**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО"**

Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота з дисципліни

“Візуалізація графічної та геометричної інформації”

на тему “Операції з текстурними координатами”

Виконав студент групи ТР-32мп

Швагрун Андрій

Київ - 2024

1. **Завдання**

* Нанести текстуру на поверхню з практичного завдання №2.
* Реалізувати масштабування/обертання текстури (координати текстури) масштабування/обертання навколо визначеної користувачем точки – непарні варіанти реалізують масштабування, парні варіанти реалізують обертання.
* Розробити функціонал для переміщення точки 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.

1. **Теорія**

WebGL (Web Graphics Library) - це технологія, що надає можливість веб-розробникам використовувати 3D-графіку на веб-сторінках. Використання WebGL дозволяє наносити текстури на поверхні 3D-моделей, що відкриває широкі можливості для створення візуально привабливих та динамічних веб-додатків.

1. Нанесення текстури на поверхню

Нанесення текстур на поверхні реалізується за допомогою текстурних координат, які прив'язують певну точку текстури до конкретної точки на поверхні 3D-моделі. У WebGL це зазвичай використовується шляхом передачі координат текстури в вершинний та фрагментний шейдери.

// Вершинний шейдер

attribute vec2 a\_texCoord;

varying vec2 v\_texCoord;

void main() {

// Координати текстури передаються в фрагментний шейдер

v\_texCoord = a\_texCoord;

// Інші операції вершинного шейдера

}

// Фрагментний шейдер

precision mediump float;

varying vec2 v\_texCoord;

uniform sampler2D u\_texture;

void main() {

2. Масштабування та обертання текстури

Масштабування та обертання текстури реалізуються шляхом зміни координат текстури відносно визначеної точки. Для цього можна використовувати матриці трансформації.

// Масштабування текстури

mat3 scale(vec2 scale, vec2 pivot) {

mat3 m = mat3(1.0);

m[0][0] = scale.x;

m[1][1] = scale.y;

m[2][0] = (1.0 - scale.x) \* pivot.x;

m[2][1] = (1.0 - scale.y) \* pivot.y;

return m;

}

// Обертання текстури

mat3 rotate(float angle, vec2 pivot) {

mat3 m = mat3(1.0);

float c = cos(angle);

float s = sin(angle);

m[0][0] = c;

m[0][1] = -s;

m[1][0] = s;

m[1][1] = c;

m[2][0] = pivot.x \* (1.0 - c) + pivot.y \* s;

m[2][1] = pivot.y \* (1.0 - c) - pivot.x \* s;

return m;

}

3. Переміщення точки за допомогою клавіатури

Для переміщення точки вздовж параметрів u та v за допомогою клавіатури, потрібно обробляти введення клавіш і змінювати значення параметрів. Для цього можна використовувати обробник подій клавіш у JavaScript та змінювати координати точки.

// Приклад управління клавішами для переміщення точки

document.addEventListener('keydown', function(event) {

switch(event.key) {

case 'A':

// Зміна параметра u вліво

point.u -= step;

break;

case 'D':

// Зміна параметра u вправо

point.u += step;

break;

case 'W':

// Зміна параметра v вгору

point.v += step;

break;

case 'S':

// Зміна параметра v вниз

point.v -= step;

break;

}

});

Загальний підхід до розв'язання поставленого завдання полягає в поєднанні цих концепцій та методів для досягнення бажаного результату у WebGL. Розуміння основ цих тем дозволяє розробнику створювати більш складні та функціональні 3D-застосунки для веб-середовища.

1. **Деталі розробки**

У PA2 було візуалізовано модель, що представляє аналітичну поверхню за допомогою WebGL Virich Cuclic Surface.

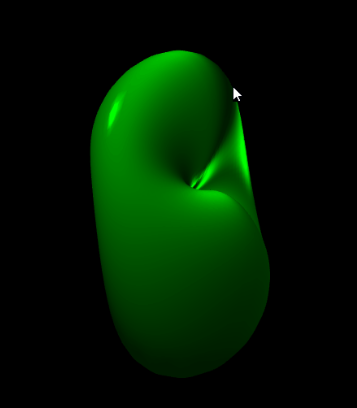


Рисунок 1 – Задана поверхня

Для виконання РГР в якості текстури було обрано зображення формату . jpg розміром 512х512 пікселів.



Рисунок 2 - Зображення текстури

Для накладання текстури на поверхню було підготовлено буфер текстурних координат, кожна з яких відповідає елементу масива з буферу вершин. В програмі шейдера для визначення кольору пікселя фігури на яку накладено текстуру необхідно використовувати функцію texture2D(). Ця функція першим аргументом приймає обʼєкт класу sampler2D, який зберігає дані про зображення, а другим текстурну координату.

