

## Laporan Tugas Web GL Rotate

Nama: Muhammad Rifat Abiwardani

NIM: 13519205

Kelas: K04

### 1. Tujuan

Pada tugas ini, WebGL digunakan untuk membuat huruf 'F' yang terdiri dari tiga persegi panjang (atau enam segitiga) pada browser. Objek tersebut memiliki orientasi pada canvas yang dapat diubah menggunakan *slider angle*, serta warna random. Proyek ini dibuat tanpa mengimport fungsionalitas webgl-utils dan webgl-lessons-ui.

### 2. Setup Canvas dan GL

Berikut markup pada file index.html untuk mendefinisikan canvas dan wadah slider:

```
<canvas id="canvas"></canvas>
  <div id="uiContainer">
    <div id="ui">
      <div id="angle"></div>
    </div>
  </div>
```

Berikut kode javascript pada file rotate.js untuk mempersiapkan canvas, membuat konteks rendering WebGL, menginisialisasi objek program WebGL yang nanti digunakan untuk me-link shader dan membuat program, mendefinisikan variabel-variabel yang akan digunakan untuk mengatur letak objek grafik, melakukan binding buffer untuk mengubah letak pada waktu render, menginisialisasi objek "F", serta men-set nilai warna sebagai warna random:

```
var canvas = document.querySelector("#canvas");
var gl = canvas.getContext("webgl");
if (!gl) {
  return;
}

var program = createProgramFromScripts(gl);

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

var positionBuffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
setGeometry(gl);
var color = [Math.random(), Math.random(), Math.random(), 1];

drawScene();
```

Potongan kode berikut menginisialisasi uniform-uniform (variabel global dalam WebGL):

```
var resolutionLocation = gl.getUniformLocation(program,
"u_resolution");
var colorLocation = gl.getUniformLocation(program, "u_color");
var translationLocation = gl.getUniformLocation(program,
```

```
"u_translation");  
var rotationLocation = gl.getUniformLocation(program, "u_rotation");
```

Berikut isi indexstyles.css yang mengatur ukuran canvas:

```
body {  
    margin: 0;  
}  
canvas {  
    width: 90vw;  
    height: 90vh;  
    display: block;  
}
```

### 3. Utilities

Pada potongan kode setup GL di bagian sebelumnya, fungsi `createProgramFromScripts()` digunakan. Berikut *source code* fungsi `createProgramFromScripts()`:

```
function createProgramFromScripts(gl, opt_errorCallback) {  
    const vertexShader = loadVertexShader(gl, opt_errorCallback);  
    const fragmentShader = loadFragmentShader(gl, opt_errorCallback);  
    const shaders = [vertexShader, fragmentShader];  
    return createProgram(gl, shaders, opt_errorCallback);  
}
```

Fungsi `createProgramFromScripts()` pertama memanggil fungsi `loadVertexShader()` dan `loadFragmentShader()` untuk memperoleh objek `WebGLShader` sebagai vertex shader dan fragment shader dari program. Kemudian shader tersebut di-pass ke fungsi `createProgram()`.

Berikut *source code* fungsi `loadVertexShader()` dan `loadFragmentShader()` yang digunakan:

```
function loadVertexShader(gl, opt_errorCallback) {  
    let shaderSource = '';  
    const shaderScript = document.getElementById("vertex-shader-2d");  
    if (!shaderScript) {  
        throw ('*** Error: unknown vertex shader');  
    }  
    shaderSource = shaderScript.text;  
    const errFn = opt_errorCallback || error;  
  
    const shader = gl.createShader(gl.VERTEX_SHADER);  
    gl.shaderSource(shader, shaderSource);  
    gl.compileShader(shader);  
  
    const compiled = gl.getShaderParameter(shader,  
gl.COMPILE_STATUS);  
    if (!compiled) {  
        const lastError = gl.getShaderInfoLog(shader);  
        errFn('*** Error compiling shader \'' + shader + '\': ' +  
lastError + '\n' + shaderSource.split('\n').map((l,i) => `${i + 1}:  
${l}`).join('\n'));
```

```

        gl.deleteShader(shader);
        return null;
    }

    return shader;
}

function loadFragmentShader(gl, opt_errorCallback) {
    let shaderSource = '';
    const shaderScript =
document.getElementById("fragment-shader-2d");
    if (!shaderScript) {
        throw ('*** Error: unknown fragment shader');
    }
    shaderSource = shaderScript.text;
    const errFn = opt_errorCallback || error;

    const shader = gl.createShader(gl.FRAGMENT_SHADER);
    gl.shaderSource(shader, shaderSource);
    gl.compileShader(shader);

    const compiled = gl.getShaderParameter(shader,
gl.COMPILE_STATUS);
    if (!compiled) {
        const lastError = gl.getShaderInfoLog(shader);
        errFn('*** Error compiling shader \'' + shader + '\': ' +
lastError + '\n' + shaderSource.split('\n').map((l,i) => `${i + 1}:
${l}`).join('\n'));
        gl.deleteShader(shader);
        return null;
    }

    return shader;
}

