МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра цифрових технологій в енергетиці

**Візуалізація графічної та геометричної інформації**

Розрахунково графічна робота

**Виконав:** Студент 5-го курсу, ІАТЕ,

гр. ТР-22мп

Шевчук Д.О.

**Перевірив:** Демчишин А.А.

**Київ-2022**

**Завдання**

1. Нанести текстуру на поверхню з лабораторної роботи №2.
2. Розробити програму для обертання текстури навколо точки що задає користувач.
3. Розробити можливість змінювати положення точки користувача вздовж u, v за допомогою клавіатури: клавіші A і D переміщують точку вздовж параметра u, а клавіші W і S переміщують точку вздовж параметра v.

4. Створити гілку CGW в репозиторії та завантаєити розроблену програму.

5. Створити звіт до розрахунково-графічної роботи та завантажити в гілку.

**Теоретична інформація**  
 WebGL бібліотека що використовується для візуалізації інтерактивної двох та трьох вимірної графіки у веб браузерах без використання додаткових HTML плагінів за допомогою звичайного canvas. WebGl підтримується основними сучасними браузерами, та для написання програм з ним використовується мова програмування JavaScript в основному для зв'язування коду спеціальних ефектів (фрагментного та вертексного шейдера) з канвасом та DOM. В свою чергу WebGl використовує GPU для виконання коду що написаний в вертексному та фрагментному шейдерах.  
 За допомогою бібліотеки WebGL можна досить легко та зручно взаємодіяти з відображення текстур на поверхні будь якої складності. Для цього потрібно зробити деякі маніпуляції з вершиним шейдером щоб він використовував надалі текстуру, а не кольори. Також у вершиному шейдері, якщо потрібно, можна змінювати координати текстури, масштабувати переносити та перетворювати їх за допомогою різних матриць. Після рефакторингу вершинного шейдера потрібно завантажити текстуру використовуючи JavaScript та зв'язати її з буфером координат текстури.   
 Як правило для роботи для роботи з текстурою що має 2 координати, а поверхня 3, як в нашому випадку, ми повинні перейти до координатної сітки UV.   
 Для накладання текстури у фрагментному шейдері ми використовуємо sampler та функцію texture2d.  
 Для обертання чи масштабування текстури навколо точки що задає користувач потрібно виконати наступні дії:  
1. Перенести точку на координати (0:0)  
2. Виконати обертання чи масштабування навколо даної точки.  
3. Повернути точку на свої координати  
 Використовуючи дану інформацію можна з легкістю виконати розрахунково графічну роботу.

**Реалізація**

Перед початком виконання розрахунково графічної роботи потрібно створити гілку в репозиторію на Github під назвою CGW у яку, надалі, будуть комітитися файли та частини коду, що буде потрібний для виконання завдання.

Після створення гілки потрібно розробити функції для накладання текстури в файлі main.js. Перш за все функція буде створювати WebGl об'єкт текстури та об'єкт картинки Image, після чого картинка буде завантажуватися з Інтернету за допомогою функції onLoad та привязуватися до об'єкту текстури. Функція має наступний вигляд:

function loadTexture() {

const texture = gl.createTexture();

gl.bindTexture(gl.TEXTURE\_2D, texture);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.LINEAR);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MAG\_FILTER, gl.LINEAR);

const image = new Image();

const imgURL = 'https://www.the3rdsequence.com/texturedb/download/9/texture/jpg/1024/brick+wall-1024x1024.jpg';

image.src = imgURL;

image.crossOrigin = 'anonymous';

image.onload = () => {

gl.bindTexture(gl.TEXTURE\_2D, texture);

gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, image);

draw();

};

}

Після чого нам потрібно створити буфер для координат текстури та зробити прив'язку цього буфера до його атрибуту:

// TextureCoords

const tBuffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY\_BUFFER, tBuffer);

gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(textCoords), gl.STREAM\_DRAW);

gl.enableVertexAttribArray(shProgram.iAttribTextCoords);

gl.vertexAttribPointer(shProgram.iAttribTextCoords, 2, gl.FLOAT, false, 0, 0);

