SPRAWOZDANIE

Zajęcia: Grafika komputerowa
Prowadzący: mgr inż. Mikołaj Grygiel

Laboratorium 11
21.05.2025
Temat: "Grafika 3D w bibliotece WebGL/GLSL "
Wariant 6

Bartłomiej Mędrzak
s61324
Informatyka I stopień,
stacjonarne,
4 semestr,
Gr.1A

1. 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).

2. Wprowadzane dane:

Zastąpienie sześcianiu wiatrakiem o n łopatkach tak jak pokazano na obrazku

3. Wykorzystane komendy:

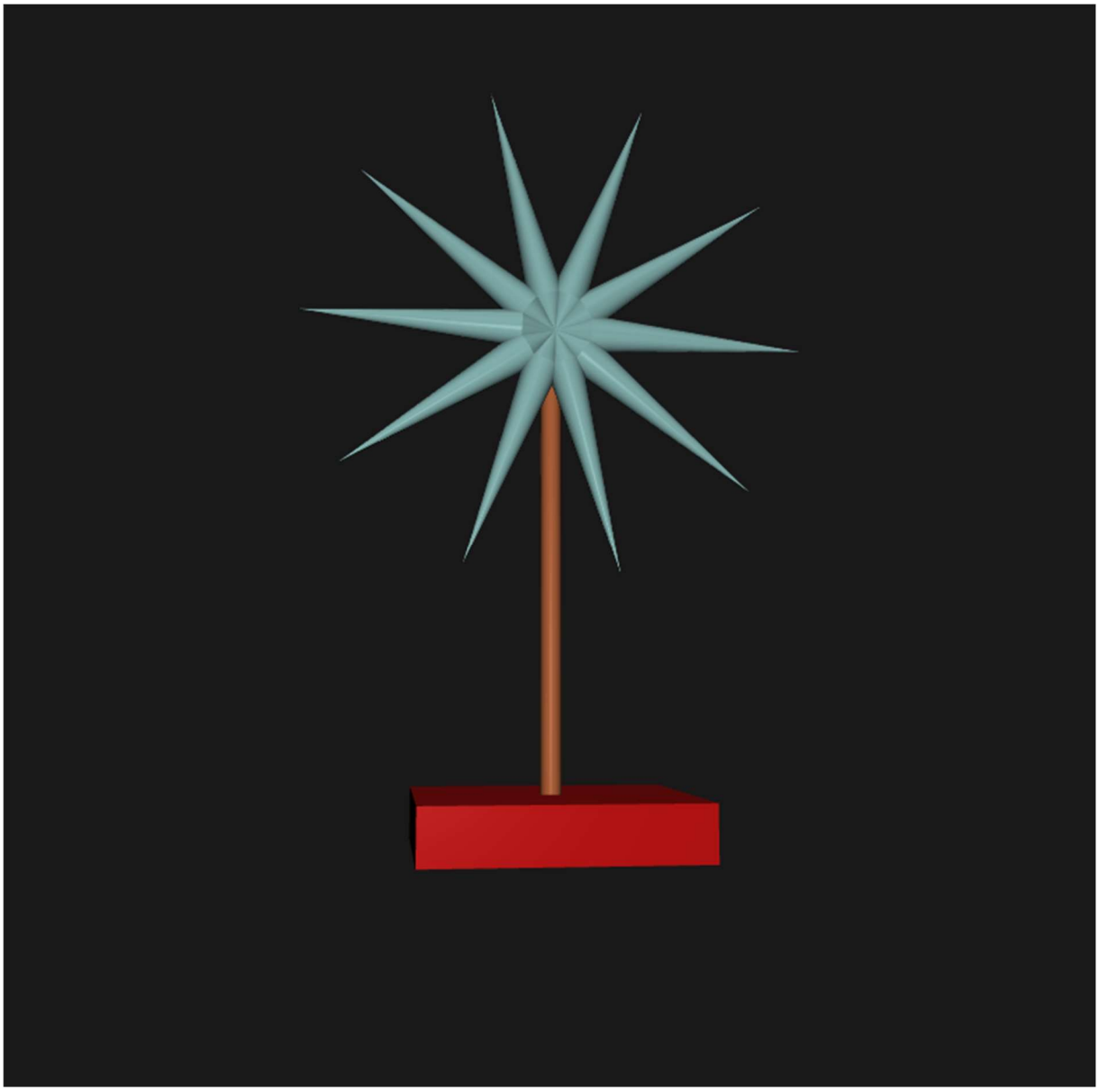
```
function createWingComponent()
{
    pushTransform();
    activeColor = [180/255, 0.95, 240/255];
    mat4.rotateY(mainModelViewMatrix, mainModelViewMatrix, Math.PI / 2);
    mat4.translate(mainModelViewMatrix, mainModelViewMatrix, [0, 0, 2.7]);
    mat4.scale(mainModelViewMatrix, mainModelViewMatrix, [0.7, 0.7, 4.1]);
    shape_cone.draw();
    popTransform();

    pushTransform();
    activeColor = [180/255, 0.95, 240/255];
    mat4.translate(mainModelViewMatrix, mainModelViewMatrix, [0.3, 0, 0]);
    mat4.rotateY(mainModelViewMatrix, mainModelViewMatrix, Math.PI / 2);
    mat4.rotateX(mainModelViewMatrix, mainModelViewMatrix, Math.PI );
    mat4.scale(mainModelViewMatrix, mainModelViewMatrix,[0.7, 0.7, 0.7]);
    shape_cone.draw();
    popTransform();
}
let numberOfWings = 10;
var animationRotationSpeed = 10;
```

```
function renderScene() {
    webGL.clearColor(0.1, 0.1, 0.1, 1);
    webGL.clear(webGL.COLOR_BUFFER_BIT | webGL.DEPTH_BUFFER_BIT);

    mat4.perspective(mainProjectionMatrix, Math.PI / 4, 1, 1, 50); // Zmieniony FOV
    webGL.uniformMatrix4fv(unif_projectionMatrix, false, mainProjectionMatrix );

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

    mat4.rotateX(mainModelViewMatrix,mainModelViewMatrix,viewRotationX);
    mat4.rotateY(mainModelViewMatrix,mainModelViewMatrix,viewRotationY);

    pushTransform();
    activeColor = [0.95, 0.1, 0.1];
    mat4.translate(mainModelViewMatrix, mainModelViewMatrix, [0, -5, 0]);
    mat4.scale(mainModelViewMatrix,mainModelViewMatrix,[5, 1, 5]);
    shape_cube.draw();