```

Kedua fungsi memiliki kegunaan yang mirip. Pertama, fungsi menyimpan *source code* shader vertex atau fragment dalam GLSL yang sudah disiapkan di index.html dalam variabel `shaderSource`. Kemudian dibuat objek `WebGLShader` untuk masing vertex shader dan fragment shader. Method `WebGL shaderSource()` digunakan untuk menyetel *source code* objek shader sebagai kode GLSL yang sudah diperoleh, dan `compileShader()` digunakan untuk meng-*compile* kode tersebut. Jika compile berhasil, objek shader dikembalikan. Jika tidak, maka fungsi load shader melempar error.

Selain `loadVertexShader()` dan `loadFragmentShader()`, fungsi `createProgramFromScripts()` juga menggunakan fungsi `createProgram()`. Berikut *source code* fungsi `createProgram()`:

```

function error(msg) {
    if (topWindow.console) {
        if (topWindow.console.error) {
            topWindow.console.error(msg);
        } else if (topWindow.console.log) {
            topWindow.console.log(msg);
        }
    }
}

```

```

}

function createProgram(gl, shaders, opt_errorCallback) {
  const errFn = opt_errorCallback || error;
  const program = gl.createProgram();
  shaders.forEach(function(shader) {
    gl.attachShader(program, shader);
  });
  gl.linkProgram(program);

  //if not linked
  const linked = gl.getProgramParameter(program, gl.LINK_STATUS);
  if (!linked) {
    const lastError = gl.getProgramInfoLog(program);
    errFn('Error in program linking:' + lastError);

    gl.deleteProgram(program);
    return null;
  }
  return program;
}

```

Fungsi `createProgram()` pertama membuat objek program WebGL dengan method `createProgram()` dari `WebGLRenderingContext`. Vertex shader dan fragment shader di-link ke objek program tersebut. Nantinya, *interface* pengendali shader akan di-link juga ke objek program (dalam proyek ini *slider* translasi, rotasi, dan penskalaan). Kemudian, method `linkProgram()` digunakan untuk mempersiapkan GPU untuk vertex shader dan fragment shader dari proyek ini. Jika link status dari program bernilai false, maka fungsi `createProgram()` melempar error. Jika tidak, maka objek program yang berhasil diinisialisasi dikembalikan.

Saat waktu render nanti, program juga membutuhkan fungsi untuk *resize* canvas agar sesuai dengan ukuran halaman pada layar. Fungsi `resizeCanvasToDisplaySize()` berikut digunakan. Pada dasarnya, fungsi ini menghitung ukuran canvas dengan mengali multiplier dengan ukuran awal.

```

function resizeCanvasToDisplaySize(canvas, multiplier) {
  multiplier = multiplier || 1;
  const width = canvas.clientWidth * multiplier | 0;
  const height = canvas.clientHeight * multiplier | 0;
  if (canvas.width !== width || canvas.height !== height) {
    canvas.width = width;
    canvas.height = height;
    return true;
  }
  return false;
}

```

#### 4. UI

Untuk mempermudah pembuatan slider, beberapa fungsi UI digunakan.

```

function setupSlider(selector, options) {
  var parent = document.querySelector(selector);
  if (!parent) {
    return;
  }
}

```

```

    }
    if (!options.name) {
        options.name = selector.substring(1);
    }
    return createSlider(parent, options);
}

```

Fungsi `setupSlider()` mencari *container* dari *slider* yang sudah ada di halaman (nilai `selector` adalah id dari div yang mengandung masing *slider*), lalu menggunakannya dalam fungsi `createSlider()` untuk membuat komponen *slider* pada halaman.

