Laporan Tugas WebGL 1

Disusun Oleh :
Aufa Nabil Amiri - 0721 17 4000 0029

Teknik Komputer
Fakultas Teknologi Elektro dan Informatika Cerdas Institut Teknologi Sepuluh Nopember

1 File HTML

```
<body onload="startup();">
  <canvas id="canvas" width="500" height="500" />

  <script type="text/javascript" src="initShader.js"></script>
  <script type="text/javascript" src="index.js"></script>
```

Gambar 1: file html yang digunakan

seperti yang bisa dilihat dalam gambar 1, file html ini akan berfungsi untuk mengatur ukuran dari canvas yang akan digunakan. Canvas sendiri adalah tempat dimana webgl bisa melakukan manipulasi piksel secara bebas

selain itu, juga dilakukan import *script initShader.js* dan *index.js*. *initShader.js* digunakan untuk melakukan inisiasi Vertex Shader dan Fragment Shader. sedangkan *index.js* berisi program utama.

2 initShader.js

2.1 loadShader

```
function loadShader(gl, type, shaderSource) {
  const shader = gl.createShader(type);

  gl.shaderSource(shader, shaderSource);
  gl.compileShader(shader);
      You, 3 days ago • initial test
  // if something happened, delete the shader and return null
  if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
    console.error(`ERROR ${gl.getShaderInfoLog(shader)}`);
    gl.deleteShader(shader);
    return null;
  }

  return shader;
}
```

Gambar 2: fungsi *loadShader* dalam *initShader.js*

fungsi dalam gambar 2 digunakan untuk melakukan kompilasi terhadap shader yang digunakan. Hasil dari kompilasi akan digunakan untuk mengatur

bagaimana sebuah vertex akan ditampilkan di dalam canvas. Apabila dalam proses kompilasi terjadi error, maka shader yang sudah dikompilasi tadi akan dihapus dan fungsi akan melakukan return null.

2.2 setupShader

```
function setupShaders(gl, vertexSource, fragmentSource) {
  var vertextShader = loadShader(gl, gl.VERTEX_SHADER, vertexSource);
  var fragmentShader = loadShader(gl, gl.FRAGMENT_SHADER, fragmentSource);

  shaderProgram = gl.createProgram();
  gl.attachShader(shaderProgram, vertextShader);
  gl.attachShader(shaderProgram, fragmentShader);
  gl.linkProgram(shaderProgram);

  if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
    console.error("failed to setup shaders");
  }

  return shaderProgram;
}
```

Gambar 3: fungsi setupShader dalam initShader.js

setiap program webgl akan memerlukan vertex shader dan fragment shader. program di gambar 3 digunakan untuk menggabungkan antara vertex shader dan fragment shader dan dengan menggunakan *linkProgram* menghubungkan antara program dengan shader yang sudah digabung tadi.

3 index.js

3.1 setup vertexShader

yang pertama kali harus di set dalam program webgl adalah vertex shader dan fragment shader. dalam vertex Shader, harus terdapat fungsi main, yaitu fungsi yang pertama kali akan dipanggil oleh webgl. Selain itu, juga harus terdapat assign value ke gl_Position yang akan digunakan oleh webgl untuk mengetahui posisi piksel yang akan digambar.

hal lain yang harus dilihat adalah *attribute* dan *varying* yang terdapat dalam program vertex. *attribute* digunakan untuk menghubungkan antara program (dalam hal ini index.js) dengan shader. sedangkan *varying* digunakan untuk menghubungkan antara vertex Shader dengan fragment Shader.

```
var vertextShaderSource = `
attribute vec3 aVertexPosition;
attribute vec3 color;
varying vec3 vColor;

void main() {
    gl_Position = vec4(aVertexPosition, 1.0);
    vColor = color;
}
`;
```

Gambar 4: vertexShader

Terdapat satu lagi jenis variabel yang tidak dipakai dalam shader ini adalah *uniform*.

untuk saat ini, dalam vertex Shader kita hanya melakukan assign gl_Position dan vColor.

3.2 setup fragmentShader

```
var fragmentShaderSource = `
precision mediump float;
    You, 4 days ago • initial commit
varying vec3 vColor;

void main() {
    gl_FragColor = vec4(vColor, 1.0);
}
`;
```

Gambar 5: fragmentShader

fragment shader berguna untuk melakukan set bagaimana warna dari suatu vertex. dalam gambar 5, kita hanya mengambil value yang di set di vertex Shader dan memasukkannya ke dalam gl_FragColor. gl_FragColor digunakan oleh webgl dan harus ada dalam fragment Shader.

```
function createGLContext(canvas) {
  var names = ["webgl", "experimental-webgl"];
  var context = null;

  // checking if the browser support WebGL or not
  for (var i = 0; i < names.length; i++) {
    try {
      context = canvas.getContext(names[i]);
    } catch (e) {
      console.error(e);
    }

    if (context) {      You, 4 days ago • initial com
      break;
    }
  }

  if (context) {
    context.viewportWidth = canvas.width;
    context.viewportHeight = canvas.height;
  } else {
    console.error("failed to craete WEBGL context");
  }

  return context;
}
```

Gambar 6: fungsi createContext

3.3 create WebGL Context

sebelum melakukan proses gambar dengan menggunakan webgl, kita memerlukan context webGL terlebih dahulu. secara general, kita hanya tinggal memanggil fungsi *getContext*, namun terdapat beberapa browser yang tidak memiliki support webGL secara default.