    popTransform();


    pushTransform();
    activeColor = [0.9, 0.5, 0.3];
    mat4.translate(mainModelViewMatrix, mainModelViewMatrix, [0, 0, 0]);
    mat4.rotateX(mainModelViewMatrix,mainModelViewMatrix, Math.PI * 0.5);
    mat4.scale(mainModelViewMatrix,mainModelViewMatrix,[0.4, 0.4, 10]);
    shape_cylinder.draw();
    popTransform();

    pushTransform();
    mat4.translate(mainModelViewMatrix, mainModelViewMatrix, [-2.9, 2, 0.2]);
    pushTransform();
    mat4.translate(mainModelViewMatrix, mainModelViewMatrix, [2.9, 2, 0]);
    mat4.rotate(mainModelViewMatrix, mainModelViewMatrix, animationRotationSpeed, [0, 0, 1]);
    mat4.translate(mainModelViewMatrix, mainModelViewMatrix, [2.9, -2, 0]);

    for(let k = 0; k < numberOfWings; k++){
        pushTransform();
        mat4.translate(mainModelViewMatrix, mainModelViewMatrix, [-3, 1.95, 0]);
        mat4.rotateZ(mainModelViewMatrix, mainModelViewMatrix, k * (360 / numberOfWings) * (Math.PI / 180));
        mat4.rotateY(mainModelViewMatrix, mainModelViewMatrix, Math.PI);
        createWingComponent();
        popTransform();
    }
    popTransform();
    popTransform();
}
```

```
function pushTransform() {
    transformMatrixStack.push( mat4.clone(mainModelViewMatrix) );
}


function popTransform() {
    mainModelViewMatrix = transformMatrixStack.pop();
}


function buildGeometricModel(geometricData) {
    var newModel = {};
    newModel.vertexBuffer = webGL.createBuffer();
    newModel.normalBuffer = webGL.createBuffer();
    newModel.indexBuffer = webGL.createBuffer();
    newModel.vertexCount = geometricData.indices.length;

    webGL.bindBuffer(webGL.ARRAY_BUFFER, newModel.vertexBuffer);
    webGL.bufferData(webGL.ARRAY_BUFFER, geometricData.vertexPositions, webGL.STATIC_DRAW);
    webGL.bindBuffer(webGL.ARRAY_BUFFER, newModel.normalBuffer);
    webGL.bufferData(webGL.ARRAY_BUFFER, geometricData.vertexNormals, webGL.STATIC_DRAW);
    webGL.bindBuffer(webGL.ELEMENT_ARRAY_BUFFER, newModel.indexBuffer);
