

Tugas IF3260 Grafika Komputer

Web GL Fundamentals - Manipulation and Image Processing



Nama: Leonard Matheus

NIM: 13519215

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

A. Manipulation - How it Works

```
<!-- vertex shader -->
<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>
```

Pada vertex shader, didefinisikan matriks yang digunakan sebagai faktor pengali supaya representasi piksel yang ada di posisi tertentu dapat disematkan pada canvas

```
<!-- 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>
```

Fragment shader digunakan untuk mengisi warna pada permukaan akibat ada vector yang tidak didefinisikan pixel pembentuknya.

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

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

Pada context di atas, didefinisikan webgl pada canvas dengan membuat program yang akan mengimplementasikan shader dan fragment di atas canvas.

```

// look up where the vertex data needs to go.
var positionLocation = gl.getAttribLocation(program, "a_position");
var colorLocation = gl.getAttribLocation(program, "a_color");

// lookup uniforms
var matrixLocation = gl.getUniformLocation(program, "u_matrix");

// Create a buffer for the positions.
var positionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
// Set Geometry.
setGeometry(gl);

// Create a buffer for the colors.
var colorBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
// Set the colors.
setColors(gl);

var translation = [200, 150];
var angleInRadians = 0;
var scale = [1, 1];

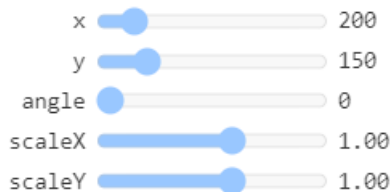
drawScene();

```

Kita akan assign variable untuk `a_position` dan `a_color` supaya vertex mengerti dimana operasi akan dilakukan di canvas. Matrix juga perlu ditentukan supaya operasi manipulasi lebih mudah untuk dilakukan. Selanjutnya, kita akan bind buffer pada posisi yang bersangkutan. Buffer ini berguna untuk menyiapkan wadah tempat dilakukannya operasi manipulasi WebGL. Buffer ada dua buah, yakni `array_buffer` dan `color_buffer`. Array buffer dapat digunakan untuk position canvas, sedangkan Color Buffer digunakan untuk mengisi shader yang ada pada canvas. Translation berguna untuk mendefinisikan ukuran shape yang akan dimanipulasi, `angleInRadians` berfungsi untuk mendefinisikan sudut awal shape yang akan dimanipulasi.

```
// Setup a ui. You, 2 days ago • adding project hiw
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,
});
```

Sebagai tambahan, fungsi ini digunakan sebagai slider yang akan menambah fitur pada webgl untuk manipulasi shape seperti ini



x 200
y 150
angle 0
scaleX 1.00
scaleY 1.00

```
function updatePosition(index) {
  return function (event, ui) {
    translation[index] = ui.value;
    drawScene();
  };
}

function updateAngle(event, ui) {
  var angleInDegrees = 360 - ui.value;
  angleInRadians = (angleInDegrees * Math.PI) / 180;
  drawScene();
}

function updateScale(index) {
  return function (event, ui) {
    scale[index] = ui.value;
    drawScene();
  };
}
```

Fungsi update ini akan disematkan pada slider yang berfungsi sebagai alat manipulasi shape. Penggunaan ada pada tampilan sebelumnya

```
// Draw the scene.
function drawScene() {
    webglUtils.resizeCanvasToDisplaySize(gl.canvas);

    // Tell WebGL how to convert from clip space to pixels
    gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);

    // Clear the canvas.
    gl.clear(gl.COLOR_BUFFER_BIT);

    // Tell it to use our program (pair of shaders)
    gl.useProgram(program);

    // Turn on the position attribute
    gl.enableVertexAttribArray(positionLocation);

    // Bind the position buffer.
    gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);

    // Tell the position attribute how to get data out of positionBuffer (ARRAY_BUFFER)
    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
    );
};
```