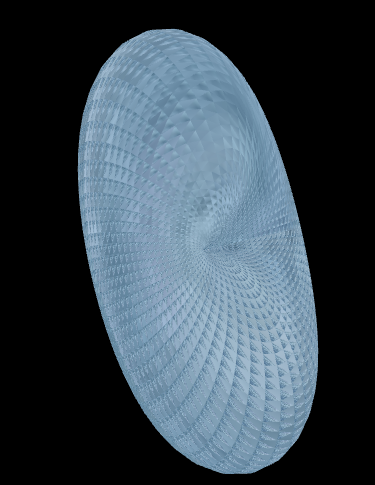


Рисунок 3 - Текстура накладена на поверхню

Після було створено новий обʼєкт класу Model для відображення точки (у вигляді сфери) відносно якої буде здійснюватись трансформація текстури. Точка має перебувати на поверхні фігури.

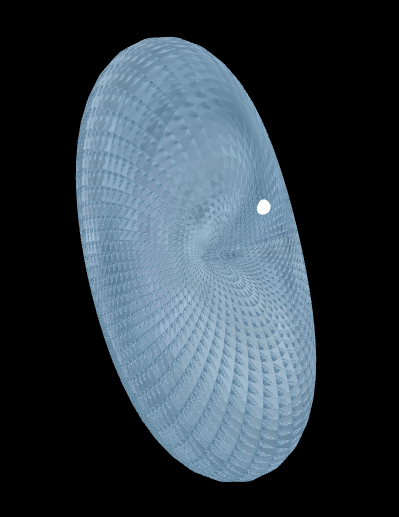


Рисунок 4 - Точка на текстурованій поверхні

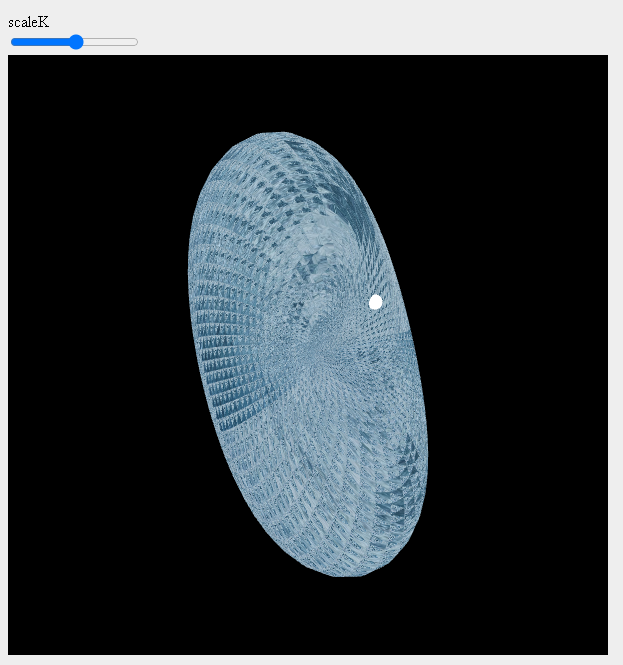
Згідно варіанту було реалізовано масштабування текстури відносно точки на поверхні.

Рисунок 5 – Демонстрація масштабування

1. **Інструкції користувача**

Фігуру можна обертати відносно центру за допомогою лівої клавіші миші або пальця на сенсорних екранах затиснувши в будь-якому місці полотна та потягнувши в сторону бажаного обертання.

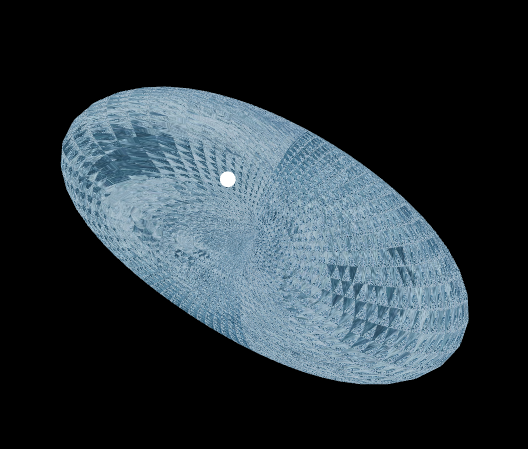
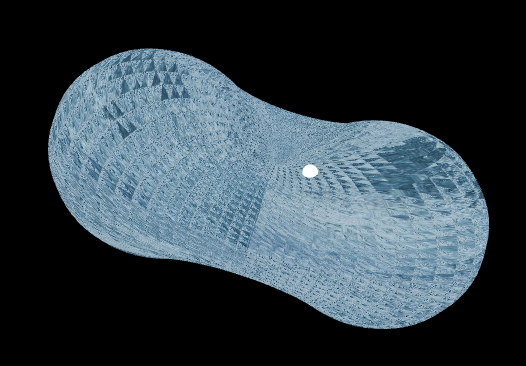


