Laporan Implementasi Phong Shading
Grafika Komputer

Semester Ganjil 2020

Hafizh Fauzan 0721 17 4000 0019

Departemen Teknik Komputer

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

Langkah

Membuat Canvas pada HTML

```
<body>
  <div>
    <canvas id="gl-canvas" width="512" height="512">
    </canvas>
  </div>
 </body>
```

Diperlukan canvas pada project ini, sehingga inisiasi terlebih dahulu dengan cara memanggil *canvas* dan set id nya

Membuat Vertex Shader

```
<script id="vertex-shader" type="x-shader/x-vertex">
  attribute vec4 vNormal;
  attribute vec4 vPosition;
  varying vec3 L, N, E;

  uniform mat4 modelViewMatrix;
  uniform mat4 projectionMatrix;
  uniform vec3 theta;
  uniform vec4 lightPosition;

  void main()
  {
   vec3 angles = radians(theta);
   vec3 c = cos(angles);
   vec3 s = sin(angles);
   mat4 rx = mat4(
     1.0, 0.0, 0.0, 0.0,
     0.0, c.x, s.x, 0.0,
     0.0, -s.x, c.x, 0.0,
     0.0, 0.0, 0.0, 1.0
   );
   mat4 ry = mat4(
     c.y, 0.0, -s.y, 0.0,
     0.0, 1.0, 0.0, 0.0,
     s.y, 0.0, c.y, 0.0,
     0.0, 0.0, 0.0, 1.0
   );

   vec3 pos = (modelViewMatrix * rx * ry * vPosition).xyz;
   vec3 lightPos = (modelViewMatrix * lightPosition).xyz;

   L = normalize(lightPos - pos);
   N = normalize((modelViewMatrix * rx * ry * vNormal).xyz);
   E = -normalize(pos);

   gl_Position = projectionMatrix * modelViewMatrix * rx * ry * vPosition;
  }
 </script>
```

Pada index.html juga kita inisiasi vertex shader, dimiana juga terdapat variable untuk posisi koordinat objek dan posisi cahaya

Membuat Fragment Shader

```
<script id="fragment-shader" type="x-shader/x-fragment">
  precision mediump float;
  varying vec3 L, N, E;
```

```
  uniform vec4 ambientProduct;
  uniform vec4 diffuseProduct;
  uniform vec4 specularProduct;
  uniform float shininess;

  void main()
  {
    vec4 diffuse = max(dot(L, N), 0.0) * diffuseProduct;
    vec3 H = normalize(L+E);
    vec4 specular = pow(max(dot(N, H), 0.0), shininess) * specularProduct;

    if (dot(L, N) < 0.0)
      specular = vec4(0.0, 0.0, 0.0, 1.0);

    vec4 fColor = ambientProduct + diffuse + specular;
    fColor.a = 1.0;

    gl_FragColor = fColor;
  }
</script>
```

Fragment Shader juga diinisialisasi pada index.html untuk mengatur warna berdasarkan cahaya dan sifat material

