Міністерство Науки і Освіти України

Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського”

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

**Звіт з розрахунково-графічної роботи**

Дисципліна: “ Візуалізація графічної та геометричної інформації”

Варіант 8

**Виконала**:

студентка 5 курсу групи ТР-41мп Клебан Д. А.

**Перевірив:**

доц. Демчишин А. А.

Київ

2024

**РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА**

**ЗАВДАННЯ**

Реалізувати обертання текстури (координат текстури) навколо визначеної користувачем точки. Забезпечити можливість пересування точки в просторі поверхні (u, v) за допомогою використання клавіатури. Клавіші A і D повинні переміщувати точку вздовж параметра u, а клавіші W і S – вздовж параметра v.

**ТЕОРІЯ**

**Текстура** – це зображення (растровий формат), що застосовується до полігональної моделі шляхом накладання, з метою деталізації моделі, додаючи до неї кольори, рельєфи, відблиски чи колірне забарвлення.

**Текстурування** – це процес нанесення зображення (текстури) на поверхню тривимірного об’єкта. Текстурування дозволяє надати поверхні реалістичності і схожості з реальним об’єктом.

Координати текстури (u, v) визначають, як зображення буде накладено на об'єкт, де u та v відповідають горизонтальній і вертикальній складовим текстури відповідно. Координати текстури в WebGL зазвичай нормалізовані і перебувають в діапазоні від 0 до 1. Це означає, що значення 0 для u або v відповідає одному краю текстури (зазвичай лівому чи нижньому), а значення 1– протилежному краю (правому чи верхньому).

**Типи текстур.**

Існують різні види текстур, які виконують свої специфічні функції:

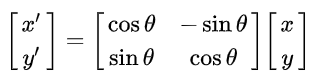
* Дифузна текстура (Diffuse Map): визначає основний колір об’єкта, тобто, як поверхня виглядає під стандартним освітленням.
* Дзеркальна текстура (Specular Map): відповідає за блиск поверхні та моделює, наскільки сильно вона відбиває світло.
* Нормальна текстура (Normal Map): створює ілюзію рельєфу на гладких поверхнях, імітуючи тінь і світло для дрібних деталей без зміни геометрії моделі.

**Обертання текстури**.

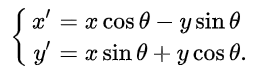
Обертання текстури дозволяє змінити орієнтацію зображення на поверхні об'єкта. Це досягається шляхом застосування матриці/формул обертання до текстурних координат (u, v). Обертання можна виконати як над центром координат так і над заданою точкою.

Для реалізації обертання текстури використовується матриця або система перетворень, що включає в себе поворот відносно заданої точки. Спочатку координати текстури переміщуються таким чином, щоб точка обертання збіглася з початком координат. Потім застосовується матриця обертання, що враховує заданий кут. На завершення обернена текстура повертається на початкові координати відносно яких здійснювалось обертання.

Для проведення обертання з використанням матриць, точку записують у вигляді вектора, потім множать на матрицю від кута θ:

,

– координати точки після обертання, можуть бути записані так:



**ДЕТАЛІ РЕАЛІЗАЦІЇ**

Додання нововведень базується на виконаній 3 лабораторній роботі, яка полягала у накладанні текстури на поверхню сполучення коаксіального циліндра та конуса. У даній роботі додано можливість обертання текстури в координатах u та v навколо заданої користувачем точки. Переміщення цієї точки здійснюється за допомогою натискання клавіш на клавіатурі, а кут обертання можна задавати через інтерфейсний елемент – повзунок.

В HTML додано нові елементи, зокрема повзунок, який використовується для задавання кута обертання текстури, а також поле для зазначення координат точки, яка буде центром обертання (рис. 1).

