

SPRAWOZDANIE
Zajęcia: Grafika komputerowa
Prowadzący: prof. dr. hab. Vasyl Martsenyuk

Laboratorium 11
13 V 2021 r.
Temat: „Grafika 3D w bibliotece WebGL/GLSL”
Wariant:
Liczba kątów: 5

Robert Laszczak
Informatyka I stopień
Stacjonarne, 4 semestr
Grupa 2B

1. Polecenie

Plik lab12.html pokazuje mały sześcián, który można obrócić, przeciągając myszą na płótnie. Zadaniem jest zastąpienie sześciánu dużym wiatrakiem siedzącym na prostokątnej podstawie, jak pokazano na rysunku. Łopatki wiatraka powinny obracać się po włączeniu animacji. Każda łopatka wiatraka powinna być zbudowana z dwóch stożków.

2. Wprowadzam dane:

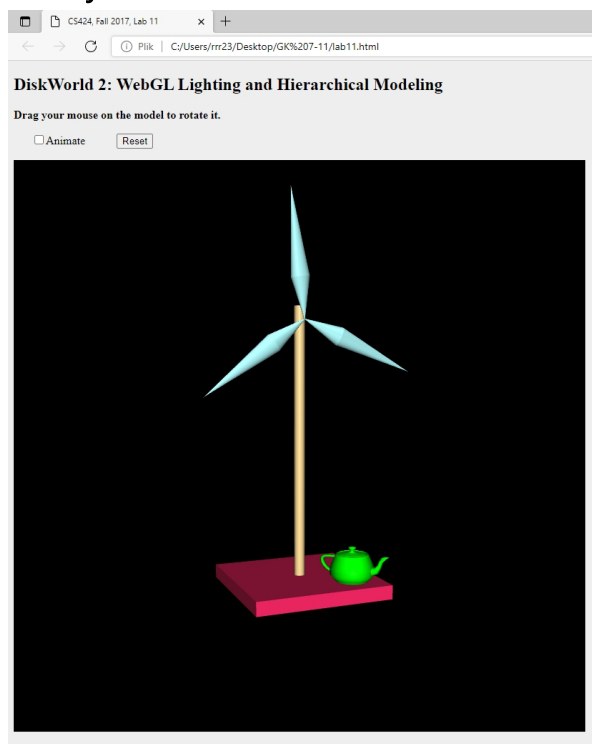
Liczba kątów - 5

3. Wykorzystane komendy:

Link do zdalnego repozytorium:

https://github.com/RLaszczak/Lab_GK

4. Wyniki działania:



5. Wnioski

Na podstawie otrzymanego wyniku można stwierdzić, że:

- Możemy sterować kolorami i wielkością brył oraz tworzyć animację

Kod:

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <title>CS424, Fall 2017, Lab 11</title>
  <style>
    body {
      background-color: #EEEEEE;
    }
    label {
      white-space: pre;
      margin-left: 25px;
    }
  </style>

  <script type="x-shader/x-vertex" id="vshader-source">
attribute vec3 a_coords;
attribute vec3 a_normal;
uniform mat4 modelview;
uniform mat4 projection;
varying vec3 v_normal;
varying vec3 v_eyeCoords;
void main() {
  vec4 coords = vec4(a_coords,1.0);
  vec4 eyeCoords = modelview * coords;
  gl_Position = projection * eyeCoords;
  v_normal = normalize(a_normal);
  v_eyeCoords = eyeCoords.xyz/eyeCoords.w;
}
</script>

  <script type="x-shader/x-fragment" id="fshader-source">
#ifdef GL_FRAGMENT_PRECISION_HIGH
  precision highp float;
#else
  precision mediump float;
#endif
  struct MaterialProperties {
    vec3 diffuseColor;    // diffuseColor.a is alpha for the fragment
    vec3 specularColor;
    vec3 emissiveColor;
    float specularExponent;
  };
  struct LightProperties {
    bool enabled;
    vec4 position;
    vec3 color;
  };
  uniform MaterialProperties material; // do two-sided lighting, but assume front and back materials are the same
```