Membuat Kotak

```
function createCube(size)
{
 this.colorFaces = vec4(0.0, 0.0, 0.0, 1.0);
 this.colorEdges = vec4(0.0, 0.0, 0.0, 1.0);

 this.count_vertices_faces = 6 * 6;
 this.count_vertices_edges = 6 * 8;

 var pos =  size / 2.0;

 this.vertices = [
    vec4( -pos, -pos, pos, 1.0 ),
    vec4( -pos, pos, pos, 1.0 ),
    vec4( pos, pos, pos, 1.0 ),
    vec4( pos, -pos, pos, 1.0 ),
    vec4( -pos, -pos, -pos, 1.0 ),
    vec4( -pos, pos, -pos, 1.0 ),
    vec4( pos, pos, -pos, 1.0 ),
    vec4( pos, -pos, -pos, 1.0 )
 ];


 this.faces_as_triangles = function (normals = false)
 {
  var arr = [];

  this._quad_triangles( arr, normals, 1, 2, 6, 5 );
  this._quad_triangles( arr, normals, 5, 4, 0, 1 );
  this._quad_triangles( arr, normals, 1, 0, 3, 2 );
  this._quad_triangles( arr, normals, 2, 3, 7, 6 );
  this._quad_triangles( arr, normals, 7, 3, 0, 4 );
  this._quad_triangles( arr, normals, 7, 4, 5, 6 );

  return arr;
 }

 this._quad_triangles = function(arr, normals, a, b, c, d)
 {
    var t1 = vec4(subtract(this.vertices[b], this.vertices[a]));
    var t2 = vec4(subtract(this.vertices[c], this.vertices[b]));
    var normal = vec4(normalize(cross(t1, t2)));
    normal[3] = 0.0;

    var indices = [ a, b, c, a, c, d ];
    for ( var i = 0; i < indices.length; ++i ) {
       arr.push( this.vertices[indices[i]] );
```

```
        if (normals)
        {
           normals.push(normal);
        }
     }
  }

  this.edges_as_line_segments = function ()
  {
   var arr = [];

   this._square_line_segments( arr, 1, 2, 6, 5 );
   this._square_line_segments( arr, 5, 4, 0, 1 );
   this._square_line_segments( arr, 1, 0, 3, 2 );
   this._square_line_segments( arr, 2, 3, 7, 6 );
   this._square_line_segments( arr, 7, 3, 0, 4 );
   this._square_line_segments( arr, 7, 4, 5, 6 );

   return arr;
  }

  this._square_line_segments = function(arr, a, b, c, d)
  {
   var indices = [ a, b, b, c, c, d, d, a ];
   for ( var i = 0; i < indices.length; ++i ) {
      arr.push( this.vertices[indices[i]] );
   }
  }

  return this;
}
```

Pada geometry.js dibuat fungsi untuk membuat kubus berdasarkan besar yg diinginkan melalui perhitungan vertex, indices.

Inisiasi Shader

```
function initShaders( gl, vertexShaderId, fragmentShaderId )
{
    var vertShdr;
    var fragShdr;

    var vertElem = document.getElementById( vertexShaderId );
    if ( !vertElem ) {
        alert( "Unable to load vertex shader " + vertexShaderId );
        return -1;
    }
    else {
        vertShdr = gl.createShader( gl.VERTEX_SHADER );
        gl.shaderSource( vertShdr, vertElem.text );
        gl.compileShader( vertShdr );
        if ( !gl.getShaderParameter(vertShdr, gl.COMPILE_STATUS) ) {
            var msg = "Vertex shader failed to compile.  The error log is:"
                    + "<pre>" + gl.getShaderInfoLog( vertShdr ) + "</pre>";
            alert( msg );
            return -1;
        }
    }

    var fragElem = document.getElementById( fragmentShaderId );
    if ( !fragElem ) {
        alert( "Unable to load vertex shader " + fragmentShaderId );
        return -1;
    }
    else {
        fragShdr = gl.createShader( gl.FRAGMENT_SHADER );
        gl.shaderSource( fragShdr, fragElem.text );
        gl.compileShader( fragShdr );
        if ( !gl.getShaderParameter(fragShdr, gl.COMPILE_STATUS) ) {
            var msg = "Fragment shader failed to compile.  The error log is:"
                    + "<pre>" + gl.getShaderInfoLog( fragShdr ) + "</pre>";
```

```
            alert( msg );
            return -1;
        }
    }

    var program = gl.createProgram();
    gl.attachShader( program, vertShdr );
    gl.attachShader( program, fragShdr );
    gl.linkProgram( program );

    if ( !gl.getProgramParameter(program, gl.LINK_STATUS) ) {
        var msg = "Shader program failed to link.  The error log is:"
            + "<pre>" + gl.getProgramInfoLog( program ) + "</pre>";
        alert( msg );
        return -1;
    }

    return program;
}
```