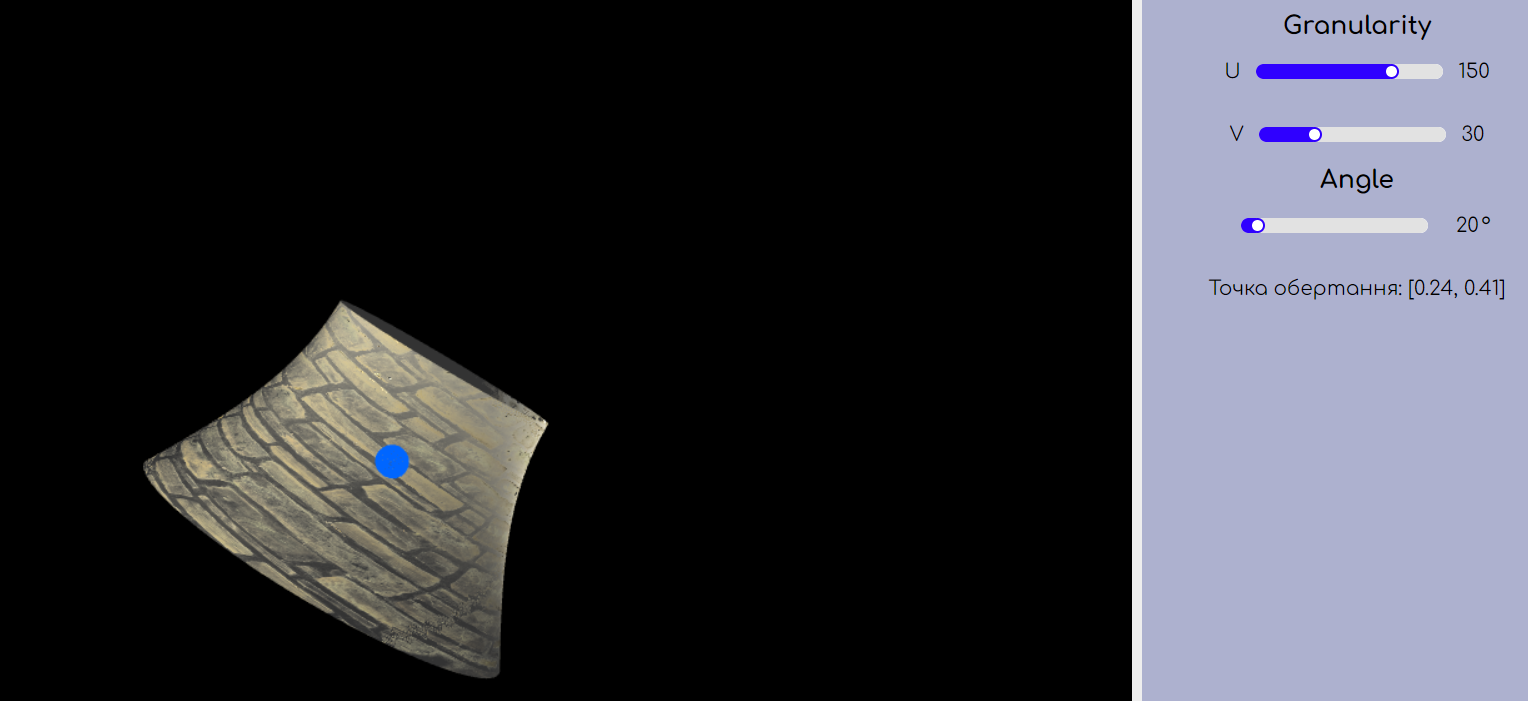


Рисунок 1 – Нововведення інтерфейсу

Було визначено функцію для створення графічного відображення точки, яка слугуватиме центром обертання. Для переміщення цієї точки використовуються клавіші A, D, S, W. При натисканні клавіш W та S (англійська розкладка) змінюється координата u точки, а при натисканні клавіш D та A – відповідно координата v. Зміна координат відбувається з кроком 0.01 за кожне натискання. Також, при зміні, відображається значення координат u та v поточного місцезнаходження цієї точки в відповідному полі на інтерфейсі.

Обертання текстури здійснюється наступним чином: спочатку отримується кут обертання, заданий користувачем, за допомогою повзунка в інтерфейсі, після чого текстурні координати кожної вершини поверхні перетворюються в нормалізований діапазон від 0 до 1. Далі ці координати переміщуються в початок системи координат (0,0) та виконується обертання навколо початку координат за допомогою відповідних математичних формул на заданий користувачем кут. На заключному етапі обернена текстура переміщується в початкове положення, відповідно до точки обертання. Реалізація даного функціоналу продемонстрована на рисунку 2.

