**SPRAWOZDANIE**

Zajęcia: Grafika komputerowa

Prowadzący: prof. dr hab. Vasyl Martsenyuk

**Laboratorium II**

Data 28.05.2025

**Temat: ZadanieWebGL3d**

**Wariant 8 + 4**

Jakub Bąk

Informatyka I stopień,

stacjonarne,

4 semestr,

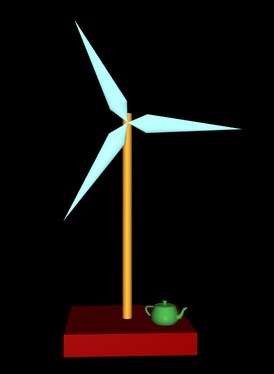
Gr.3b

# Polecenie

Plik lab12.html pokazuje mały sześcian, który można obrócić, przeciągając myszą na płótnie. Zadaniem jest zastąpienie sześcianu 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. (Dodanie czajniczka, który znajduje się na podstawie, jest konieczne dla uzyskania oceny "5")

Program zawiera trzy zmienne instancji reprezentujące podstawowe obiekty: cube, cone, cylinder. Te zmienne mają metody instancji cube.render(), cone.render(), cylinder.render(), które można wywołać w celu narysowania obiektów. Obiekty nietransformowane mają rozmiar 1 we wszystkich trzech kierunkach i mają swój środek na (0,0,0). Oś stożka i oś cylindra są wyrównane wzdłuż osi Z. Wszystkie obiekty na scenie powinny być przekształconymi wersjami podstawowych obiektów (lub podstawowego obiektu czajnika).

# Zdjęcie z Przykładu:

****

# Wykorzystane komendy:

Link do github: <https://github.com/Szeladin/grafika.git>

Kod Programu:

Odpowiada za wiatrak z dwunastoma skrzydłami i czajnik

<!DOCTYPE html>

<html lang="pl">

<head>

<meta charset="UTF-8">

<title>Lab 12</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 tolds 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; // Overal 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, 0.8, 0];

mat4.rotateY(modelview,modelview, Math.PI/2);

mat4.scale(modelview,modelview,[0.5, 0.7, 3.1]);

cone.render();

popMatrix();

pushMatrix();

currentColor = [0, 0.8, 0];

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);

for (var i = 0; i < 12; i++) {

pushMatrix();

mat4.rotateZ(modelview, modelview, i \* 2 \* Math.PI / 12);

mat4.translate(modelview, modelview, [3, 0, 0.5]);

vane();

popMatrix();

}

popMatrix();

pushMatrix();

currentColor = [0,0.3,1];

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();

}

function pushMatrix() {

matrixStack.push( mat4.clone(modelview) );

}

function popMatrix() {

modelview = matrixStack.pop();

}

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.

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;

}

function createProgram(gl, vertexShaderID, fragmentShaderID, attribute0) {

function getTextContent( elementID ) {

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;

}

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")

};

u\_lights = new Array(4);

for (var i = 0; i < 4; i++) {

u\_lights[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( u\_lights[i].enabled, 0 );

gl.uniform4f( u\_lights[i].position, 0, 0, 1, 0 );

gl.uniform3f( u\_lights[i].color, 1,1,1 );

}

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

gl.uniform1i( u\_lights[0].enabled, 1 ); // light is a "viewpoint light"

gl.uniform4f( u\_lights[0].position, 0,0,0,1 ); // positional, at viewpoint

gl.uniform3f( u\_lights[0].color, 0.6, 0.6, 0.6 );

gl.uniform1i( u\_lights[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.

}

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);

}

}

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);

}

}

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());

cylinder = createModel(uvCylinder());

cube = createModel(cube());

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>

Wynik końcowy:



# Wyniki i wnioski:

Na podstawie otrzymanych wyników i wykonanej pracy byłem wstanie zrozumieć podstawy WEBGL3d i zobaczyć, jak tworzyć animacje z wykorzystaniem 3d.