```

uniform LightProperties lights[4];
uniform mat3 normalMatrix;
varying vec3 v_normal;
varying vec3 v_eyeCoords;
vec3 lightingEquation( LightProperties light, MaterialProperties material,
                      vec3 eyeCoords, vec3 N, vec3 V ) {
    // N is normal vector, V is direction to viewer.
    vec3 L, R; // Light direction and reflected light direction.
    if ( light.position.w == 0.0 ) {
        L = normalize( light.position.xyz );
    }
    else {
        L = normalize( light.position.xyz/light.position.w - v_eyeCoords );
    }
    if (dot(L,N) <= 0.0) {
        return vec3(0.0);
    }
    vec3 reflection = dot(L,N) * light.color * material.diffuseColor;
    R = -reflect(L,N);
    if (dot(R,V) > 0.0) {
        float factor = pow(dot(R,V),material.specularExponent);
        reflection += factor * material.specularColor * light.color;
    }
    return reflection;
}

void main() {
    vec3 normal = normalize( normalMatrix*v_normal );
    vec3 viewDirection = normalize( -v_eyeCoords); // (Assumes a perspective projection.)
    vec3 color = material.emissiveColor;
    for (int i = 0; i < 4; i++) {
        if (lights[i].enabled) {
            if (gl_FrontFacing) {
                color += lightingEquation( lights[i], material, v_eyeCoords,
                                          normal, viewDirection);
            }
            else {
                color += lightingEquation( lights[i], material, v_eyeCoords,
                                          -normal, viewDirection);
            }
        }
    }
    gl_FragColor = vec4(color, 1);
}
</script>

<script src="gl-matrix.js"> </script>
<script src="basic-object-models-IFS.js"> </script>
<script src="teapot-model-IFS.js"> </script>

<script>

"use strict";

var gl; // The WebGL context
var canvas; // The canvas where gl draws

var a_coords_loc; // Location of the a_coords attribute variable in the shader program
var a_normal_loc; // Location of a_normal attribute

var u_modelview; // Locations for uniform matrices
var u_projection;
var u_normalMatrix;

var u_material; // An object holds uniform locations for the material.
var u_lights; // An array of objects that holds uniform locations for light properties.

var projection = mat4.create(); // projection matrix
var modelview = mat4.create(); // modelview matrix
var normalMatrix = mat3.create(); // matrix for transforming normal vectors

```

```

var frameNumber = 0; // frame number during animation

var cone, cylinder, cube, teapot; // Basic objects, created using function createModel and basic-object-models-IFS.js.
    // The cube is 1 unit on each side and is centered at (0,0,0).
    // the cone and cylinder have diameter 1 and height 1 and are centered at
    // (0,0,0), with their axes aligned along the z-axis.

var matrixStack = []; // A stack of matrices for implementing hierarchical graphics

var currentColor = [1,1,1]; // The current diffuse color; render() functions in the basic objects set
    // the diffuse color to currentColor when it is called before drawing the object
    // Specular color properties, which don't change, are set in initGL()

var rotateX = 0, rotateY = 0; // Overall rotation of model, in radians, set by mouse dragging.

/**
 * Draws the image, which consists of either the "world" or a closeup of the "car".
 */

function vane() {
    pushMatrix();
    currentColor = [0.690, 0.980, 0.988];
    mat4.rotateY(modelview,modelview, Math.PI/2);
    mat4.scale(modelview,modelview,[0.5, 0.7, 3.1]);
    cone.render();
    popMatrix();

    pushMatrix();
    currentColor = [0.690, 0.980, 0.988];
    mat4.translate(modelview, modelview, [-2.3, 0, 0]);
    mat4.rotateY(modelview,modelview, Math.PI/2);
    mat4.rotateX(modelview,modelview, Math.PI);
    mat4.scale(modelview,modelview,[0.5, 0.7, 1.5]);
    cone.render();
    popMatrix();
}

var rotateEachFrame = 0;

function draw() {
    gl.clearColor(0,0,0,1);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(projection, Math.PI/4, 1, 1, 50);
    gl.uniformMatrix4fv(u_projection, false, projection );

    mat4.lookAt(modelview, [0,0,25], [0,0,0], [0,1,0]);

    mat4.rotateX(modelview,modelview,rotateX);
    mat4.rotateY(modelview,modelview,rotateY);

    pushMatrix();
    currentColor = [0.984, 0.156, 0.403];
    mat4.translate(modelview, modelview, [0, -5, 0]);
    mat4.scale(modelview,modelview,[5, 0.5, 5]);
    cube.render();
    popMatrix();

    pushMatrix();
    currentColor = [0.976, 0.847, 0.584];
    mat4.translate(modelview, modelview, [0, 0, 0]);
    mat4.rotateX(modelview,modelview, Math.PI/2);
    mat4.scale(modelview,modelview,[0.4, 0.4, 10]);
    cylinder.render();
    popMatrix();

    pushMatrix();
    mat4.translate(modelview, modelview, [0, 4.5, 0]);
    mat4.rotateZ(modelview, modelview, -Math.PI/2 - rotateEachFrame);

```