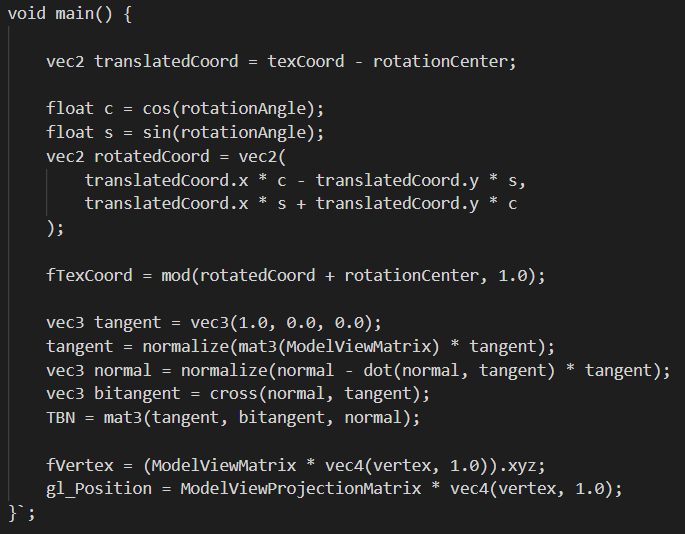


Рисунок 2 – Обертання відносно точки

**ІНСТРУКЦІЯ КОРИСТУВАЧА**

При запусканні програми користувач може бачити наступне: поверхню сполучення коаксіального циліндра та конуса з накладеними текстурами, інтерфейсні елементи такі як повзунки для задання зернистості поверхні, повзунок для задання кута обертання та інформаційне поле місцезнаходження точки обертання.

За допомогою використання мишки користувач може обертати фігуру навколо її центру.