Berikut isi fungsi `createSlider()`:

```

function getQueryParams() {
    var params = {};
    if (window.hackedParams) {
        Object.keys(window.hackedParams).forEach(function(key) {
            params[key] = window.hackedParams[key];
        });
    }
    if (window.location.search) {
        window.location.search.substring(1).split("&").forEach(function(pair) {
            {
                var keyValue = pair.split("=").map(function(kv) {
                    return decodeURIComponent(kv);
                });
                params[keyValue[0]] = keyValue[1];
            }
        });
    }
    return params;
}

function createSlider(parent, options) {
    const gopt = getQueryParams();

    var precision = options.precision || 0;
    var min = options.min || 0;
    var step = options.step || 1;
    var value = options.value || 0;
    var max = options.max || 1;
    var fn = options.slide;
    var name = gopt["ui-" + options.name] || options.name;
    var uiPrecision = options.uiPrecision === undefined ? precision :
options.uiPrecision;
    var uiMult = options.uiMult || 1;

    min /= step;
    max /= step;
    value /= step;

    parent.innerHTML = ""
    parent.innerHTML += "<div class='gman-widget-outer'>\n"
    parent.innerHTML += "    <div
class='gman-widget-label'>" + name.toString() + "</div>\n";
    parent.innerHTML += "    <div class='gman-widget-value'></div>\n"
    parent.innerHTML += "    <input class='gman-widget-slider'
type='range' min='" + min.toString() + "' max='" + max.toString() + "'

```

```

value='"+value.toString()+"' />\n";
parent.innerHTML += "</div>";
var valueElem = parent.querySelector(".gman-widget-value");
var sliderElem = parent.querySelector(".gman-widget-slider");

function updateValue(value) {
    valueElem.textContent = (value * step *
uiMult).toFixed(uiPrecision)
}

updateValue(value);

function handleChange(event) {
    var value = parseInt(event.target.value);
    updateValue(value);
    fn(event, { value: value * step });
}

sliderElem.addEventListener('input', handleChange);
sliderElem.addEventListener('change', handleChange);

return {
    elem: parent,
    updateValue: (v) => {
        v /= step;
        sliderElem.value = v;
        updateValue(v);
    },
};
}

```

Kode di atas mengambil nilai-nilai pengaturan *slider* dari variabel options serta nama komponen menggunakan `getQueryParams()` lalu mengisi `innerHTML` *container slider* dengan widget slider jQuery. Fungsi `createSlider()` juga men-set nilai *slider* dan fungsi yang dipanggil saat keadaan *slider* berubah.

## 5. Pemrosesan Vertex dan Fragment

Berikut script GLSL pada file `index.html` yang mendefinisikan vertex shader dan fragment shader. Posisi adalah vektor attribute (yakni struktur data yang didefinisikan terhadap vertex) berukuran dua (x, y) dan merupakan hasil transformasi posisi absolut objek terhadap rotasi yang ingin dilakukan user. Pada vertex shader terdapat variabel-variabel konstan (disebut uniform) yang nilainya diperoleh dari fungsi `render` pada `rotate.js`. Posisi dihitung dengan mengambil nilai awal pada koordinat pixel lalu menerapkan beberapa transformasi pada koordinat posisi tersebut. Pertama posisi x dan y dihitung rotasinya dengan perkalian matriks rotasi  $\{\{\cos(t), -\sin(t)\}, \{\sin(t), \cos(t)\}\}$ , lalu ditranslasikan sebanyak nilai translasi tetap (350, 350), dan terakhir dikonversi menjadi koordinat *clip space*. Nilai akhir pada *clip space* inilah yang disimpan pada variabel `gl_Position` untuk nanti dikirim ke GPU. Sementara itu, warna didefinisikan sebagai vektor berukuran empat (yakni struktur data yang didefinisikan per-pixel) yang nilainya adalah tiga bilangan riil random antara 0 sampai 1 untuk nilai merah, hijau, dan biru, serta alpha (*transparency*) 1.

```

<script id="vertex-shader-2d" type="x-shader/x-vertex">
    attribute vec2 a_position;

