

**IF3260 GRAFIKA KOMPUTER
WEB GL FUNDAMENTALS
MANIPULATION AND IMAGE PROCESSING**



Disusun Oleh:

Muhammad Rayhan Ravianda 13519201

**TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2021**

How It Works

Pada modul “*How It Works*” dipelajari bagaimana WebGL bekerja. Pada modul ini terdapat 3 file yaitu index.js, index.html, dan index.css.

File index.js berisi kode sebagai berikut

```
// WebGL - Rectangle with random colors
// from https://webglfundamentals.org/webgl/webgl-2d-rectangle-with-random-colors.html

"use strict";

function main() {
    // Get A WebGL context
    /** @type {HTMLCanvasElement} */
    var canvas = document.querySelector("#canvas");
    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 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();

// 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();
    };
}

```

```

    };
}

// 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.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);

    // 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(matrixLocation, false, matrix);

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

// 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);
}

// 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) {
    // 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);
}

main();

```

Pada file ini dibuat fungsi-fungsi yang diperlukan program. Terdapat fungsi `setGeometry` yang digunakan untuk melakukan inisiasi geometri, dan `setColors` yang digunakan untuk memberi warna pada geometri. Fungsi `updatePosition`, `updateAngle`, dan `updateScale` digunakan untuk merubah state geometri baik posisi, rotasi sudut, dan skala yang nantinya akan dihubungkan dengan ui pada file `index.html`. Terdapat juga `drawScene` yang digunakan untuk menggambar geometri setelah state diubah.

File `index.html` berisi kode sebagai berikut

```

<canvas id="canvas"></canvas>
<div id="uiContainer">
    <div id="ui">
        <div id="x"></div>
        <div id="y"></div>
    </div>
</div>

```

```

    <div id="angle"></div>
    <div id="scaleX"></div>
    <div id="scaleY"></div>
  </div>
</div>

<!-- 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>

<!-- 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><!--

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"></scrip
t>
<script
src="https://webglfundamentals.org/webgl/resources/webgl-lessons-ui.js"></
script>
<script
src="https://webglfundamentals.org/webgl/resources/m3.js"></script>
```

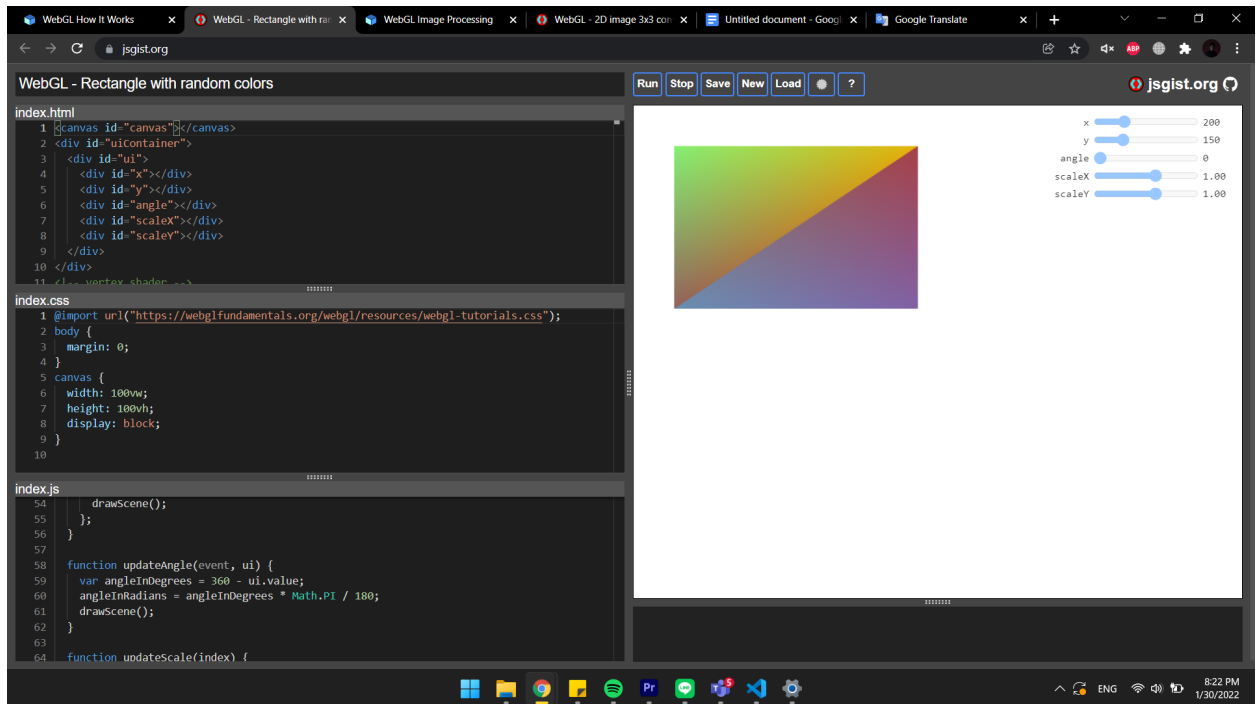
Pada file diatas ditulis kode untuk menampilkan ui yang akan digunakan user untuk merubah state geometri dan untuk menampilkan geometri ke layar.