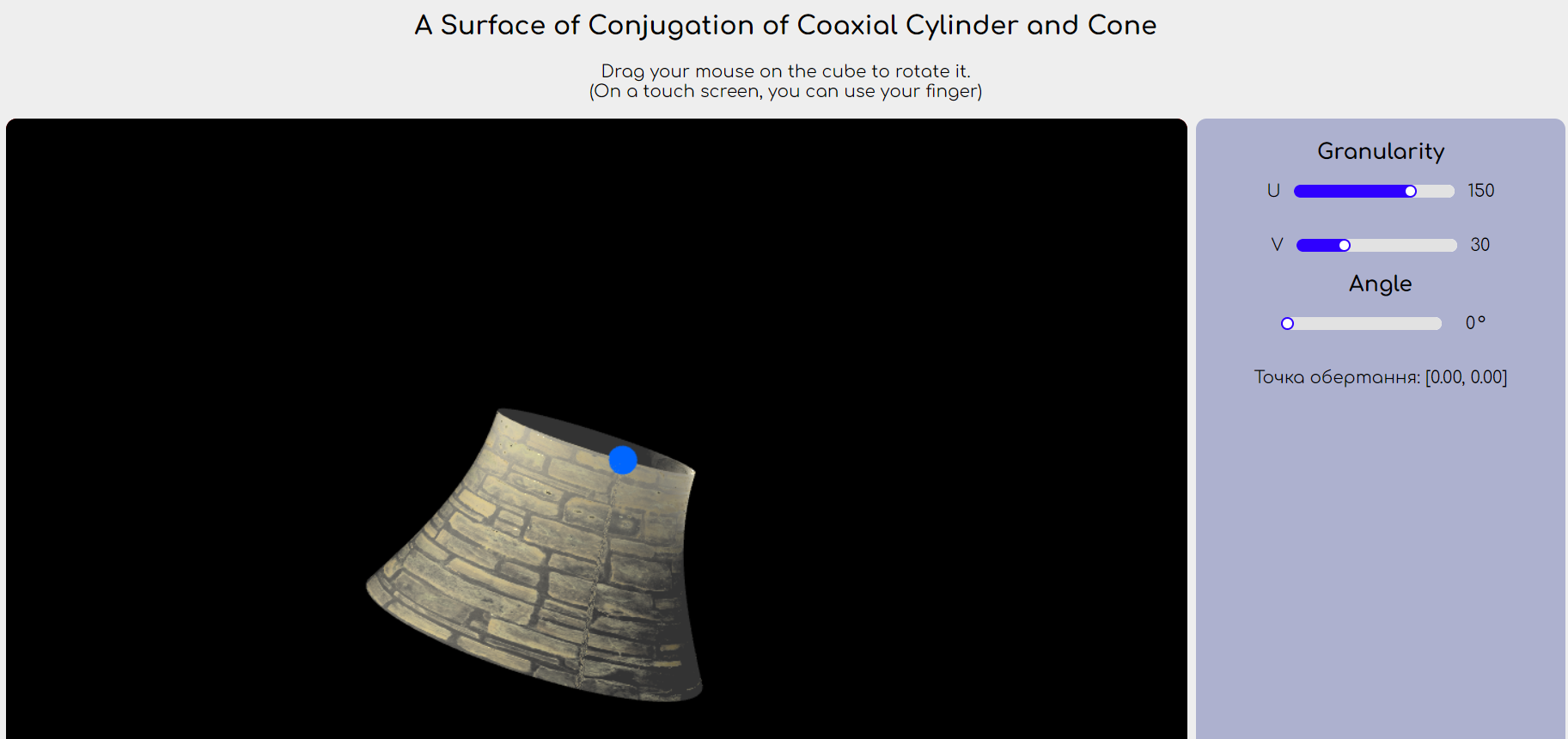


Рисунок 3 – Інтерфейс програми

За допомогою повзунка Angle користувач може обирати кут, на який бажає обернути текстуру. Нижче повзунка розташоване інформаційне поле, в якому відображаються координати точки, що слугує центром обертання.

При натисканні клавіш W та S (з англійською розкладкою) користувач має можливість змінювати координату u точки, навколо якої здійснюється обертання. Клавіші D та A відповідають за зміну координати v. Кожна зміна відбувається з кроком 0.01 при кожному натисканні. На рисунку 4 зображено поверхню та переміщену точна на координати [0.14, 0.39].