fungsi di gambar 6 secara otomatis memilih backend webgl, apakah itu webgl atau experimental-webgl. context ini nantinya akan digunakan dalam seluruh program.

```
function draw(gl, bufferData, itemSize, attrLocation) {
  const tempBuffer = gl.createBuffer();

  gl.bindBuffer(gl.ARRAY_BUFFER, tempBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, bufferData, gl.STATIC_DRAW);

  gl.vertexAttribPointer(attrLocation, itemSize, gl.FLOAT, false, 0, 0);
  gl.enableVertexAttribArray(attrLocation);
}
```

Gambar 7: fungsi draw

3.4 fungsi Draw

fungsi draw ini digunakan untuk melakukan assign terhadap attribute - attribute yang sudah ditulis baik di vertex Shader maupun fragment Shader. dalam WebGl, buffer hanya dapat menyimpan 1 variabel saja, sehingga apabila terdapat 2 attribute atau lebih kita harus melakukan bindBuffer lagi.

dalam gambar 7, dilakukan inisialisasi buffer kosong dan melakukan bind terhadapt buffer kosong tersebut ke dalam internal variabel webGL. Untuk mengisi variabel yang sudah di bind tersebut, kita menggunakan fungsi *bufferData*. di dalam fungsi tersebut, selain melakukan assign data ke buffer, kita juga harus mengatur bagaimana penggunaan variabel tersebut, untuk saat ini menggunakan konstanta gl.STATIC_DRAW yang berarti variabel tersebut di set sekali dan digunakan berkali - kali.

selanjutnya, dilakukan assign variabel tersebut ke dalam vertex Shader dengan menggunakan fungsi *vertexAttribPointer* dan dilakukan enable terhadap fungsi tersebut menggunakan *enableVertexAttribArray*.

3.5 fungsi Startup

fungsi ini adalah fungsi yang pertama kali dipanggil saat page sudah diload. Seperti bisa dilihat pada gambar 8, pada bagian pertama kita membuat context dengan fungsi yang sudah ditulis diatas. Selanjutnya mengatur shader yang digunakan dan menyimpan value attribute shader kedalam variabel. *useProgram* digunakan untuk memilih shaderProgram yang mana yang akan digunakan oleh program saat ini.

Untuk menggambar segitiga, kita harus mengetahui koordinat dari webGL, yaitu pada pojok kiri atas memiliki koordinat (-1,1), pojok kanan atas (1,1) dan seterusnya dengan koordinat (0,0) berada di tengah - tengah. Sedangkan untuk melakukan assign warna, perlu diketahui bahwa warna di WebGl menggunakan float antara 0 - 1.

```
function startup() {
  /** @type {HTMLCanvasElement} */
  const canvas = document.getElementById("canvas");

  /** @type {WebGLRenderingContext} */
  const gl = createGLContext(canvas);

  const shaderProgram = setupShaders(
    gl,
    vertextShaderSource,
    fragmentShaderSource
  );

  const programInfo = {
    attr: {
      vertexPostition: gl.getAttribLocation(shaderProgram, "aVertexPosition"),
      color: gl.getAttribLocation(shaderProgram, "color"),
    },
  };

  var triangleVertices = [0.0, 0.5, 0.0, -0.5, -0.5, 0.0, 0.5, -0.5, 0.0];
  var colors = [0, 0, 1, 1, 0, 0, 0, 1, 0];

  var newTirangle = [0.5, 0.2, 0.0, 0.1, 0.5, 0.0, 0.9, 0.5, 0.0];
  var newColors = [1, 1, 0, 1, 0, 1, 0, 1, 1];

  gl.useProgram(shaderProgram);

  gl.clearColor(102 / 255, 153 / 255, 1.0, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT);
  gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);

  // draw vertice
  draw(
    gl,
    new Float32Array(triangleVertices),
    3,
    programInfo.attr.vertexPostition
  );
  // draw colors
  draw(gl, new Float32Array(colors), 3, programInfo.attr.color);
  gl.drawArrays(gl.TRIANGLES, 0, triangleVertices.length / 3);

  draw(gl, new Float32Array(newTirangle), 3, programInfo.attr.vertexPostition);
  draw(gl, new Float32Array(newColors), 3, programInfo.attr.color);
  gl.drawArrays(gl.TRIANGLES, 0, newTirangle.length / 3);
}
```

Gambar 8: fungsi startup

Hal lain yang harus kita lakukan sebelum mulai menggambar segitiga adalah mengatur warna background, diatur menggunakan fungsi *clearColor*.

Selain itu, juga perlu mengatur berapa ukuran viewport atau dalam hal ini canvas yang dipakai oleh webGL.

Setelah memanggil fungsi draw yang sudah dibuat diatas, kita hanya tinggal memanggil *drawArrays*. *drawArrays* akan menggambar berdasarkan value - value yang sudah di set pada saat memanggil fungsi draw. Perlu dilihat bahwa untuk saat ini kita menggunakan konstanta *gl.TRIANGLES* karena hanya perlu menggambar segitiga saja, untuk menggambar objek lain mungkin akan menggunakan konstanta yag lain.

Apabila ingin menggambar beberapa objek sekaligus dengan shader yang sama, kita hanya perlu memanggil fungsi draw dan drawArrays lagi maka objek - objek tersebut akan ditampilkan di dalam canvas.

Catatan

seluruh kode dalam laporan ini dapat diakses pada url `https://github.com/chillytaka/webGL`
demo untuk project ini dapat diakses pada url `https://chillytaka.github.io/webgl-1`