File index.css berisi kode sebagai berikut

```
@import
url("https://webglfundamentals.org/webgl/resources/webgl-tutorials.css");
body {
  margin: 0;
}
canvas {
  width: 100vw;
  height: 100vh;
  display: block;
}
```

Pada file diatas dilakukan setup ukuran kanvas yang digunakan untuk meletakkan geometri.

Hasil keluaran setelah ketiga program dijalankan adalah sebagai berikut



Dalam program user dapat merubah state dari geometri sesuka hati dan program akan menggambarkan geometri sesuai state terkini secara real-time.

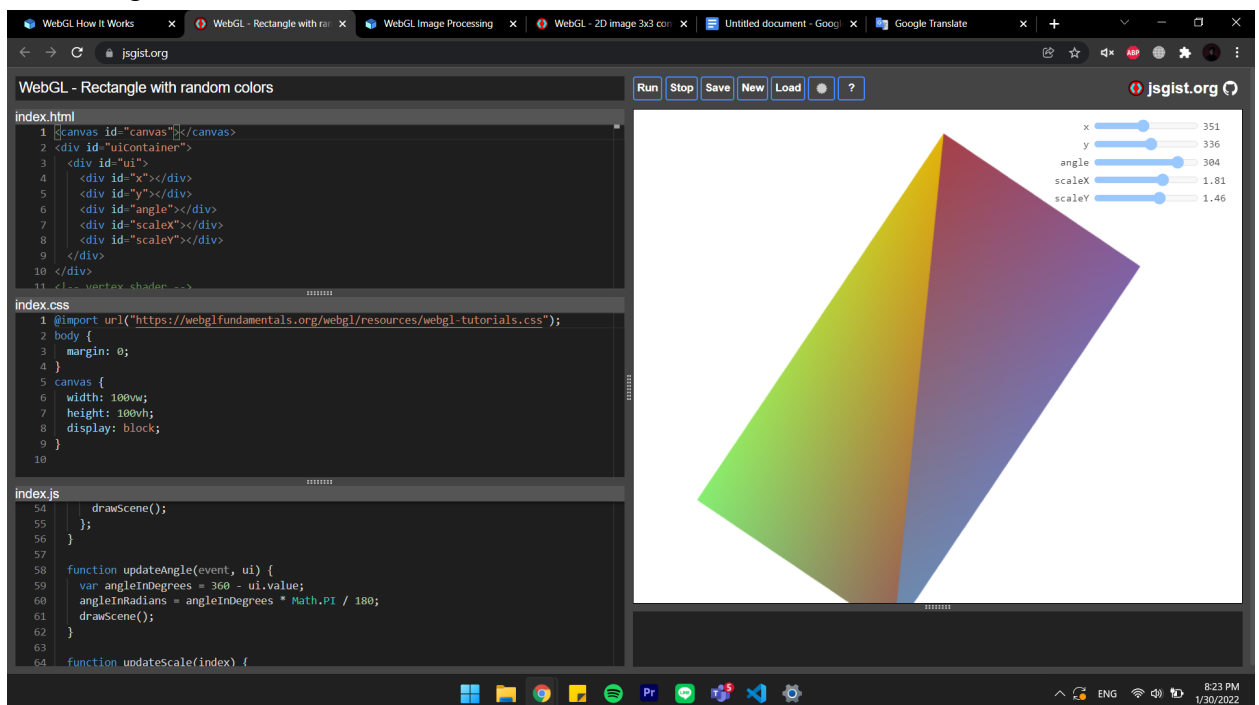


Image Processing

Pada modul “*Image Processing*” dipelajari bagaimana gambar dapat diproses pada kanvas. Gambar akan ditampilkan dan dapat dilakukan penyuntingan sederhana dengan fitur yang tersedia. Pada modul ini terdapat 3 file yaitu index.js, index.html, dan index.css.