initShaders.js digunakan untuk melakukan inisisasi shader yang akan digunakan

Perhitungan Perspective

```
function radians(degrees) {
  return degrees * Math.PI / 180.0;
}

function _argumentsToArray(args) {
  return [].concat.apply([], Array.prototype.slice.apply(args));
}

function flatten(v) {
  if (v.matrix === true) {
    v = transpose(v);
  }

  var n = v.length;
  var elemsAreArrays = false;

  if (Array.isArray(v[0])) {
    elemsAreArrays = true;
    n *= v[0].length;
  }

  var floats = new Float32Array(n);

  if (elemsAreArrays) {
    var idx = 0;
    for (var i = 0; i < v.length; ++i) {
      for (var j = 0; j < v[i].length; ++j) {
        floats[idx++] = v[i][j];
      }
    }
  } else {
    for (var i = 0; i < v.length; ++i) {
      floats[i] = v[i];
    }
  }

  return floats;
}

//----------------------------------------------------------------------------
//
//  Vector constructors
//

function vec3() {
  var result = _argumentsToArray(arguments);

  switch (result.length) {
    case 0:
      result.push(0.0);
```

```
    case 1:
      result.push(0.0);
    case 2:
      result.push(0.0);
  }

  return result.splice(0, 3);
}

function vec4() {
  var result = _argumentsToArray(arguments);

  switch (result.length) {
    case 0:
      result.push(0.0);
    case 1:
      result.push(0.0);
    case 2:
      result.push(0.0);
    case 3:
      result.push(1.0);
  }

  return result.splice(0, 4);
}

//----------------------------------------------------------------------------
//
//  Vector Functions
//

function length(u) {
  return Math.sqrt(dot(u, u));
}

function dot(u, v) {
  if (u.length != v.length) {
    throw "dot(): vectors are not the same dimension";
  }

  var sum = 0.0;
  for (var i = 0; i < u.length; ++i) {
    sum += u[i] * v[i];
  }

  return sum;
}

function cross(u, v) {
  if (!Array.isArray(u) || u.length < 3) {
    throw "cross(): first argument is not a vector of at least 3";
  }

  if (!Array.isArray(v) || v.length < 3) {
    throw "cross(): second argument is not a vector of at least 3";
  }

  var result = [
    u[1] * v[2] - u[2] * v[1],
    u[2] * v[0] - u[0] * v[2],
    u[0] * v[1] - u[1] * v[0]
  ];

  return result;
}

function negate(u) {
  var result = [];
  for (var i = 0; i < u.length; ++i) {
    result.push(-u[i]);
  }

  return result;
}
```

```
function normalize(u, excludeLastComponent) {
  if (excludeLastComponent) {
    var last = u.pop();
  }

  var len = length(u);

  if (!isFinite(len)) {
    throw "normalize: vector " + u + " has zero length";
  }

  for (var i = 0; i < u.length; ++i) {
    u[i] /= len;
  }

  if (excludeLastComponent) {
    u.push(last);
  }

  return u;
}

//----------------------------------------------------------------------------
//
//  Matrix constructors
//

function mat4() {
  var v = _argumentsToArray(arguments);

  var m = [];
  switch (v.length) {
    case 0:
      v[0] = 1;
    case 1:
      m = [
        vec4(v[0], 0.0, 0.0, 0.0),
        vec4(0.0, v[0], 0.0, 0.0),
        vec4(0.0, 0.0, v[0], 0.0),
        vec4(0.0, 0.0, 0.0, v[0])
      ];
      break;

    default:
      m.push(vec4(v));
      v.splice(0, 4);
      m.push(vec4(v));
      v.splice(0, 4);
      m.push(vec4(v));
      v.splice(0, 4);
      m.push(vec4(v));
      break;
  }

  m.matrix = true;

  return m;
}

//----------------------------------------------------------------------------
//
//  Rotation matrix generators
//

function rotateX(theta) {
  var c = Math.cos(radians(theta));
  var s = Math.sin(radians(theta));
  var rx = mat4(
    1.0, 0.0, 0.0, 0.0,
    0.0, c, s, 0.0,
    0.0, -s, c, 0.0,
    0.0, 0.0, 0.0, 1.0
  );
  return rx;
}
```

```
function rotateY(theta) {
  var c = Math.cos(radians(theta));
  var s = Math.sin(radians(theta));
  var ry = mat4(
    c, 0.0, -s, 0.0,
    0.0, 1.0, 0.0, 0.0,
    s, 0.0, c, 0.0,
    0.0, 0.0, 0.0, 1.0
  );
  return ry;
}

//----------------------------------------------------------------------------
//
//  View matrix generators
//

function lookAt(eye, at, up) {
  if (!Array.isArray(eye) || eye.length != 3) {
    throw "lookAt(): first parameter [eye] must be an a vec3";
  }

  if (!Array.isArray(at) || at.length != 3) {