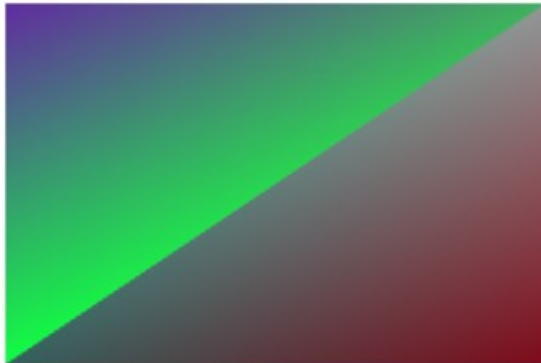
Draw scene akan berfungsi untuk mengeksekusi perilaku canvas pada web. Setelah itu clip space yang didefinisikan pada setiap sudut canvas akan dikonversi menjadi pixel menjadi lebih jelas. Untuk manajemen memory, kita akan clear canvasnya, lalu menggunakan program. Untuk persiapan iterasi, kita akan menentukan offset, ukuran size per iterasi, dan tipe data yang digunakan.

```
// Turn on the color attribute
gl.enableVertexAttribArray(colorLocation);

// Bind the color buffer.
gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);

// Tell the color attribute how to get data out of colorBuffer (ARRAY_BUFFER)
var size = 4; // 4 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(
    colorLocation,
    size,
    type,
    normalize,
    stride,
    offset
);
```


setColor ini akan mengisi buffer dengan warna dari 2 segitiga yang saling berimpitan. Untuk 3 baris pertama untuk segitiga pertama, dan 3 baris kedua untuk segitiga kedua. Random disini mengartikan warna yang akan muncul setelah browser direfresh akan terus berganti.



Hasilnya akan terlihat seperti di atas ini.

B. Image Processing

```
<!-- vertex shader --> You, 2 days ago • adding improc
<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() {
    // convert the rectangle from pixels to 0.0 to 1.0
    vec2 zeroToOne = a_position / u_resolution;

    // convert from 0->1 to 0->2
    vec2 zeroToTwo = zeroToOne * 2.0;

    // convert from 0->2 to -1->+1 (clipSpace)
    vec2 clipSpace = zeroToTwo - 1.0;

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

    // pass the texCoord to the fragment shader
    // The GPU will interpolate this value between points.
    v_texCoord = a_texCoord;
  }
</script>
```

Vertex akan dibuat dari skala 0 ke 1 dari hasil proporsi posisi koordinat pixel terhadap resolusi canvas. Setelah itu, skala diubah menjadi 0 ke 2 dan dikurang 1 sehingga menjadi -1 ke 1. Skala ini merupakan representasi clipSpace yang nantinya akan dipakai sebagai gl_Position pada manipulasi gambar tertentu untuk dilakukan interpolasi.

```

<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>

```

Pada fragment-shader, digunakan untuk mendefinisikan gambar beserta ukuran kernelnya. Karena kernel berukuran 9 kotak, maka akan dilakukan manipulasi sehingga colorsum yang tercipta pada gambar dapat dilihat pada rumus diatas.

```

function main() {
    var image = new Image();
    image.src =
        "https://asset.kompas.com/crops/glj1Zo-8XM4S0gLEBU7a_zLeyxE=/5x5:2240x1495/750x500/data/photo/2020/05/27/5ecdd9dd4df19.jpg";
    image.crossOrigin = "anonymous";
    image.width = 300;
    image.height = 200;
    image.onload = function () {
        render(image);
    };
}

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

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

    // look up where the vertex data needs to go.
    var positionLocation = gl.getAttribLocation(program, "a_position");
    var texcoordLocation = gl.getAttribLocation(program, "a_texCoord");
}