```

```

uniform vec2 u_resolution;
uniform vec2 u_translation;
uniform vec2 u_rotation;

void main() {
    vec2 rotatedPosition = vec2(
        a_position.x * u_rotation.y + a_position.y * u_rotation.x,
        a_position.y * u_rotation.y - a_position.x * u_rotation.x);

    vec2 position = rotatedPosition + u_translation;

    vec2 zeroToOne = position / u_resolution;

    vec2 zeroToTwo = zeroToOne * 2.0;

    vec2 clipSpace = zeroToTwo - 1.0;

    gl_Position = vec4(clipSpace * vec2(1, -1), 0, 1);
}
</script>

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

    uniform vec4 u_color;

    void main() {
        gl_FragColor = u_color;
    }
</script>

```

Berikut definisi dari fungsi `setGeometry()` yang digunakan pada bagian setup:

```

function setGeometry(gl) {
    gl.bufferData(
        gl.ARRAY_BUFFER,
        new Float32Array([
            // left column
            0, 0,
            30, 0,
            0, 150,
            0, 150,
            30, 0,
            30, 150,

            // top rung
            30, 0,
            100, 0,
            30, 30,
            30, 30,
            100, 0,
            100, 30,

            // middle rung
            30, 60,
            67, 60,
            30, 90,
            30, 90,

```

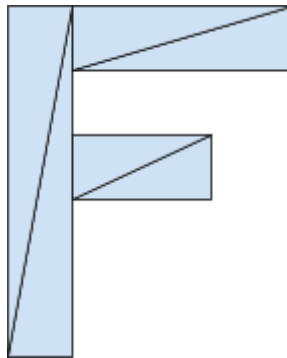
```

        67, 60,
        67, 90,
    ]),
    gl.STATIC_DRAW);
}

```

Koordinat-koordinat berlabel left column mendefinisikan dua segitiga yang digunakan sebagai batang dari objek "F". Koordinat-koordinat berlabel top rung merupakan bagian horizontal atas, dan yang berlabel middle rung merupakan bagian horizontal tengah dari objek "F". Segitiga-segitiga tersebut ditulis ke buffer lalu diinisialisasi dalam objek WebGLRenderingContext.

Berikut contoh representasi objek:



Bagian kode javascript berikut mendefinisikan *slider* rotasi serta nilai awalnya. Translasi (350, 350) juga ditetapkan sebagai nilai *fixed*.

```

var translation = [350, 350];
var rotation = [0, 1];

setupSlider("#angle", {slide: updateAngle, max: 360});

function updateAngle(event, ui) {
    var angleInDegrees = 360 - ui.value;
    var angleInRadians = angleInDegrees * Math.PI / 180;
    rotation[0] = Math.sin(angleInRadians);
    rotation[1] = Math.cos(angleInRadians);
    drawScene();
}

```

## 6. Rendering

Fungsi drawScene() yang digunakan pada potongan kode pendefinisian *slider* sebelumnya menggambar objek "F" menggunakan program yang sudah dibuat di awal.

```

function drawScene() {
    resizeCanvasToDisplaySize(gl.canvas);

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

    gl.clear(gl.COLOR_BUFFER_BIT);

    gl.useProgram(program);
}

```



```
gl.enableVertexAttribArray(positionLocation);

gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);

...
```

Potongan kode di atas menginisialisasi canvas dan buffer. Pertama, canvas di-*resize* menggunakan prosedur UI `resizeCanvasToDisplaySize()`. Kemudian, viewport atau area objek yang terlihat pada layar didefinisikan mulai di pixel (0, 0) hingga seukuran canvas yang didefinisikan di `indexstyles.css`.

```
...
var size = 2;
var type = gl.FLOAT;
var normalize = false;
var stride = 0; each iteration to get the next position
var offset = 0;
gl.vertexAttribPointer(positionLocation, size, type, normalize,
stride, offset);

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

gl.uniform4fv(colorLocation, color);

gl.uniform2fv(translationLocation, translation);

gl.uniform2fv(rotationLocation, rotation);

var primitiveType = gl.TRIANGLES;
var offset = 0;
var count = 18;
gl.drawArrays(primitiveType, offset, count);
}
```

Potongan kode fungsi `drawScene()` di atas pertama mengatur cara membaca data dari buffer (size 2 karena masing data posisi memiliki nilai x dan y, tipe bilangan riil, data tidak perlu dinormalisasi, data posisi disimpan secara kontigu, dan pembacaan dimulai dari data pertama). Kemudian data tersebut dibaca ke uniform `positionLocation`. Setelah itu, ukuran canvas disimpan di uniform `resolutionLocation`, serta nilai translasi dan rotasi disimpan ke uniform-uniform yang bersangkutan. Setelah itu, method `drawArrays()` digunakan untuk menggambar segitiga-segitiga yang sudah didefinisikan menggunakan vertex shader dan fragment shader yang sudah diatur.

