menjadi sebuah tekstur.

```
var texCoordLocation = gl.getAttribLocation(program,
"a_texCoord");
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);

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

  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);

  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
gl.UNSIGNED_BYTE, image);
```

Kernel merupakan sebuah matriks yang digunakan untuk membuat tekstur sebuah gamar menjadi lebih kabur, tajam, timbul, dan lain – lain. Hal ini dilakukan dengan cara *convolution*. Pada program ini, terdapat 10 jenis matriks yang dapat digunakan untuk covolution sebuah gambar. Convolution akan mejumlah setiap elemen yang ada pada gambar kepada tetangga lokalnya yang telah diukur dari kernel.

```
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
    ]
  };
```

Fungsi computeKernelWeight merupakan sebuah fungsi yang digunakan untuk menghitung berat dari kernel yang berada di sekitar sebuah pixel.

```
function computeKernelWeight(kernel) {
    var weight = kernel.reduce(function(prev, curr) {
        return prev + curr;
    });
    return weight <= 0 ? 1 : weight;
  }
```

Fungsi drawWithKernel merupakan fungsi yang digunakan oleh program kita untuk menggambarkan gambar yang telah kita muat sesuai dengan pilihan kernel yang dipilih. Proses awal penggambaran hampir sama dengan proses menggambar persegi panjang pada modul sebelumnya, dimana proses dimulai dengan proses mengubah clip space menjadi pixel, pembersihan canvas, dan penggunaan program yang telah kita buat sebelumnya. Lalu dilanjutkan dengan proses penetapan koordinat posisi dan tekstur dengan menggunakan positionLocation dan texcoordLocation. Berikutnya program akan menentukan resolusi, ukuran, kernel beserta dengan berat yang telah dihitung dari gambar yang kita pilih. Langkah terakhir adalah menggambar gambar yang telah diproses dengan menggunakan drawArrays yang memiliki parameter primitiveType berupa segitiga, offset sebesar 0, dan count sebesar 6.

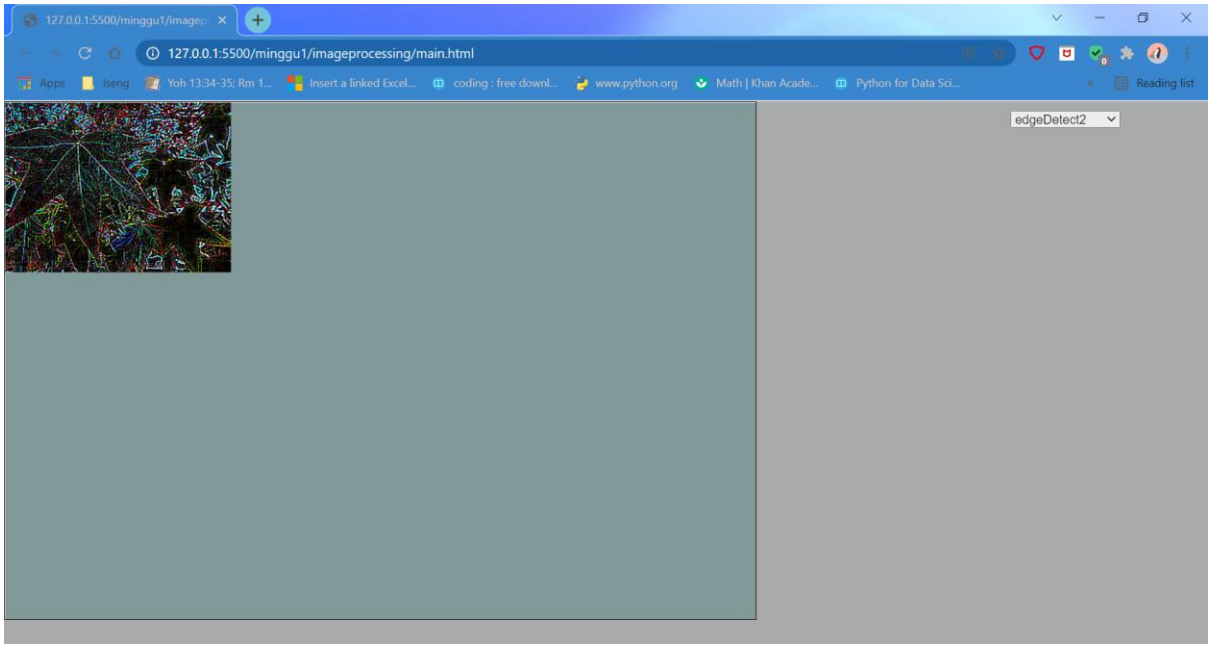
```
  function drawWithKernel(name) {
    webglUtils.resizeCanvasToDisplaySize(gl.canvas);

    gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);

    gl.clearColor(0.5, 0.6, 0.6, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    gl.useProgram(program);

    gl.enableVertexAttribArray(positionLocation);

    gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);

    var size = 2;          // 2 components per iteration
    var type = gl.FLOAT;   // the data is 32bit floats
    var normalize = false; // don't normalize the data
    var stride = 0;        // 0 = move forward size * sizeof(type)
each iteration to get the next position
    var offset = 0;        // start at the beginning of the buffer
    gl.vertexAttribPointer(
        positionLocation, size, type, normalize, stride, offset);

    gl.enableVertexAttribArray(texcoordLocation);

    gl.bindBuffer(gl.ARRAY_BUFFER, texcoordBuffer);

    var size = 2;          // 2 components per iteration
    var type = gl.FLOAT;   // the data is 32bit floats
    var normalize = false; // don't normalize the data
```

```
    var stride = 0;        // 0 = move forward size * sizeof(type)
each iteration to get the next position
    var offset = 0;        // start at the beginning of the buffer
    gl.vertexAttribPointer(
        texcoordLocation, size, type, normalize, stride, offset);


    gl.uniform2f(resolutionLocation, gl.canvas.width,
gl.canvas.height);

    gl.uniform2f(textureSizeLocation, image.width, image.height);


    gl.uniform1fv(kernelLocation, kernels[name]);
    gl.uniform1f(kernelWeightLocation,
computeKernelWeight(kernels[name]));


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

Fungsi setRectangle merupakan sebuah fungsi yang digunakan untuk membuat persegi panjang pada fungsi render. Fungsi ini mendeklarasikan titik koordinat sebuah persegi panjang berikut dengan data pasangan koordinat yang akan membentuk sebuah sisi.

```
function setRectangle(gl, x, y, width, height) {
  var x1 = x;
  var x2 = x + width;
  var y1 = y;
  var y2 = y + height;
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([
     x1, y1,
     x2, y1,
     x1, y2,
     x1, y2,
     x2, y1,
     x2, y2,
  ]), gl.STATIC_DRAW);
}
```

Berikut ini merupakan foto hasil program dijalankan.



Link Youtube : https://youtu.be/5H_-YsJ1NkY

Link repository Git : https://github.com/ClarisaNatalia/WebGL-ImageProcessing