    throw "lookAt(): first parameter [at] must be an a vec3";
  }

  if (!Array.isArray(up) || up.length != 3) {
    throw "lookAt(): first parameter [up] must be an a vec3";
  }

  if (equal(eye, at)) {
    return mat4();
  }

  var v = normalize(subtract(at, eye)); // view direction vector
  var n = normalize(cross(v, up)); // perpendicular vector
  var u = normalize(cross(n, v)); // "new" up vector

  v = negate(v);

  var result = mat4(
    vec4(n, -dot(n, eye)),
    vec4(u, -dot(u, eye)),
    vec4(v, -dot(v, eye)),
    vec4()
  );

  return result;
}

//----------------------------------------------------------------------------
//
//  Projection Matrix Generators
//

function perspective(fovy, aspect, near, far) {
  var f = 1.0 / Math.tan(radians(fovy) / 2);
  var d = far - near;

  var result = mat4();
  result[0][0] = f / aspect;
  result[1][1] = f;
  result[2][2] = -(near + far) / d;
  result[2][3] = -2 * near * far / d;
  result[3][2] = -1;
  result[3][3] = 0.0;

  return result;
}



//----------------------------------------------------------------------------
//
```

```
//  Matrix functions
//

function transpose(m) {
  if (!m.matrix) {
    return "transpose(): trying to transpose a non-matrix";
  }

  var result = [];
  for (var i = 0; i < m.length; ++i) {
    result.push([]);
    for (var j = 0; j < m[i].length; ++j) {
      result[i].push(m[j][i]);
    }
  }

  result.matrix = true;

  return result;
}

//---------------------------------------------------------------------------
//
//  Vector and matrix functions
//


function mult(u, v) {
  var result = [];

  if (u.matrix && v.matrix) {
    if (u.length != v.length) {
      throw "mult(): trying to add matrices of different dimensions";
    }

    for (var i = 0; i < u.length; ++i) {
      if (u[i].length != v[i].length) {
        throw "mult(): trying to add matrices of different dimensions";
      }
    }

    for (var i = 0; i < u.length; ++i) {
      result.push([]);

      for (var j = 0; j < v.length; ++j) {
        var sum = 0.0;
        for (var k = 0; k < u.length; ++k) {
          sum += u[i][k] * v[k][j];
        }
        result[i].push(sum);
      }
    }

    result.matrix = true;

    return result;
  }

  if (u.matrix && (u.length == v.length)) {
    for (var i = 0; i < v.length; i++) {
      var sum = 0.0;
      for (var j = 0; j < v.length; j++) {
        sum += u[i][j] * v[j];
      }
      result.push(sum);
    }
    return result;
  } else {
    if (u.length != v.length) {
      throw "mult(): vectors are not the same dimension";
    }

    for (var i = 0; i < u.length; ++i) {
      result.push(u[i] * v[i]);
    }
```

```
    return result;
  }
}


function subtract(u, v) {
  var result = [];

  if (u.matrix && v.matrix) {
    if (u.length != v.length) {
      throw "subtract(): trying to subtract matrices" +
        " of different dimensions";
    }

    for (var i = 0; i < u.length; ++i) {
      if (u[i].length != v[i].length) {
        throw "subtract(): trying to subtact matrices" +
          " of different dimensions";
      }
      result.push([]);
      for (var j = 0; j < u[i].length; ++j) {
        result[i].push(u[i][j] - v[i][j]);
      }
    }

    result.matrix = true;

    return result;
  } else if (u.matrix && !v.matrix || !u.matrix && v.matrix) {
    throw "subtact(): trying to subtact  matrix and non-matrix variables";
  } else {
    if (u.length != v.length) {
      throw "subtract(): vectors are not the same length";
    }

    for (var i = 0; i < u.length; ++i) {
      result.push(u[i] - v[i]);
    }

    return result;
  }
}

function equal(u, v) {
  if (u.length != v.length) {
    return false;
  }

  if (u.matrix && v.matrix) {
    for (var i = 0; i < u.length; ++i) {
      if (u[i].length != v[i].length) {
        return false;
      }
      for (var j = 0; j < u[i].length; ++j) {
        if (u[i][j] !== v[i][j]) {
          return false;
        }
      }
    }
  } else if (u.matrix && !v.matrix || !u.matrix && v.matrix) {
    return false;
  } else {
    for (var i = 0; i < u.length; ++i) {
      if (u[i] !== v[i]) {
        return false;
      }
    }
  }

  return true;
}
```