```

Pada fungsi main, akan digunakan untuk mendefinisikan gambar yang ada pada layar. Ukuran yang didefinisikan yaitu 300x200 pixel. Pada context di atas, didefinisikan

webgl pada canvas dengan membuat program yang akan mengimplementasikan shader dan fragment di atas canvas.

```
// Create a buffer to put three 2d clip space points in
var positionBuffer = gl.createBuffer();
// Bind it to ARRAY_BUFFER (think of it as ARRAY_BUFFER = positionBuffer)
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
// Set a rectangle the same size as the image.
setRectangle(gl, 0, 0, image.width, image.height);

// 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
);

// 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);

// lookup uniforms
var resolutionLocation = gl.getUniformLocation(program, "u_resolution");
var textureSizeLocation = gl.getUniformLocation(program, "u_textureSize");
var kernelLocation = gl.getUniformLocation(program, "u_kernel[0]");
var kernelWeightLocation = gl.getUniformLocation(program, "u_kernelWeight");
```

Data akan dibind ke dalam buffer supaya canvas akan berisi data yang siap dimanipulasi. Karena data yang diolah bukan berupa titik/garis, melainkan gambar, amak harus didefinisikan parameter pendukung supaya berbagai gambar dapat diolah. Setelah itu, gambar akan diubah menjadi texture.

```
// 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],
};
var initialSelection = "edgeDetect2";
```

Fungsi ini merupakan kernel yang akan didefinisikan untuk melakukan pengolahan citra. Kernel ini nantinya akan diiterasi pada seluruh gambar untuk menghasilkan gambar yang berbeda dari aslinya.

```

// Setup UI to pick kernels.
var ui = document.querySelector("#ui");
var select = document.createElement("select");
for (var name in kernels) {
    var option = document.createElement("option");
    option.value = name;
    if (name === initialSelection) {
        option.selected = true;
    }
    option.appendChild(document.createTextNode(name));
    select.appendChild(option);
}
select.onchange = function (event) {
    drawWithKernel(this.options[this.selectedIndex].value);
};
ui.appendChild(select);
drawWithKernel(initialSelection);

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

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

    // Tell WebGL how to convert from clip space to pixels
    gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);

    // Clear the canvas
    gl.clearColor(0, 0, 0, 0);
    gl.clear(gl.COLOR_BUFFER_BIT);
}

```

Akan terdapat pilihan pengolahan citra yang berbeda-beda. Compute Kernel Weight berfungsi untuk mengembalikan bobot perkalian antara angka dalam kernel dengan representasi citra yang asli. Selanjutnya, webgl akan diberi arahan dalam bentuk viewport sehingga bisa meresize dengan baik.

```

// Tell the position attribute how to get data out of positionBuffer (ARRAY_BUFFER)
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
);

// Turn on the texcoord attribute
gl.enableVertexAttribArray(texcoordLocation);

// bind the texcoord buffer.
gl.bindBuffer(gl.ARRAY_BUFFER, texcoordBuffer);

// Tell the texcoord attribute how to get data out of texcoordBuffer (ARRAY_BUFFER)
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
);

```

Buffer akan didefinisikan baik pada arraybuffer sebanyak 2 ukuran per iterasi. Tipe data float akan digunakan juga untuk melakukan manipulasi dalam skala decimal.

```

    // set the resolution
    gl.uniform2f(resolutionLocation, gl.canvas.width, gl.canvas.height);

    // set the size of the image
    gl.uniform2f(textureSizeLocation, image.width, image.height);

    // set the kernel and it's weight
    gl.uniform1fv(kernelLocation, kernels[name]);
    gl.uniform1f(kernelWeightLocation, computeKernelWeight(kernels[name]));

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

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, x2, y2]),
        gl.STATIC_DRAW
    );
}

main();

```

Selain itu, image juga akan diset ukurannya agar sesuai dengan ukuran texture yang didefinisikan di awal. Setelah itu, kernel weight dan location akan ditampilkan di sana untuk melakukan pemrosesan gambar. Terakhir, setRectangle berguna untuk menggambar buffer ke dalam canvas sehingga gambar yang telah diolah dapat ditampilkan di sana juga.



Ini adalah contoh hasil pengolahan unsharpen dengan citra Plaza Widya Nusantara

LINK VIDEO:

<https://youtu.be/wq6uEg1Vnao>

LINK REPOSITORY

https://github.com/leomatt547/webgl_hiw_improc