    webGL.bufferData(webGL.ELEMENT_ARRAY_BUFFER, geometricData.indices, webGL.STATIC_DRAW);

    newModel.draw = function() {
        webGL.bindBuffer(webGL.ARRAY_BUFFER, this.vertexBuffer);
        webGL.vertexAttribPointer(attr_vertexPosition, 3, webGL.FLOAT, false, 0, 0);
        webGL.bindBuffer(webGL.ARRAY_BUFFER, this.normalBuffer);
        webGL.vertexAttribPointer(attr_vertexNormal, 3, webGL.FLOAT, false, 0, 0);

        webGL.uniform3fv(unif_materialData.diffuseRGB, activeColor);
        webGL.uniformMatrix4fv(unif_modelViewMatrix, false, mainModelViewMatrix );
        mat3.normalFromMat4(mainNormalMatrix, mainModelViewMatrix);
        webGL.uniformMatrix3fv(unif_normalMatrix, false, mainNormalMatrix);

        webGL.bindBuffer(webGL.ELEMENT_ARRAY_BUFFER, this.indexBuffer);
        webGL.drawElements(webGL.TRIANGLES, this.vertexCount, webGL.UNSIGNED_SHORT, 0);
    }
    return newModel;
}

 var animationIsRunning = false;

function animationStep() {
    if (animationIsRunning) {
        animationRotationSpeed = animationRotationSpeed + Math.PI * 0.012;
        animationFrameCounter++;
        renderScene();
        requestAnimationFrame(animationStep);
    }
 }

function toggleAnimationState() {
    var isChecked = document.getElementById("animationToggle").checked;
    if (isChecked != animationIsRunning) {
        animationIsRunning = isChecked;
        if (animationIsRunning)
            requestAnimationFrame(animationStep);
    }
 }
```

https://github.com/castehard33/Grafika_Komputerowa/tree/main/11%20Grafika%203D%20w%20bibliotece%20WebGL%20GLSL

4. Wynik działania:



5. Wnioski:

Implementacja złożonego, animowanego obiektu 3D, jakim jest wiatrak, pozwoliła na praktyczne przećwiczenie kluczowych technik modelowania hierarchicznego w WebGL. Wykorzystanie stosu macierzy transformacji okazało się niezbędne do poprawnego pozycjonowania i orientowania poszczególnych komponentów wiatraka (podstawy, wieży, a zwłaszcza złożonych łopatek) względem siebie oraz względem sceny globalnej.