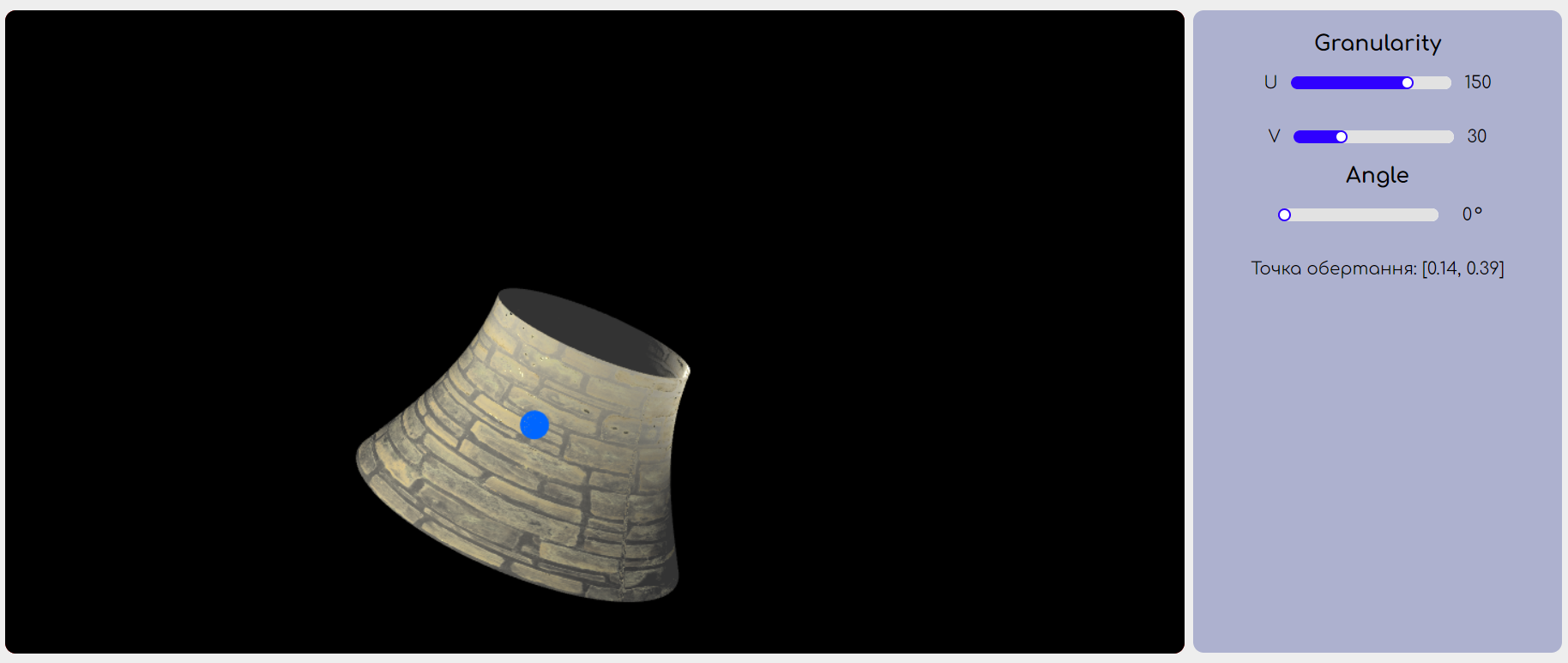


Рисунок 4 – Переміщення точки

При натисканні та переміщенні повзунка Angle користувач визначає кут обертання текстури. На рисунку 5 показано текстуру, обернену на 20° з центром обертання в точці [0.31, 0.41].