```

    pushMatrix();
    mat4.rotateZ(modelview,modelview, Math.PI/2);
    mat4.translate(modelview, modelview, [3, 0, 0.5]);
    vane();
    popMatrix();

    pushMatrix();
    mat4.rotateZ(modelview,modelview, Math.PI/2 - 2*Math.PI/3);
    mat4.translate(modelview, modelview, [3, 0, 0.5]);
    vane();
    popMatrix();

    pushMatrix();
    mat4.rotateZ(modelview,modelview, Math.PI/2 + 2*Math.PI/3);
    mat4.translate(modelview, modelview, [3, 0, 0.5]);
    vane();
    popMatrix();
popMatrix();

pushMatrix();
currentColor = [0,1,0];
mat4.translate(modelview, modelview, [1.5,-4.15,1.5]);
mat4.rotateY(modelview, modelview, -Math.PI/4);
mat4.scale(modelview, modelview, [0.08, 0.08, 0.08]);
teapot.render();
popMatrix();

} // end draw();

/**
 * Push a copy of the current modelview matrix onto the matrix stack.
 */
function pushMatrix() {
    matrixStack.push( mat4.clone(modelview) );
}

/**
 * Restore the modelview matrix to a value popped from the matrix stack.
 */
function popMatrix() {
    modelview = matrixStack.pop();
}

/**
 * Create one of the basic objects. The modelData holds the data for
 * an IFS using the structure from basic-object-models-IFS.js. This function
 * creates VBOs to hold the coordinates, normal vectors, and indices
 * from the IFS, and it loads the data into those buffers. The function
 * creates a new object whose properties are the identities of the
 * VBOs. The new object also has a function, render(), that can be called to
 * render the object, using all the data from the buffers. That object
 * is returned as the value of the function. (The second parameter,
 * xtraTranslate, is there because this program was ported from a Java
 * version where cylinders were created in a different position, with
 * the base on the xy-plane instead of with their center at the origin.
 * The xtraTranslate parameter is a 3-vector that is applied as a
 * translation to the rendered object. It is used to move the cylinders
 * into the position expected by the code that was ported from Java.)
 */
function createModel(modelData) {
    var model = {};
    model.coordsBuffer = gl.createBuffer();
    model.normalBuffer = gl.createBuffer();
    model.indexBuffer = gl.createBuffer();
    model.count = modelData.indices.length;
    gl.bindBuffer(gl.ARRAY_BUFFER, model.coordsBuffer);

```

```

gl.bufferData(gl.ARRAY_BUFFER, modelData.vertexPositions, gl.STATIC_DRAW);
gl.bindBuffer(gl.ARRAY_BUFFER, model.normalBuffer);
gl.bufferData(gl.ARRAY_BUFFER, modelData.vertexNormals, gl.STATIC_DRAW);
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, model.indexBuffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, modelData.indices, gl.STATIC_DRAW);
model.render = function() { // This function will render the object.
    // Since the buffer from which we are taking the coordinates and normals
    // changes each time an object is drawn, we have to use gl.vertexAttribPointer
    // to specify the location of the data. And to do that, we must first
    // bind the buffer that contains the data. Similarly, we have to
    // bind this object's index buffer before calling gl.drawElements.
    gl.bindBuffer(gl.ARRAY_BUFFER, this.coordsBuffer);
    gl.vertexAttribPointer(a_coords_loc, 3, gl.FLOAT, false, 0, 0);
    gl.bindBuffer(gl.ARRAY_BUFFER, this.normalBuffer);
    gl.vertexAttribPointer(a_normal_loc, 3, gl.FLOAT, false, 0, 0);
    gl.uniform3fv(u_material.diffuseColor, currentColor);
    gl.uniformMatrix4fv(u_modelview, false, modelview );
    mat3.normalFromMat4(normalMatrix, modelview);
    gl.uniformMatrix3fv(u_normalMatrix, false, normalMatrix);
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, this.indexBuffer);
    gl.drawElements(gl.TRIANGLES, this.count, gl.UNSIGNED_SHORT, 0);
    if (this.xtraTranslate) {
        popMatrix();
    }
}
return model;
}

```