Рисунок 6 - Вигляд фігури до та після обертання

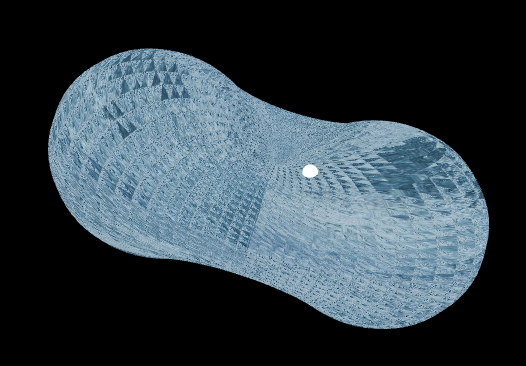
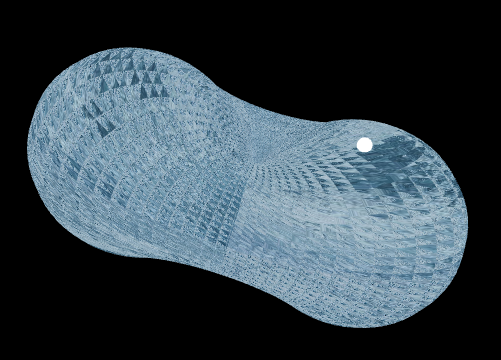
Переміщувати точку відносно якої здійснюється обертання можна за допомогою клавіш WASD. Кожне натискання переміщує точку по поверхні на визначений крок. Переміщення здійснюється до визначеної межі.

Рисунок 6 - Демонстрація переміщення точки відносно нерухомої фігури

За допомогою слайдера з підписом “scaleK” можна змінювати коефіцієнт масштабування відносно умовної точки на поверхні.



Рисунок 7 - Демонстрація зміни коефіцієнту масштабування текстури

При перезавантаженні сторінки обертання фігури, а також значення коефіцієнту масштабування буде встановлено по замовчуванню. Положення точки відносно поверхні також буде скинуто до значення по замовчуванню.

1. **Код програми**

Створення буферу для текстурних вершин

function Model(name) {  
 this.name = name;  
 this.iVertexBuffer = *gl*.createBuffer();  
 this.iVertexTextureBuffer = *gl*.createBuffer();  
 this.count = 0;  
  
 this.BufferData = function(vertices, verticesTexture) {  
  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, this.iVertexBuffer);  
 *gl*.bufferData(*gl*.*ARRAY\_BUFFER*, new *Float32Array*(vertices), *gl*.*STREAM\_DRAW*);  
  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, this.iVertexTextureBuffer);  
 *gl*.bufferData(*gl*.*ARRAY\_BUFFER*, new *Float32Array*(verticesTexture), *gl*.*STREAM\_DRAW*);  
  
 this.count = vertices.length/3;  
 }  
  
 this.Draw = function() {  
  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, this.iVertexBuffer);  
 *gl*.vertexAttribPointer(*shProgram*.iAttribVertex, 3, *gl*.*FLOAT*, false, 0, 0);  
 *gl*.enableVertexAttribArray(*shProgram*.iAttribVertex);  
  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, this.iVertexTextureBuffer);  
 *gl*.vertexAttribPointer(*shProgram*.iAttribVertexTexture, 2, *gl*.*FLOAT*, false, 0, 0);  
 *gl*.enableVertexAttribArray(*shProgram*.iAttribVertexTexture);  
  
 *gl*.drawArrays(*gl*.*TRIANGLES*, 0, this.count);  
 }  
  
 this.DrawSphere = function () {  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, this.iVertexBuffer);  
 *gl*.vertexAttribPointer(*shProgram*.iAttribVertex, 3, *gl*.*FLOAT*, false, 0, 0);  
 *gl*.enableVertexAttribArray(*shProgram*.iAttribVertex);  
 *gl*.drawArrays(*gl*.*TRIANGLE\_STRIP*, 0, this.count);  
 }  
}

