Tugas 1 IF3260 Grafika Komputer

Manipulasi dan Image Processing

oleh:

13519221 Allief Nuriman

Institut Teknologi Bandung

2022

A. Source Code Project 1: Manipulation

Berikut adalah *source code* yang digunakan untuk project 1 Image Manipulation. Repository untuk *source code* dapat dilihat pada tautan berikut: https://github.com/alliefn/grafkom-how-it-works. File dipecah menjadi beberapa bagian untuk diberikan penjelasan untuk sub-bagian tertentu.

index.html

```
<!-- Licensed under a BSD license. See license.html for license -->
<!DOCTYPE html>
<html>
```

```
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0,
user-scalable=yes">
<title>WebGL - Rectangle with 2 colors using bytes</title>
<link type="text/css"
href="https://webglfundamentals.org/webgl/resources/webgl-tutorials.css"
rel="stylesheet" />
</head>
```

Bagian "head" dari HTML (Untuk judul, dan metadatanya)

```
<body>
<div class="description">
Drag sliders to translate, rotate, and scale.
</div>
<canvas id="canvas"></canvas>
<div id="uiContainer">
  <div id="ui">
    <div id="x"></div>
    <div id="y"></div>
    <div id="angle"></div>
    <div id="scaleX"></div>
    <div id="scaleY"></div>
  </div>
</div>
</body>
```

Body dari HTML, disini didefinisikan canvas yang akan digunakan, pada body juga terdapat section untuk melakukan manipulasi pada canvas/image (Lihat div dengan id "uiContainer")

```
<!-- 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>
```

Fungsi vertex shader, ini digunakan untuk menghitung (menentukan) posisi "vertex" (Titik sudut)

```
<!-- 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 ialah fungsi untuk menentukan warna dari setiap pixel yang dibentuk oleh vertex shader

```
<!--
for most samples webgl-utils only provides shader compiling/linking and
canvas resizing because why clutter the examples with code that's the same
in every sample.
See https://webglfundamentals.org/webgl/lessons/webgl-boilerplate.html
and
https://webglfundamentals.org/webgl/lessons/webgl-resizing-the-canvas.html
for webgl-utils, m3, m4, and webgl-lessons-ui.
-->
<script
src="https://webglfundamentals.org/webgl/resources/webgl-utils.js"></script
>
<script
src="https://webglfundamentals.org/webgl/resources/webgl-lessons-ui.js"></s
cript>
<script src="https://webglfundamentals.org/webgl/resources/m3.js"></script>
<script src="script.js"></script>
```

webgl-utils.js, webgl-lesson-ui.js, dan m3.js digunakan untuk meresize

ukuran canvas dan utilitas lainnya.
script.js ialah script untuk mengatur bagaimana image dimanipulasi, yang akan dibahas selanjutnya.

```
</html>
```

script.js

```
"use strict";

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

Ambil bagian dokumen yang bernama “canvas”, lalu periksa konteksnya, bila browser support webgl, maka !gl seharusnya bernilai false sehingga skrip selanjutnya akan dijalankan.

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

GLSL program, digunakan untuk membentuk canvasnya berdasarkan skrip vertex-shader-2d dan fragment-shader-2d

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

Kita ingin mendapatkan dimana posisi dan warna dari bentuk canvas yang telah dibuat

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

Disini kita mensuplai program GLSL dengan matrix yang berisi pixel-pixel tersebut akan dialokasikan di tempat mana saja

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

Disini kita akan membuat buffer untuk posisi-posisinya, lalu webgl di-“bind” ke dengan positionBuffernya. Buffer sendiri adalah larik dari biner-biner data yang akan diproses GPU. Umumnya berisi posisi, garis normal, tekstur koordinat, warna vertex, dll.

```
    // Set Geometry.
    setGeometry(gl);
```

Atur geometry

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

Disini kita akan membuat buffer untuk warna-warnanya, lalu webgl di-“bind” ke dengan colorBuffernya.

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

Kita mengatur elemen default untuk gambarnya, yaitu x pada 200, y pada 150, dengan derajat 0 dan skala (1,1) (dalam x, y)

```
    drawScene();
```

Kita menggambarkan ke layar apa yang sudah disiapkan di buffer

```
    // 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 });

    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();
        };
    }
```

Pada bagian di atas, kita mendefinisikan skrip untuk mengubah atribut dari gambar, yaitu posisi, *scaling*, derajat, dll.

Setiap kali terdapat perubahan sesuai dengan slider, maka canvas akan digambar ulang.

```
    // 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);

        // 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.UNSIGNED_BYTE;  // the data is 8bit unsigned bytes
        var normalize = true;         // 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);

        // Compute the matrix
        var matrix = m3.projection(gl.canvas.clientWidth,
gl.canvas.clientHeight);
        matrix = m3.translate(matrix, translation[0], translation[1]);
        matrix = m3.rotate(matrix, angleInRadians);
        matrix = m3.scale(matrix, scale[0], scale[1]);

        // Set the matrix.
        gl.uniformMatrix3fv(matrix_location, false, matrix);

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

Ini ialah fungsi drawScene(), pada fungsi ini kita mengatur ulang ukuran dari Canvas, lalu mengatur agar WebGL melakukan konversi dari clip space ke pixel, lalu mengosongkan canvasnya. Karena Canvas telah kosong, kita akan mengisi lagi dengan program (program: Kombinasi dari vertex shader dan fragment shader + GLSL) yang sudah disiapkan. Setelah itu, kita menghidupkan kembali atribut posisi yang telah didapatkan, dan melakukan

bind pada posisi buffer. Lalu, pada atribut posisi, keluarkan data dari posisi buffer. Ini digunakan agar kita mengetahui pada posisi mana gambar dimulai. Lalu, kita melakukan hal yang sama pada atribut warna. Setelah itu, kita akan menghitung matriks lalu mengatur matriksnya, kemudian menggambar bentuk dari geometrinya.