Наступним кроком нам потрібно створити функцію для переміщення точки яку задає користувач, у Vertex шейдері, вона буде мати наступний вигляд:

mat4 getTranslate(vec2 t) {

return mat4(

1.0, 0.0, 0.0, t.x,

0.0, 1.0, 0.0, t.y,

0.0, 0.0, 1.0, 0.0,

0.0, 0.0, 0.0, 1.0

);

}

Після чого створюємо функцію для обертання текстури навколо точки що задає користувач, вона буде мати наступний вигляд:  
mat4 getRotate(float angleRad) {  
 float c = cos(angleRad);  
 float s = sin(angleRad);

return mat4(

vec4(c, s, 0.0, 0.0),

vec4(-s, c, 0.0, 0.0),

vec4(0.0, 0.0, 1.0, 0.0),

vec4(0.0, 0.0, 0.0, 1.0)

);}

**Результати розрахунково-графічної роботи**

На рис.1 наведено результати виконання РГР з нанесенням текстури.

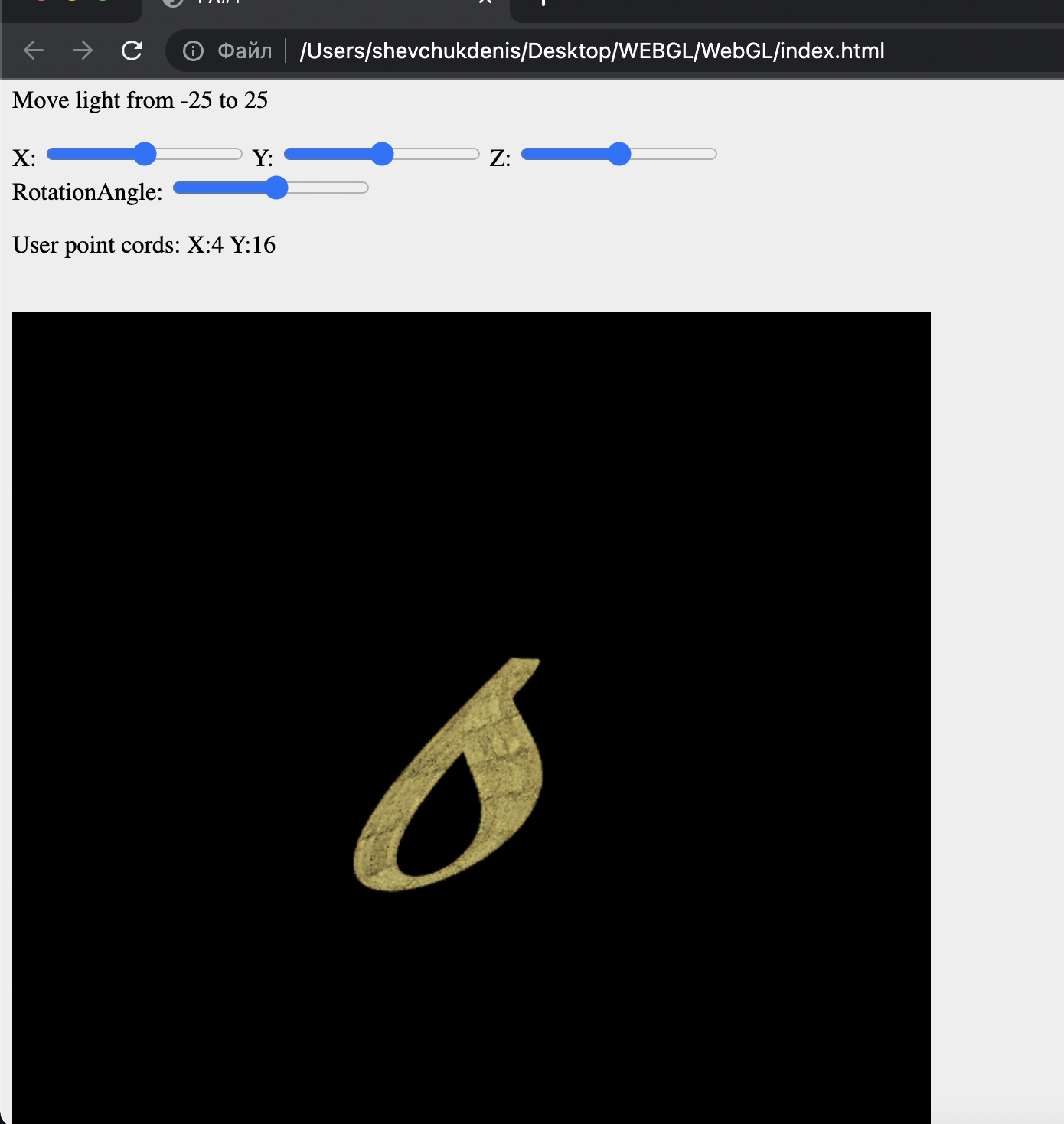


Рис. 1 --- Поверхня з нанесеною текстурою

На рис. 2 було змінено кут обертання фігури та положення точки користувача навколо якої відбувається обертання:

