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, гр. ТР-22мп Мордас І.С

**Перевірив:** Демчишин А.А.

**Київ-2022**

**Завдання**

1. Нанести текстуру на поверхню з лабораторної роботи 2.
2. Реалізувати масштабування текстури (координати текстури), обертання навколо визначеної користувачем точки.
3. Реалізувати можливість переміщати точку вздовж простору поверхні (u, v) за допомогою клавіатури: наприклад клавіші A і D переміщують точку вздовж параметра u, а клавіші W і S переміщують точку вздовж параметра v.

4. Завантажити код в репозиторій на GitHub

5. Створити гілку CGW, в яку помістити розроблений код.

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

**Теоретична інформація**

WebGL це спеціальна бібліотека для мови програмування JavaScript, яка використовується для візуалізації трьох вимірної і двох вимірної графіки.

Заснований був на OpenGl, що дозволяє використовувати його API в елементах canvas HTML5. Також за рахунок використання низькорівневих засобів підтримки OpenGl, частина коду може виконуватися безпосередньо на відео картах.

Структура програми складається з різних компонентів, а саме код керування, написаний на JS і шейдерного коду. Тобто такого, який виконується на графічному процесорі. Це дозволяє зручно працювати з текстурами, різної складності.

За допомогою відображення текстур, тобто це техніка визначення кольору для кожного фрагменту можна легко взаємодіяти з відображеннями на поверхні.

Для роботи з текстурою, потрібно перейти до координатної сітки UV, адже текстура має 2 координати, а поверхня 3, тому і здійснюються такі перетворення.

В разі необхідності можна змінювати масштабування, координати текстури, переносити та перетворювати їх за допомогою різних матриць.

Етапи роботи:

1. Створення об’єкту текстури;
2. Створення елемента HTML, який є джерелом текстури наприклад Image;
3. Визначення методу onLoad;
4. Прив’язка текстури за допомогою gl.bindTexture()
5. Перетворення текстури для перетворення її в координатну систему об’єкта
6. Загрузка текстури в GPU
7. Установка параметрів текстури

**Реалізація**

Створимо гілку в репозиторії гітлаб гілку під назвою CGW відповідно до умови завдання. В подальшому всі зміни будуть додаватися в дану гілку.

Створимо функцію у файлі main.js за допомогою де буду створюватися об’єкти текстури та картинку. В подальшому картинка буде завантажуватися за допомогою onLoad з Інтернета. Розроблена функція:

const LoadTexture = () => {

const image = new Image();

image.src = 'https://www.the3rdsequence.com/texturedb/download/116/texture/jpg/1024/irregular+wood+planks-1024x1024.jpg';

image.crossOrigin = 'anonymous';

image.addEventListener('load', () => {

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);

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

});

}

Наступним кроком створюємо буфер, який потрібний для координат текстури і його прив’язка до атрибуту:

gl.bindBuffer(gl.ARRAY\_BUFFER, this.iTextureBuffer);

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

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

gl.enableVertexAttribArray(shProgram.iTextureCoords);

Далі додаємо функцію для переміщення точки, яку буде задавати користувач. Дану функцію додаємо в в шейдері.

mat4 getTranslateMat(vec2 point) {

return mat4(

vec4(1.0, 0.0, 0.0, point.x),

vec4(0.0, 1.0, 0.0, point.y),

vec4(0.0, 0.0, 1.0, 0.0),

vec4(0.0, 0.0, 0.0, 1.0)

);

}

Також необхідно створити функцію, за допомогою якої буде здійснюватися обертання текстури навколо точки. Метод Rotate:

mat4 getRotateMat(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.