```
// Fill the buffer with the values that define a rectangle.
// Note, will put the values in whatever buffer is currently
// bound to the ARRAY_BUFFER bind point
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 setGeometry digunakan untuk mengatur buffer agar berisi nilai yang dapat menggambarkan suatu persegi panjang dengan penggambaran yang statik.

```
// Fill the buffer with colors for the 2 triangles
// that make the rectangle.
// Note, will put the values in whatever buffer is currently
// bound to the ARRAY_BUFFER bind point
function setColors(gl) {
    // Pick 2 random colors.
    var r1 = Math.random() * 256; // 0 to 255.99999
    var b1 = Math.random() * 256; // these values
    var g1 = Math.random() * 256; // will be truncated
    var r2 = Math.random() * 256; // when stored in the
    var b2 = Math.random() * 256; // Uint8Array
    var g2 = Math.random() * 256;

    gl.bufferData(
        gl.ARRAY_BUFFER,
        new Uint8Array(   // Uint8Array
            [r1, b1, g1, 255,
                r1, b1, g1, 255,
                r1, b1, g1, 255,
                r2, b2, g2, 255,
```

```
                r2, b2, g2, 255,
                r2, b2, g2, 255]),
        gl.STATIC_DRAW);
}
```

Fungsi setColor digunakan untuk mengatur buffer agar persegi panjang yang terbentuk terbagi menjadi dua bagian (Secara merata, dibagi dengan diagonal) yang berbeda warna. Dapat dillihat bahwa kita memilih dua warna acak untuk itu.

```
main();
```

B. Source Code Project 2: Image Processing

Berikut adalah *source code* yang digunakan untuk project 2 Image Processing. Repository untuk *source code* dapat dilihat pada tautan berikut: https://github.com/alliefn/grafkom-image-processing. File dipecah menjadi beberapa bagian untuk diberikan penjelasan untuk sub-bagian tertentu.

index.html

```
<!-- Licensed under a BSD license. See license.html for license -->
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0,
user-scalable=yes">
<title>WebGL - 2D image 3x3 convolution</title>
<link type="text/css"
href="https://webglfundamentals.org/webgl/resources/webgl-tutorials.css"
rel="stylesheet" />
</head>
<body>
<canvas id="canvas"></canvas>
<div id="uiContainer">
  <div id="ui"></div>
</div>
</body>
<!-- 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() {
   // 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>
```

Fungsi vertex shader, ini digunakan untuk menghitung (menentukan) posisi "vertex" (Titik sudut)

```
<!-- 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>
```

Fragment shader ialah fungsi untuk menentukan warna dari setiap pixel yang dibentuk oleh vertex shader

```
<!--
for most samples webgl-utils only provides shader compiling/linking and
canvas resizing because why clutter the examples with code that's the same
in every sample.
See https://webglfundamentals.org/webgl/lessons/webgl-boilerplate.html
and
https://webglfundamentals.org/webgl/lessons/webgl-resizing-the-canvas.html
for webgl-utils, m3, m4, and webgl-lessons-ui.
-->
<script
src="https://webglfundamentals.org/webgl/resources/webgl-utils.js"></script
>
<script src="script.js"></script>
</html>
```

script.js

```
"use strict";

function main() {
    var image = new Image();
    image.src = "img/leaves.jpg";  // MUST BE SAME DOMAIN!!!
    image.onload = function () {
        render(image);
    };
}
```

Kita akan mengambil image pada directory img/leaves.jpg untuk dirender

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

Ambil elemen canvas, lalu cek apa browser mendukung webgl, bila bisa...

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

...setup GLSL program dengan mengambil vertex-shader-2d dan fragment-shader-2d yang telah didefinisikan.

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

Ambil data ke mana vertex data akan berada (a_position dan a_texCoord).

```
    // 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);
```

Buffer untuk menaruh titik yang akan dibind ke ARRAY_BUFFER untuk mengatur ukuran persegi panjang agar seukuran image

```
    // 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);
```

Disini kita akan mempersiapkan koordinat tekstur untuk persegi panjang

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

... dan membuat teksturnya

```
    // 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);
```

Mengatur parameter sehingga apapun ukurannya, image dapat dirender

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

Menaruh image ke tekstur

```
    // 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");

    // 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';
```

Disini kita mendefinisikan convolution kernel untuk melakukan pemrosesan gambar, pertama-tama, opsi yang dipilih ialah edgeDetect2. Convolution kernel sendiri ialah matriks 3x3 dan setiap entry merepresentasikan berapa banyak yang dibutuhkan untuk mengkalikan 8 piksel disekitar piksel yang sedang direnderer.

```
    // 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);
```

Disini kita mendefinisikan UI untuk memilih kernel apa yang akan dipakai untuk memproses gambar, setiap kali ada perubahan, maka dipanggil fungsi drawWithKernel() untuk mengubah gambar.

```
    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);

        // 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);

        // 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);

        // 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);
    }
}
```

Fungsi drawwithkernel() sendiri tidak terlalu berbeda dengan fungsi drawScene() hanya saja disini buffer akan diproses dengan kernel yang terpilih untuk memproses gambar.

```
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);
}
```

Fungsi setRectangle() digunakan untuk membuat suatu persegi panjang untuk meng*contain* gambar

```
main();
```

C. Tautan Video YouTube

https://youtu.be/wbdxDRSmBHI