Т

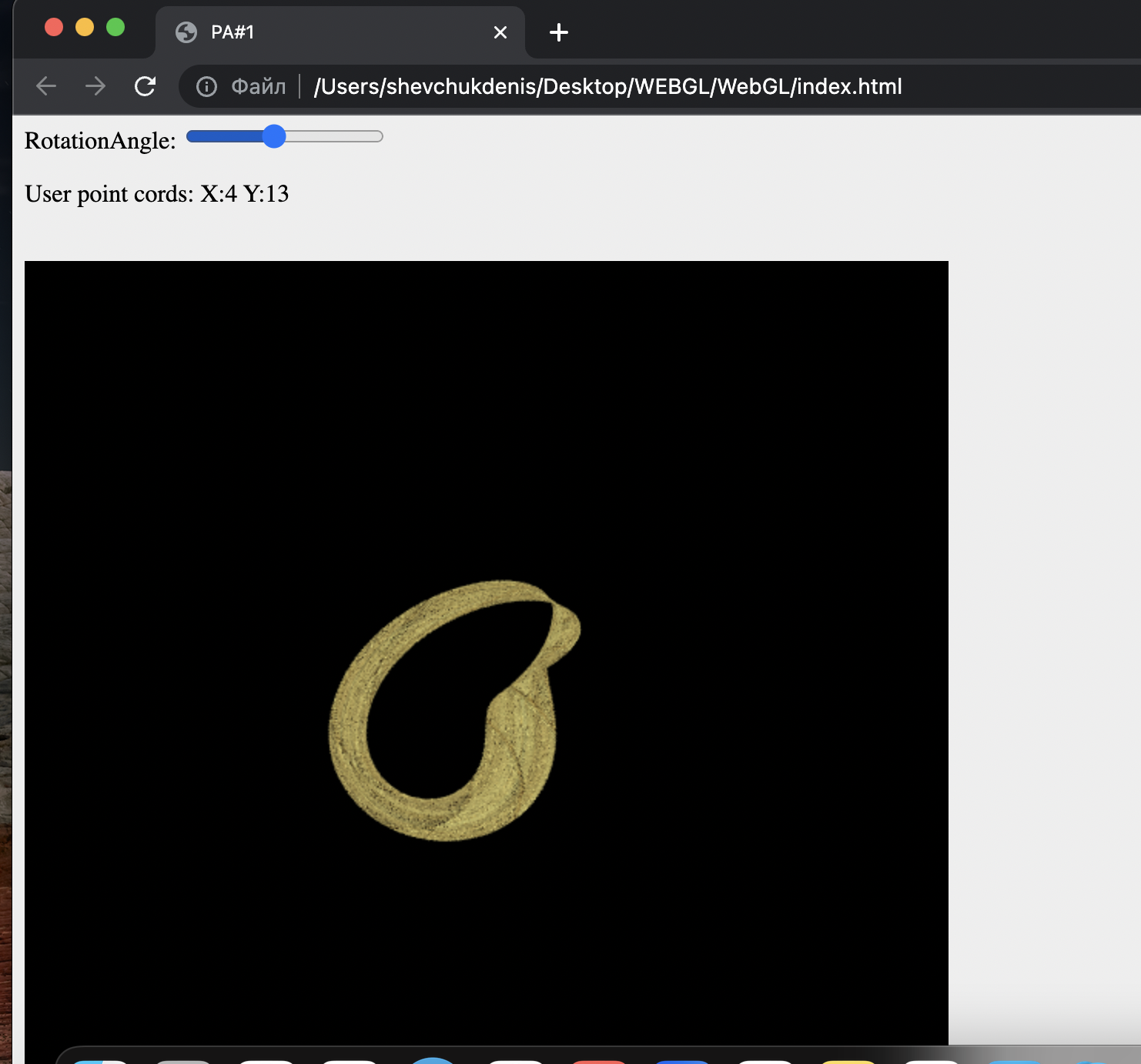


Рис. 2 --- поверхня зі зміненими параметрами

**Код**

// Vertex shader

const vertexShaderSource = `

attribute vec3 vertex;

attribute vec3 normal;

attribute vec2 textCoords;

uniform mat4 ModelViewProjectionMatrix, normalMatrix;

uniform float fAngleRad;

uniform vec2 fUserPoint;

varying vec3 normalInterp;

varying vec3 vertPos;

varying vec2 vTextCoords;

mat4 getRotate(float angleRad) {

float c = cos(angleRad);

float s = sin(angleRad);

return mat4(

vec4(c, s, 0.0, 0.0),

vec4(-s, c, 0.0, 0.0),

vec4(0.0, 0.0, 1.0, 0.0),

vec4(0.0, 0.0, 0.0, 1.0)

);

}

mat4 getTranslate(vec2 t) {

return mat4(

1.0, 0.0, 0.0, t.x,

0.0, 1.0, 0.0, t.y,

0.0, 0.0, 1.0, 0.0,

0.0, 0.0, 0.0, 1.0

);

}

void main(){

vec4 vertPos4 = ModelViewProjectionMatrix \* vec4(vertex, 1.0);

vertPos = vec3(vertPos4) / vertPos4.w;

normalInterp = vec3(normalMatrix \* vec4(normal, 0.0));

mat4 rotateMat = getRotate(fAngleRad);

mat4 translate = getTranslate(-fUserPoint);

mat4 translateBack = getTranslate(fUserPoint);

vec4 textCoordTr = vec4(textCoords, 0, 0) \* translate;

vec4 textCoordRotate = textCoordTr \* rotateMat;