Fungsi dari vecMat.js adalah sebagai kumpulan fungsi matematis dan juga melakukan perhitungan untuk perspective view dan lookAt yang akan digunakan.

Animate & Render

```
var gl;
var cube, theta, thetaLoc, colorLoc;

const eye = vec3(0, 0, 2);
const at = vec3(0.0, 0.0, 0.0);
const up = vec3(0.0, 1.0, 0.0);

const fov = 55;
const near = 0.3;
const far = 5;

var lightPosition = vec4(-1.5, 2.0, 4.0, 1.0);
var lightAmbient = vec4(0.2, 0.2, 0.2, 1.0);
var lightDiffuse = vec4(1.0, 1.0, 1.0, 1.0);
var lightSpecular = vec4(1.0, 1.0, 1.0, 1.0);

var materialAmbient = vec4(0.0, 1.0, 0.0, 1.0);
var materialDiffuse = vec4(1.0, 0.8, 1.0, 1.0);
var materialSpecular = vec4(0.0, 0.4, 0.4, 1.0);
var materialShininess = 300.0;

window.onload = function init()
{
   var canvas = document.getElementById("gl-canvas");

   gl = WebGLUtils.setupWebGL(canvas);
   if (!gl) {
      alert("WebGL isn't available");
   }

   cube = createCube(1.0);
   theta = [0.0, 0.0, 0.0];

   gl.viewport(0, 0, canvas.width, canvas.height);
   gl.clearColor(1.0, 1.0, 1.0, 1.0);

   gl.enable(gl.DEPTH_TEST);

   var program = initShaders(gl, "vertex-shader", "fragment-shader");
   gl.useProgram(program);

   var modelViewMatrix = lookAt(eye, at, up);
   modelViewMatrixLoc = gl.getUniformLocation(program, "modelViewMatrix");
   gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(modelViewMatrix));

   var aspectRatio = gl.canvas.width / gl.canvas.height;
   var projectionMatrix = perspective(fov, aspectRatio, near, far);
   projectionMatrixLoc = gl.getUniformLocation( program, "projectionMatrix" );
   gl.uniformMatrix4fv(projectionMatrixLoc, false, flatten(projectionMatrix));

   thetaLoc = gl.getUniformLocation(program, "theta");

   var points = [];
   var normals = [];
   points = points.concat(cube.faces_as_triangles(normals));

   var lightPositionLoc = gl.getUniformLocation(program, "lightPosition");
   gl.uniform4fv(lightPositionLoc, flatten(lightPosition));

   var ambientProduct = mult(lightAmbient, materialAmbient);
   var ambientProductLoc = gl.getUniformLocation(program, "ambientProduct");
   gl.uniform4fv(ambientProductLoc, flatten(ambientProduct));

   var diffuseProduct = mult(lightDiffuse, materialDiffuse);
   var diffuseProductLoc = gl.getUniformLocation(program, "diffuseProduct");
   gl.uniform4fv(diffuseProductLoc, flatten(diffuseProduct));
```

```
    var specularProduct = mult(lightSpecular, materialSpecular);
    var specularProductLoc =
      gl.getUniformLocation(program, "specularProduct");
    gl.uniform4fv(specularProductLoc, flatten(specularProduct));

    var shininessLoc = gl.getUniformLocation(program, "shininess");
    gl.uniform1f(shininessLoc, materialShininess);

    var vBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW);

    var vPosition = gl.getAttribLocation(program, "vPosition");
    gl.vertexAttribPointer(vPosition, 4, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(vPosition);

    var nBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, nBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, flatten(normals), gl.STATIC_DRAW);

    var vNormal = gl.getAttribLocation(program, "vNormal");
    gl.vertexAttribPointer(vNormal, 4, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(vNormal);

    render();
}

function render()
{
    gl.clear( gl.COLOR_BUFFER_BIT );

    theta[0] += 0.5;
    theta[1] += 1.0;

    if (theta[1] > 360.0) {
      theta[1] -= 360.0;
    }
    if (theta[0] > 360.0) {
      theta[0] -= 360.0;
    }

    gl.uniform3fv(thetaLoc, flatten(theta));

    gl.drawArrays( gl.TRIANGLES, 0, cube.count_vertices_faces );

    requestAnimFrame( render );
}
```

index.js berfungsi untuk melakukan render yang akan ditunjukan pada canvas

URL dan Source Code

Source code di upload pada https://github.com/Fizdan/Grafika-Komputer/tree/main/EAS dan Live demo bisa dilihat pada https://fizdan.github.io/grafkom/index.html