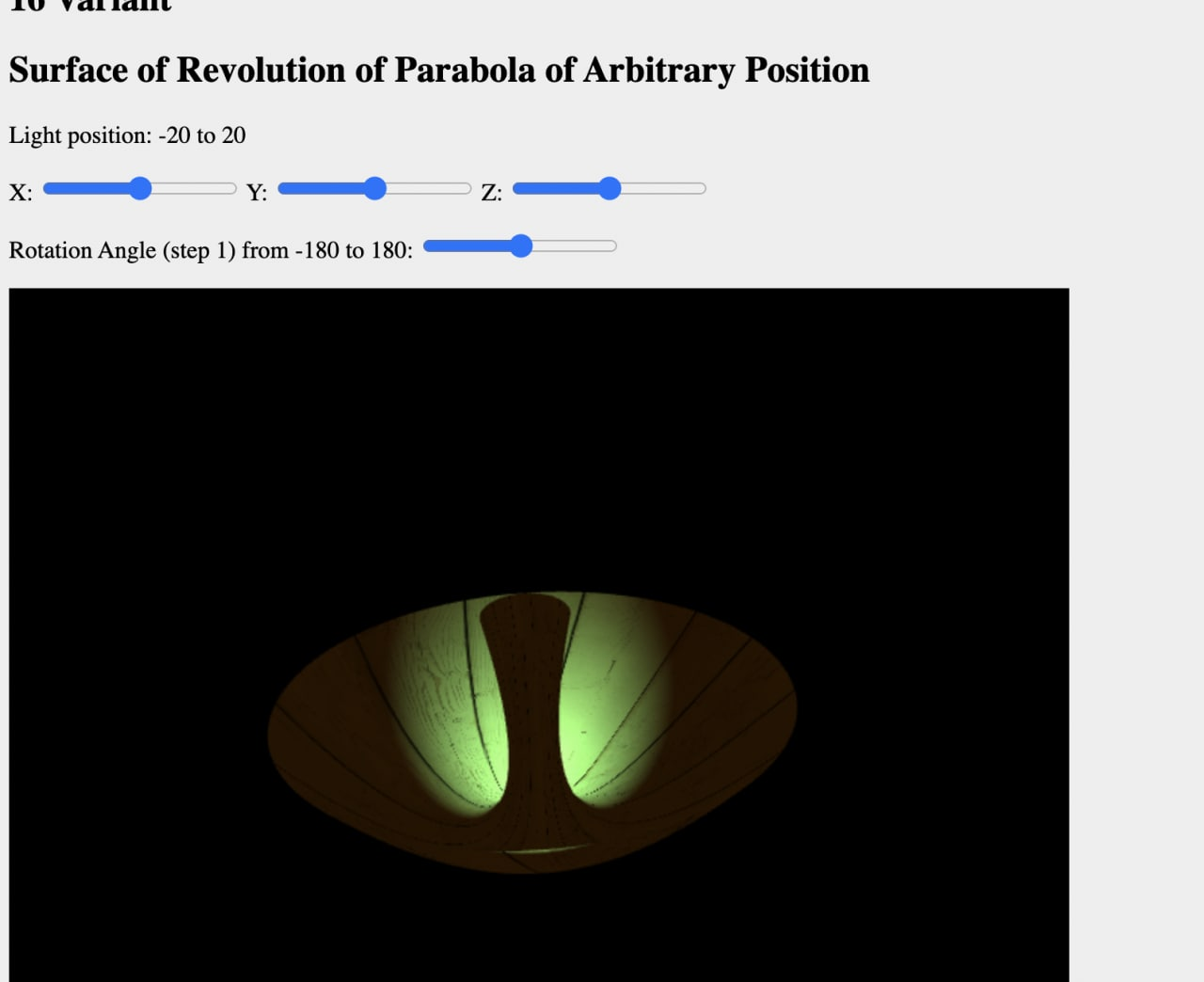


Рис. 1 --- Поверхня з текстурою

В ході виконання РГР було змінено кут обертання та відповідно положення точки навколо якої здійснюється обертання. На рис 2 результати виконання