File index.js berisi kode sebagai berikut

```
// WebGL - 2D image 3x3 convolution
// from
https://webglfundamentals.org/webgl/webgl-2d-image-3x3-convolution.html

"use strict";

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;
    }

    // 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");

        // 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");

        // 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';

// 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);

    // 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);
}
}

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);
}

main();

// This is needed if the images are not on the same domain
// NOTE: The server providing the images must give CORS permissions
// in order to be able to use the image with WebGL. Most sites
// do NOT give permission.
//
// See:
https://webglfundamentals.org/webgl/lessons/webgl-cors-permission.html
function requestCORSIfNotSameOrigin(img, url) {
    if ((new URL(url, window.location.href)).origin !==
window.location.origin) {
        img.crossOrigin = "";
    }
}

```

```
}
```

Pada file diatas ditulis fungsi-fungsi yang diperlukan oleh program. Fungsi render digunakan untuk menampilkan gambar pada layar. Terdapat fungsi setRectangle untuk melakukan setup ukuran gambar yang nantinya akan digunakan juga pada fungsi-fungsi yang akan merubah gambar dan fungsi requestCORSIIfNotSameOrgin yang digunakan untuk melakukan request CORS untuk mengunduh gambar agar gambar dapat diproses pada program.

File index.html berisi kode sebagai berikut

```
<canvas id="canvas"></canvas>
<div id="uiContainer">
  <div id="ui"></div>
</div>
<!-- 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>
```

```

<!-- 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><!--
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"></scrip
t>
```

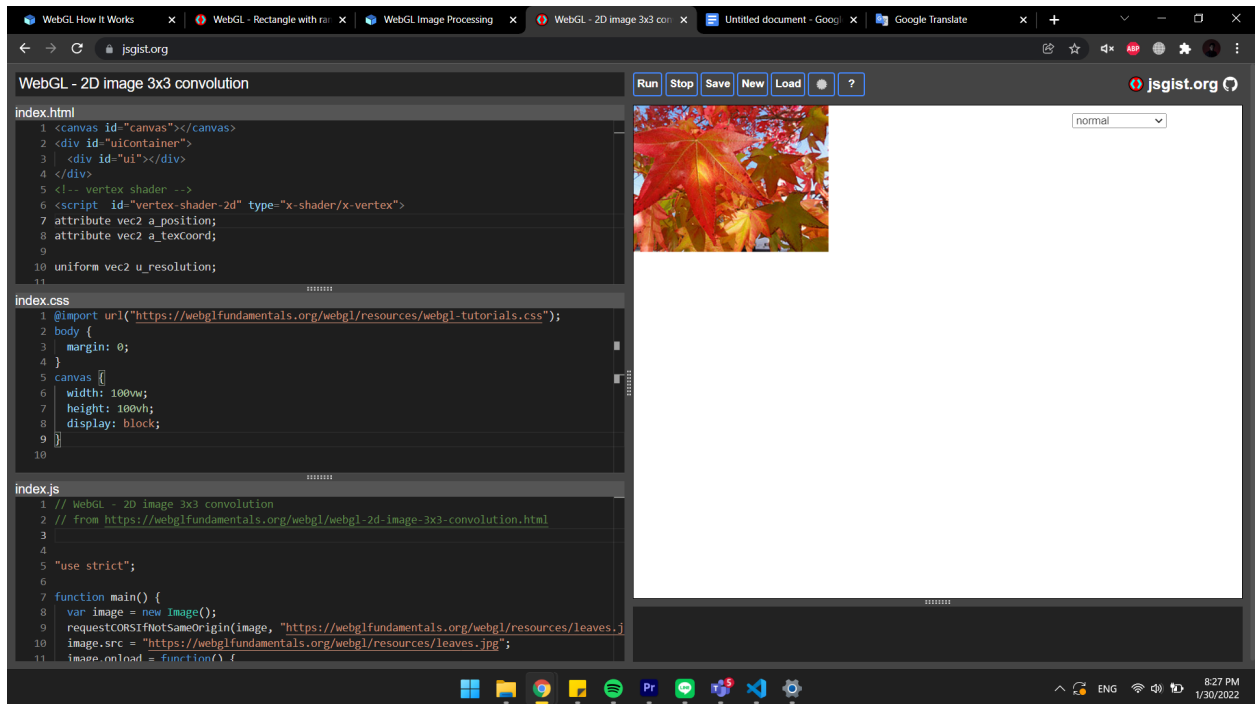
Pada file diatas ditulis kode untuk menampilkan kanvas yang nantinya akan diisi gambar dan dropdown menu yang akan digunakan user untuk memilih fitur.