Завантаження текстури

function LoadTexture() {  
 let 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();  
 image.crossOrigin = 'anonymus';  
  
 image.src = "texture.jpg";  
 image.onload = () => {  
 *gl*.bindTexture(*gl*.*TEXTURE\_2D*, texture);  
 *gl*.texImage2D(  
 *gl*.*TEXTURE\_2D*,  
 0,  
 *gl*.*RGBA*,  
 *gl*.*RGBA*,  
 *gl*.*UNSIGNED\_BYTE*,  
 image  
 );  
 draw()  
 }  
}

Керування положенням сфери

*window*.onkeydown = (e) => {  
 if (e.keyCode == 87) {  
 *userPoint*[0] = *Math*.min(*userPoint*[0] + 0.1, *Math*.PI \* 2);  
 }  
 else if (e.keyCode == 83) {  
 *userPoint*[0] = *Math*.max(*userPoint*[0] - 0.1, 0);  
 }  
 else if (e.keyCode == 68) {  
 *userPoint*[1] = *Math*.min(*userPoint*[1] + 0.1, 2 \* *Math*.PI);  
 }  
 else if (e.keyCode == 65) {  
 *userPoint*[1] = *Math*.max(*userPoint*[1] - 0.1, 0);  
 }  
 redrawSphere()  
}

Код шейдеру

// Vertex shader  
const vertexShaderSource = `  
attribute vec3 vertex;  
attribute vec2 texCoord;  
uniform mat4 ModelViewProjectionMatrix;  
  
varying vec2 texCoordV;  
uniform vec3 userPoint;  
uniform vec3 translateSphere;  
  
uniform float scaleK;  
uniform float b;  
  
mat4 translation(vec3 t) {  
 mat4 dst;  
  
 dst[0][0] = 1.0;  
 dst[0][ 1] = 0.0;  
 dst[0][ 2] = 0.0;  
 dst[0][ 3] = 0.0;  
 dst[1][ 0] = 0.0;  
 dst[1][ 1] = 1.0;  
 dst[1][ 2] = 0.0;  
 dst[1][ 3] = 0.0;  
 dst[2][ 0] = 0.0;  
 dst[2][ 1] = 0.0;  
 dst[2][ 2] = 1.0;  
 dst[2][ 3] = 0.0;  
 dst[3][ 0] = t.x;  
 dst[3][ 1] = t.y;  
 dst[3][ 2] = t.z;  
 dst[3][ 3] = 1.0;  
  
 return dst;  
}  
  
mat4 scaling(float s){  
 mat4 dst;  
  
 dst[0][0] = s;  
 dst[0][ 1] = 0.0;  
 dst[0][ 2] = 0.0;  
 dst[0][ 3] = 0.0;  
 dst[1][ 0] = 0.0;  
 dst[1][ 1] = s;  
 dst[1][ 2] = 0.0;  
 dst[1][ 3] = 0.0;  
 dst[2][ 0] = 0.0;  
 dst[2][ 1] = 0.0;  
 dst[2][ 2] = s;  
 dst[2][ 3] = 0.0;  
 dst[3][ 0] = 0.0;  
 dst[3][ 1] = 0.0;  
 dst[3][ 2] = 0.0;  
 dst[3][ 3] = 1.0;  
  
 return dst;  
}  
  
void main() {  
 vec4 tex1 = vec4(texCoord,0.,1.) \* translation(userPoint);  
 vec4 tex2 = tex1 \* scaling(scaleK);  
 vec4 tex3 = tex2 \* translation(-userPoint);  
  
 texCoordV=tex3.xy;  
 vec4 vertPos4 = ModelViewProjectionMatrix \* vec4(vertex, 1.0);  
 vec3 vertPos = vec3(vertPos4) / vertPos4.w;  
  
 gl\_Position = ModelViewProjectionMatrix \* vec4(vertex,1.0);  
 if(b>0.0){  
 vec4 sphere = translation(translateSphere)\*vec4(vertex,1.0);  
 gl\_Position=ModelViewProjectionMatrix\*sphere;  
 }  
}`;  
  
  
// Fragment shader  
const fragmentShaderSource = `  
#ifdef GL\_FRAGMENT\_PRECISION\_HIGH  
 precision highp float;  
#else  
 precision mediump float;  
#endif  
  
uniform sampler2D tmu;  
uniform float b;  
varying vec2 texCoordV;  
void main() {  
 gl\_FragColor = texture2D(tmu, texCoordV);  
 if(b>0.){  
 gl\_FragColor = vec4(1);  
 }  
}`;