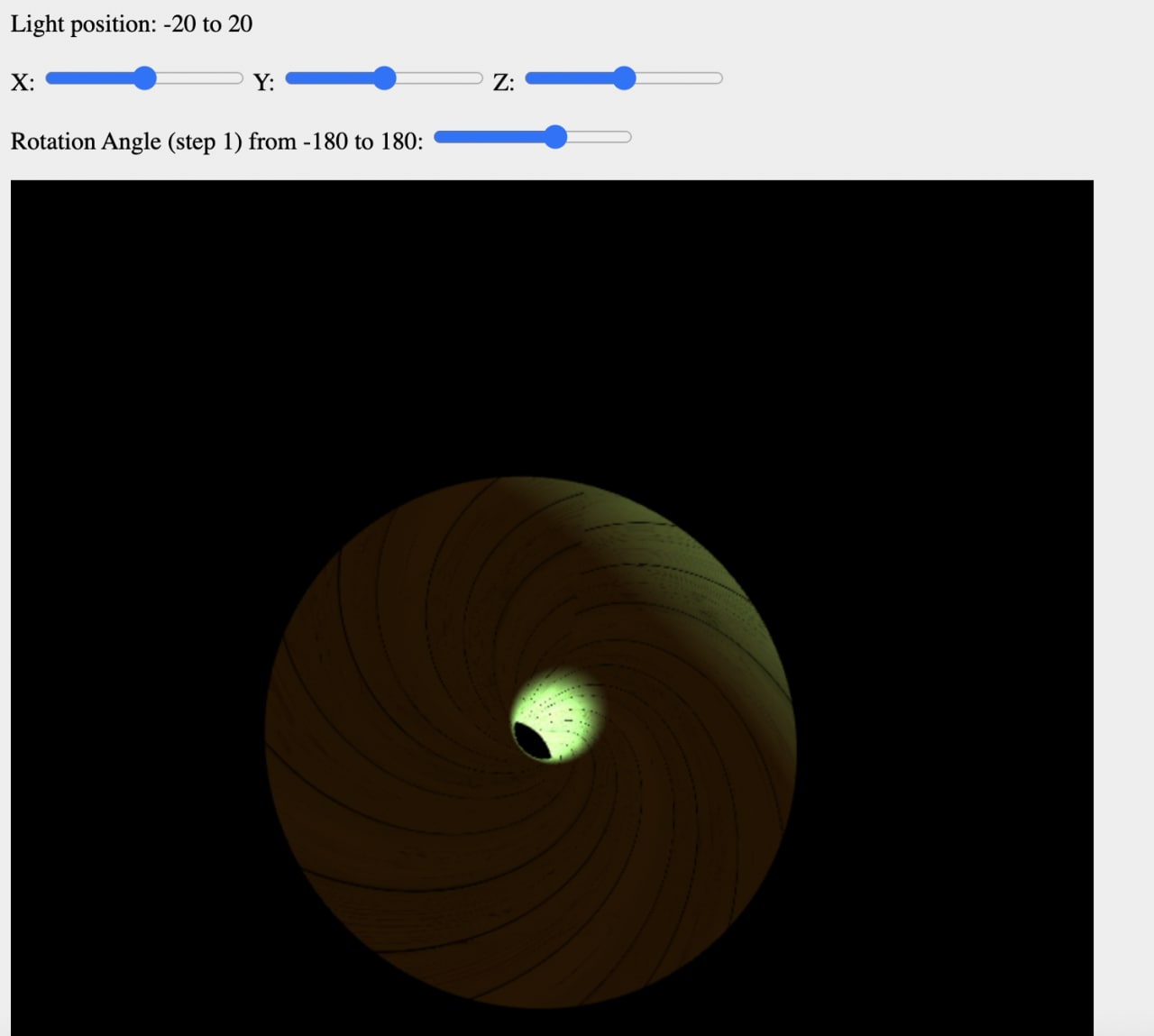
Т

Рис. 2 --- зміненими параметрами

**Код**

// Vertex shader

const vertexShaderSource = `

attribute vec3 vertex;

attribute vec3 normal;

attribute vec2 textureCoords;

uniform mat4 ModelViewProjectionMatrix, normalMat;

uniform float shininess;

uniform vec3 ambientColor;

uniform vec3 diffuseColor;

uniform vec3 specularColor;

uniform vec3 lightPosition;

// TEXTURE

uniform float fAngleRad;

uniform vec2 fUserPoint;

varying vec4 color;

varying vec2 vTextureCoords;

mat4 getRotateMat(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 getTranslateMat(vec2 point) {

return mat4(

vec4(1.0, 0.0, 0.0, point.x),

vec4(0.0, 1.0, 0.0, point.y),

vec4(0.0, 0.0, 1.0, 0.0),

vec4(0.0, 0.0, 0.0, 1.0)

);

}

void main(){

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

vec3 vertPos = vec3(vertPos4) / vertPos4.w;

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

gl\_Position = vertPos4;

vec3 normal = normalize(normalInterp);

vec3 lightDirection = normalize(lightPosition - vertPos);

float nDotLight = max(dot(normal, lightDirection), 0.0);

float specularLight = 0.0;

if (nDotLight > 0.0) {

vec3 viewDir = normalize(-vertPos);

vec3 halfDir = normalize(lightDirection + viewDir);

float specularAngle = max(dot(halfDir, normal), 0.0);

specularLight = pow(specularAngle, shininess);

}

vec3 diffuse = nDotLight \* diffuseColor;

vec3 ambient = ambientColor;

vec3 specular = specularLight \* specularColor;

mat4 rotatedMat = getRotateMat(fAngleRad);

mat4 translated = getTranslateMat(-fUserPoint);

mat4 translatedBack = getTranslateMat(fUserPoint);

vec4 tr = translated \* vec4(textureCoords, 0, 0);

vec4 rotated = tr \* rotatedMat;

vec4 trBack = rotated \* translatedBack;

vTextureCoords = vec2(trBack);

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

}`;

// Fragment shader

const fragmentShaderSource = `

#ifdef GL\_FRAGMENT\_PRECISION\_HIGH

precision highp float;

#else

precision mediump float;

#endif

varying vec4 color;

varying vec2 vTextureCoords;

uniform sampler2D tmu;

void main() {

vec4 texture = texture2D(tmu, vTextureCoords);

gl\_FragColor = texture \* color;

}`;