```

/**
 * Creates a program for use in the WebGL context gl, and returns the
 * identifier for that program. If an error occurs while compiling or
 * linking the program, an exception of type String is thrown. The error
 * string contains the compilation or linking error. If no error occurs,
 * the program identifier is the return value of the function.
 * The second and third parameters are the id attributes for <script>
 * elements that contain the source code for the vertex and fragment
 * shaders.
 * If the third parameter is present, it should be the name of an
 * attribute variable in the shader program, and the attribute should be
 * one that is always used. The attribute will be assigned attribute
 * number 0. This is done because it is suggested that there should
 * always be an attribute number 0 in use.
 */
function createProgram(gl, vertexShaderID, fragmentShaderID, attribute0) {
    function getTextContent( elementID ) {
        // This nested function retrieves the text content of an
        // element on the web page. It is used here to get the shader
        // source code from the script elements that contain it.
        var element = document.getElementById(elementID);
        var node = element.firstChild;
        var str = "";
        while (node) {
            if (node.nodeType === 3) // this is a text node
                str += node.textContent;
            node = node.nextSibling;
        }
        return str;
    }
    try {
        var vertexShaderSource = getTextContent( vertexShaderID );
        var fragmentShaderSource = getTextContent( fragmentShaderID );
    }
    catch (e) {
        throw "Error: Could not get shader source code from script elements.";
    }
    var vsh = gl.createShader( gl.VERTEX_SHADER );
    gl.shaderSource(vsh,vertexShaderSource);
    gl.compileShader(vsh);
    if ( ! gl.getShaderParameter(vsh, gl.COMPILE_STATUS) ) {

```

```

        throw "Error in vertex shader: " + gl.getShaderInfoLog(vsh);
    }
    var fsh = gl.createShader( gl.FRAGMENT_SHADER );
    gl.shaderSource(fsh, fragmentShaderSource);
    gl.compileShader(fsh);
    if ( ! gl.getShaderParameter(fsh, gl.COMPILE_STATUS) ) {
        throw "Error in fragment shader: " + gl.getShaderInfoLog(fsh);
    }
    var prog = gl.createProgram();
    gl.attachShader(prog,vsh);
    gl.attachShader(prog, fsh);
    if (attribute0) {
        gl.bindAttribLocation(prog,0,attribute0);
    }
    gl.linkProgram(prog);
    if ( ! gl.getProgramParameter( prog, gl.LINK_STATUS) ) {
        throw "Link error in program: " + gl.getProgramInfoLog(prog);
    }
    return prog;
}

/* Initialize the WebGL context. Called from init() */
function initGL() {
    var prog = createProgram(gl,"vshader-source","fshader-source", "a_coords");
    gl.useProgram(prog);
    gl.enable(gl.DEPTH_TEST);

    /* Get attribute and uniform locations */

    a_coords_loc = gl.getAttribLocation(prog, "a_coords");
    a_normal_loc = gl.getAttribLocation(prog, "a_normal");
    gl.enableVertexAttribArray(a_coords_loc);
    gl.enableVertexAttribArray(a_normal_loc);

    u_modelview = gl.getUniformLocation(prog, "modelview");
    u_projection = gl.getUniformLocation(prog, "projection");
    u_normalMatrix = gl.getUniformLocation(prog, "normalMatrix");
    u_material = {
        diffuseColor: gl.getUniformLocation(prog, "material.diffuseColor"),
        specularColor: gl.getUniformLocation(prog, "material.specularColor"),
        specularExponent: gl.getUniformLocation(prog, "material.specularExponent")
    };
    uLights = new Array(4);
    for (var i = 0; i < 4; i++) {
        uLights[i] = {
            enabled: gl.getUniformLocation(prog, "lights[" + i + "].enabled"),
            position: gl.getUniformLocation(prog, "lights[" + i + "].position"),
            color: gl.getUniformLocation(prog, "lights[" + i + "].color")
        };
    }

    gl.uniform3f( u_material.diffuseColor, 1, 1, 1 ); // set to white as a default.
    gl.uniform3f( u_material.specularColor, 0.1, 0.1, 0.1 ); // specular properties won't change
    gl.uniform1f( u_material.specularExponent, 32 );

    for (var i = 1; i < 4; i++) { // set defaults for lights
        gl.uniform1i( uLights[i].enabled, 0 );
        gl.uniform4f( uLights[i].position, 0, 0, 1, 0 );
        gl.uniform3f( uLights[i].color, 1,1,1 );
    }