File index.css berisi kode sebagai berikut

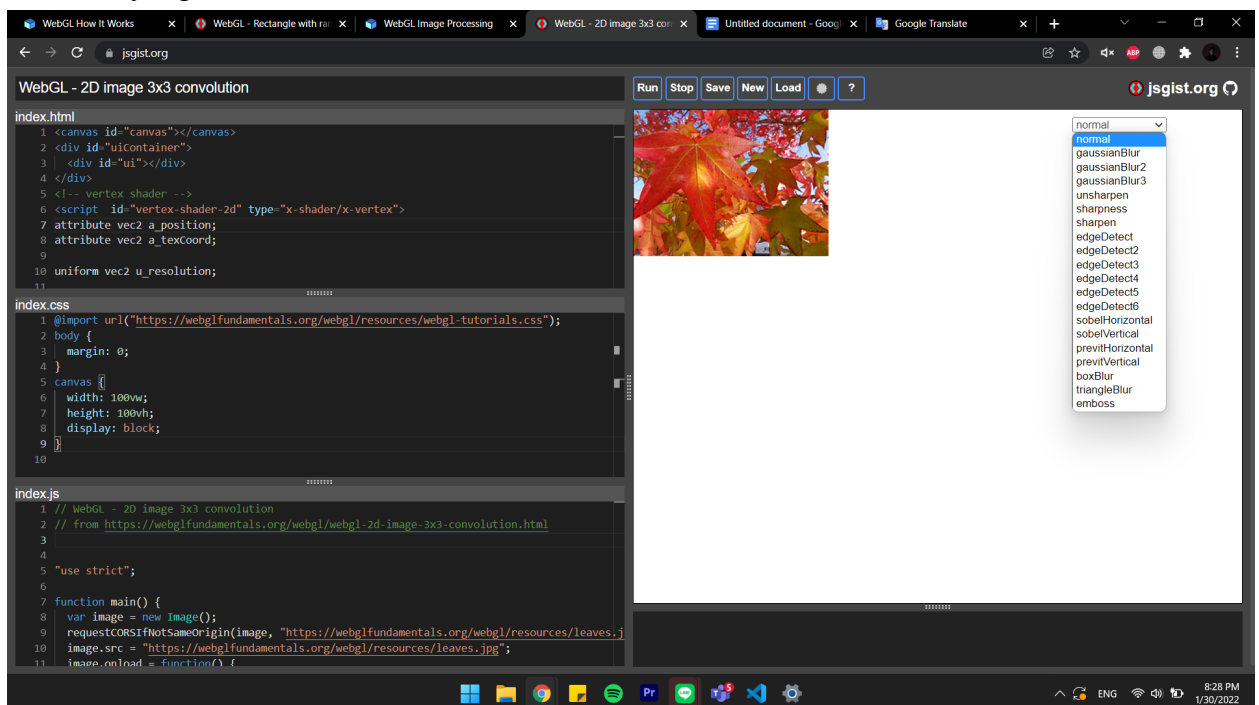
```
@import
url("https://webglfundamentals.org/webgl/resources/webgl-tutorials.css");
body {
    margin: 0;
}
canvas {
    width: 100vw;
    height: 100vh;
    display: block;
}
```

Pada file diatas dilakukan setup ukuran kanvas yang digunakan untuk meletakkan geometri.

Hasil keluaran setelah ketiga program dijalankan adalah sebagai berikut



Pada program user dapat memilih task yang ingin dilakukan terhadap gambar melalui drowdown menu yang sudah disediakan.



Setelah user memilih fitur, gambar akan menampilkan hasil gambar yang sudah diaplikasikan fitur ke layar.

WebGL - 2D image 3x3 convolution

Run Stop Save New Load ? jsglst.org

index.html

```
1 <canvas id="canvas"></canvas>
2 <div id="uicontainer">
3   <div id="ui"></div>
4 </div>
5 <!-- vertex shader -->
6 <script id="vertex-shader-2d" type="x-shader/x-vertex">
7   attribute vec2 a_position;
8   attribute vec2 a_texCoord;
9
10  uniform vec2 u_resolution;
11
```


index.css

```
1 @import url("https://webglfundamentals.org/webgl/resources/webgl-tutorials.css");
2 body {
3   margin: 0;
4 }
5 canvas {
6   width: 100vw;
7   height: 100vh;
8   display: block;
9 }
10
```

index.js

```
1 // WebGL - 2D image 3x3 convolution
2 // from https://webglfundamentals.org/webgl/webgl-2d-image-3x3-convolution.html
3
4
5 "use strict";
6
7 function main() {
8   var image = new Image();
9   requestCORSIfNotSameOrigin(image, "https://webglfundamentals.org/webgl/resources/leaves.jpg");
10  image.src = "https://webglfundamentals.org/webgl/resources/leaves.jpg";
11  image.onload = function() {
```

edgeDetect2



Pranala

Semua kebutuhan program dan file pada tugas ini dapat diakses pada <https://github.com/ravmhmd/WebGL-Fundamental.git> .

Video penjelasan singkat program dapat diakses pada <https://youtu.be/uo24nKMzggg> .

Semua program dan referensi bersumber dari <https://webglfundamentals.org/webgl/lessons/webgl-how-it-works.html> untuk modul “How It Works” dan <https://webglfundamentals.org/webgl/lessons/webgl-image-processing.html> untuk modul “Image Processing”.