vec4 textCoordTrBack = textCoordRotate \* translateBack;

vTextCoords = vec2(textCoordTrBack);

gl\_Position = vertPos4;

}`;

// Fragment shader

const fragmentShaderSource = `

#ifdef GL\_FRAGMENT\_PRECISION\_HIGH

precision highp float;

#else

precision mediump float;

#endif

uniform vec4 color;

varying vec3 normalInterp;

varying vec3 vertPos;

varying vec2 vTextCoords;

uniform float shininessVal;

uniform vec3 ambientColor;

uniform vec3 diffuseColor;

uniform vec3 specularColor;

uniform vec3 lightPos;

uniform sampler2D uTexture;

void main() {

vec3 N = normalize(normalInterp);

vec3 L = normalize(lightPos - vertPos);

float lambertian = max(dot(N, L), 0.0);

float specularL = 0.0;

if(lambertian > 0.0) {

vec3 R = reflect(-L, N);

vec3 V = normalize(-vertPos);

float specAngle = max(dot(R, V), 0.0);

specularL = pow(specAngle, shininessVal);

}

vec3 ambient = ambientColor;

vec3 diffuse = lambertian \* diffuseColor;

vec3 specular = specularL \* specularColor;

vec4 texture = texture2D(uTexture, vTextCoords);

vec4 color1 = vec4(ambient + diffuse + specular, 1.0);

gl\_FragColor = texture \* vec4(1.0, 1.0, 0.6, 1.0);

}`;