    // Set up lights here; they won't be changed. Lights are fixed in eye coordinates.

    gl.uniform1i( uLights[0].enabled, 1 ); // light is a "viewpoint light"
    gl.uniform4f( uLights[0].position, 0,0,0,1 ); // positional, at viewpoint
    gl.uniform3f( uLights[0].color, 0.6, 0.6, 0.6 );

    gl.uniform1i( uLights[1].enabled, 1 ); // light 1 is a dimmer light shining from above

```



```

gl.uniform4f( u_lights[0].position, 0,1,0,0 ); // diretionsl, from directino of positive y-axis
gl.uniform3f( u_lights[0].color, 0.4, 0.4, 0.4 );

gl.uniform1i( u_lights[2].enabled, 0 ); // lightes 2 and 3 are not used.

// Note: position and spot direction for lights 1 to 4 are managed by modeling transforms.

} // end initGL()

```

```

//----- rotation by mouse -----

```

```

function mouseDown(evt) {
    var prevX, prevY;
    prevX = evt.clientX;
    prevY = evt.clientY;
    canvas.addEventListener("mousemove",mouseMove);
    document.addEventListener("mouseup",mouseUp);
    function mouseMove(evt) {
        var dx = evt.clientX - prevX;
        var dy = evt.clientY - prevY;
        rotateX += dy/200;
        rotateY += dx/200;
        prevX = evt.clientX;
        prevY = evt.clientY;
        draw();
    }
    function mouseUp(evt) {
        canvas.removeEventListener("mousemove",mouseMove);
        document.removeEventListener("mouseup",mouseUp);
    }
}

```

```

//----- animation -----

```

```

var animating = false;

function frame() {
    if (animating) {
        rotateEachFrame=rotateEachFrame+Math.PI*0.01;
        frameNumber += 1;
        draw();

        requestAnimationFrame(frame);
    }
}

function setIsAnimating() {
    var run = document.getElementById("animCheck").checked;
    if (run !== animating) {
        animating = run;
        if (animating)
            requestAnimationFrame(frame);
    }
}

```

```

//-----

```

```

/**
 * initialization function that will be called when the page has loaded
 */
function init() {
    try {
        canvas = document.getElementById("webglcanvas");
        gl = canvas.getContext("webgl");
        if ( ! gl ) {
            throw "Browser does not support WebGL";
        }
    }
}

```

```

    }
    catch (e) {
        document.getElementById("message").innerHTML =
            "<p>Sorry, could not get a WebGL graphics context.</p>";
        return;
    }
    try {
        initGL(); // initialize the WebGL graphics context
    }
    catch (e) {
        document.getElementById("message").innerHTML =
            "<p>Sorry, could not initialize the WebGL graphics context:" + e + "</p>";
        return;
    }
    document.getElementById("animCheck").checked = false;
    document.getElementById("animCheck").addEventListener("change", setIsAnimating);
    document.getElementById("reset").addEventListener("click", function() { rotateEachFrame = 0; rotateX = rotateY = 0; draw(); });
    canvas.addEventListener("mousedown", mouseDown);

    cone = createModel(uvCone()); // create the basic objects
    cylinder = createModel(uvCylinder()); // uvCone(), uvCylinder(), and cube()
    cube = createModel(cube()); // are defined in basic-object-models-IFS.js
    teapot = createModel(teapotModel);
    draw();
}

</script>
</head>
<body onload="init()">

<h2>DiskWorld 2: WebGL Lighting and Hierarchical Modeling</h2>

<noscript> <hr> <h3>This page requires Javascript and a web browser that supports WebGL</h3> <hr> </noscript>

<p id="message" style="font-weight:bold">Drag your mouse on the model to rotate it.</p>

<p>
    <label><input type="checkbox" id="animCheck">Animate</label>
    <button id="reset" style="margin-left:40px">Reset</button>
</p>

<div>

    <canvas width=800 height=800 id="webglcanvas" style="background-color:blue"></canvas>

</div>
</body>
</html>

```