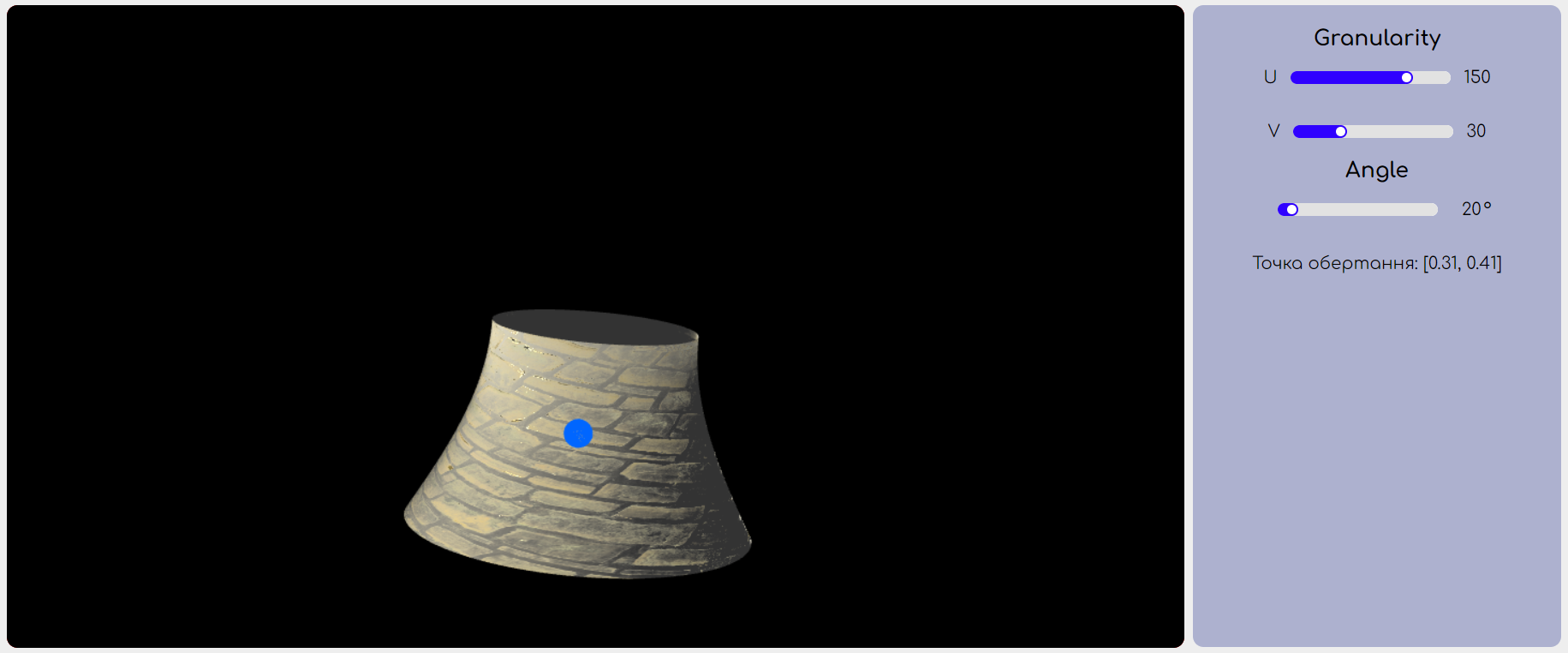


Рисунок 5 – Обертання навколо точки обраної користувачем

**ПРОГРАМНИЙ КОД**

const vertexShaderSource = `

attribute vec3 vertex;

attribute vec3 normal;

attribute vec2 texCoord;

uniform float textureScale;

uniform vec2 rotationCenter;

uniform float rotationAngle;

uniform mat4 ModelViewMatrix;

uniform mat4 ModelViewProjectionMatrix;

varying vec3 fVertex;

varying vec2 fTexCoord;

varying mat3 TBN;

void main() {

vec2 translatedCoord = texCoord - rotationCenter;

float c = cos(rotationAngle);

float s = sin(rotationAngle);

vec2 rotatedCoord = vec2(

translatedCoord.x \* c - translatedCoord.y \* s,

translatedCoord.x \* s + translatedCoord.y \* c

);

fTexCoord = mod(rotatedCoord + rotationCenter, 1.0);

vec3 normalizedNormal = normalize(mat3(ModelViewMatrix) \* normal);

vec3 tangent = vec3(1.0, 0.0, 0.0);

tangent = normalize(tangent - dot(tangent, normalizedNormal) \* normalizedNormal);

vec3 bitangent = cross(tangent, normalizedNormal);

TBN = mat3(tangent, bitangent, normalizedNormal);

fVertex = (ModelViewMatrix \* vec4(vertex, 1.0)).xyz;

gl\_Position = ModelViewProjectionMatrix \* vec4(vertex, 1.0);

}`;

// Fragment shader

const fragmentShaderSource = `

#ifdef GL\_FRAGMENT\_PRECISION\_HIGH

precision highp float;

#else

precision mediump float;

#endif

varying vec3 fVertex;

varying vec2 fTexCoord;

varying mat3 TBN;

uniform vec3 lightPos;

uniform vec3 viewPos;

uniform sampler2D diffuseTexture;

uniform sampler2D specularTexture;

uniform sampler2D normalMap;

uniform bool hasTexture;

void main() {

vec3 finalColor;

if(hasTexture){

vec3 textureNormal = texture2D(normalMap, fTexCoord).xyz;

textureNormal = 2.0 \* textureNormal - vec3(1.0, 1.0, 1.0);

vec3 adjustedNormal = normalize(TBN \* textureNormal);

vec3 ambientColor = vec3(0.2, 0.2, 0.2);

vec3 diffuse = texture2D(diffuseTexture, fTexCoord).rgb;

vec3 specular = texture2D(specularTexture, fTexCoord).rgb;

float shininess = 30.0;

vec3 lightDir = normalize(lightPos - fVertex);

float diff = max(dot(adjustedNormal, lightDir), 0.0);

vec3 diffuseColor = diffuse \* diff;

vec3 viewDir = normalize(viewPos - fVertex);

vec3 reflectDir = reflect(-lightDir, adjustedNormal);

float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);

vec3 specularColor = specular \* spec;

finalColor = ambientColor + diffuseColor + specularColor;

} else{

finalColor = vec3(0.0, 0.4, 1.0);

}

gl\_FragColor = vec4(finalColor, 1.0);

}`;

document.getElementById("angleSlider").addEventListener("input", function(event) {

let degrees = parseFloat(event.target.value);

rotationAngle = degrees \* Math.PI / 180;

draw();

});