## 7. Hasil

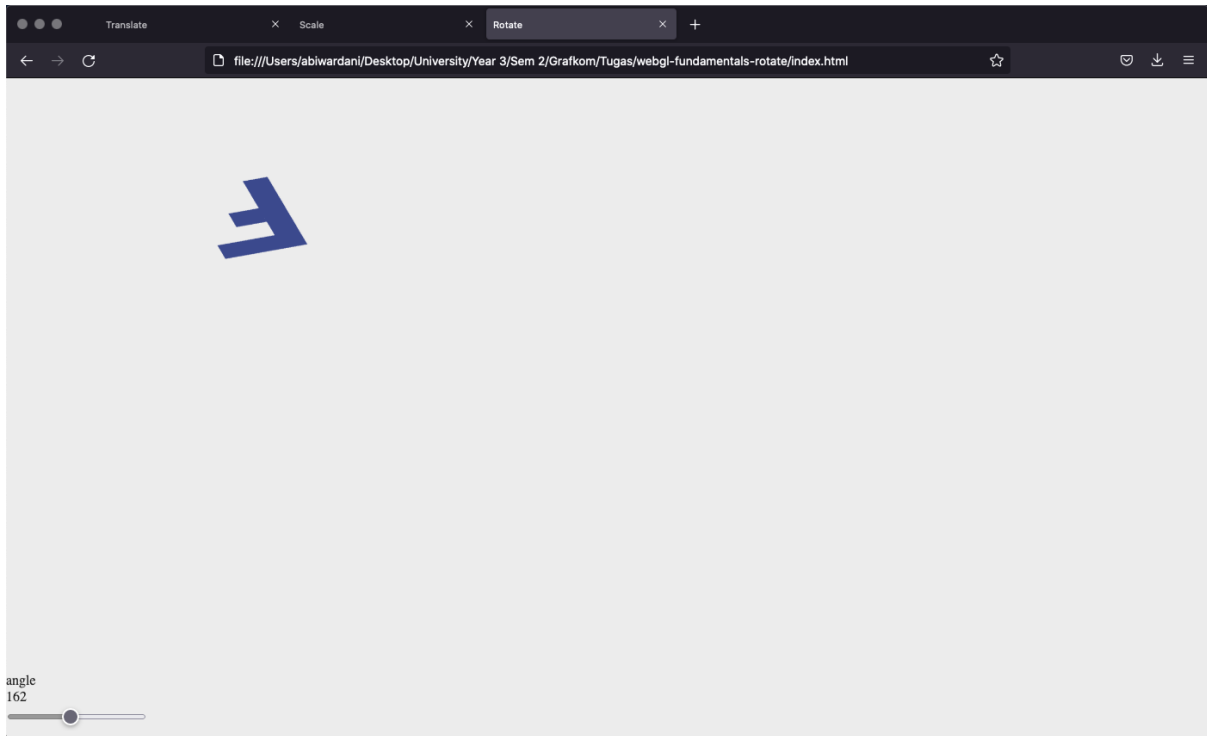
### a. Parameter default

Saat pertama membuka program, objek “F” ditampilkan pada layar.

Warna objek adalah random, dan *slider* berada pada nilai defaultnya, yakni rotasi 0 derajat.

b. Rotasi

*Slider* angle digeser ke kanan untuk merotasi objek searah jarum jam.



**8. Lampiran**

a. Source Code

Source code dapat diakses di repository github pada link <https://github.com/abwardani/webgl-fundamentals-rotate>

b. Video

Video singkat penjelasan “Rotate” dapat ditonton pada link YouTube <https://www.youtube.com/watch?v=8MxxqMiQO1k>