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

групи ТР-21мп

Сосновенко В.В.

**Перевірив:** Демчишин А.А.

**Київ-2022**

**Тема роботи:** Операціїї над тектурними координатами

**Завдання**:

* Накласти текступу на поверхню отриману в результаті виконання лабораторної роботи №2.
* Імплементувати масштабування або обертання текстури(текстурних координат) згідно з варіантом: непарні - масштабування, парні - обертання.
* Запровадити можливість переміщення точки відносно якої відбувається трансформація текстури по поверхні за рахунок зміни параметрів в просторі текстури. Наприклад, клавіші A та D для переміщення по осі абсцис, змінюючи параметр u текстури, а клавіші W та S по осі ординат, змінюючи параметр v.
* Створити гілку CGW в репозиторії та завантажити розроблену програму.
* Створити звіт до розрахунково-графічної роботи та завантажити в гілку.

**Теоретичні відомості**

Текстурування є дуже важливою частиною процесу 3D-моделювання. Усі дрібні візуальні характеристики у 3D-моделюванні, такі як зморшки чи окремі нитки килима, є продуктом текстури, нанесеної 3D-художником. Зазвичай створювані 3D-моделі мають стандартний сірий колір програми. Щоб додати кольори, малюнки та текстури, 2D-фотографії потрібно розмістити на 3D-моделях. Додавання кольорів або властивостей поверхні та матеріалу до 3D-моделі вимагає ще одного кроку вперед у процесі 3D-моделювання, тобто 3D-текстурування. Цей підхід часто призводить до повного кольору та властивостей поверхні 3D-моделі.

Стандартна процедура текстурування така:

UV Mapping and Unwrapping

Щоб почати процес 3D-текстурування, необхідно спочатку розгорнути модель, що, по суті, те саме, що розгортання 3D-сітки. Коли художники-фактуристи отримають готові моделі від відділу 3D-моделювання, вони створять UV-карту для кожного 3D-об’єкта. UV-карта — це плоске зображення поверхні 3D-моделі, яке використовується для швидкого накладання текстур. Прямо пов’язуючи 2D-зображення (текстуру) з вершинами багатокутника, UV-відображення може допомогти обернути 2D-зображення (текстуру) навколо 3D-об’єкта, а згенеровану карту можна використовувати безпосередньо в процесі текстурування та затінення.

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.

**Хід роботи**

В ході другої лабораторної роботи було створено поверню під назвою

«Двічі косий трохоїд циліндроїд». Отриману поверхню з освітленням можна побачити на рисунку 3.1.

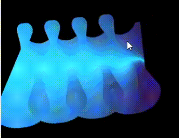


Рис. 3.1 «Двічі косий трохоїд циліндроїд» з освітленням

Для текстури було обрано картинку з інтернету формату «jpg». Після чого завантажено на github, щоб в подальшому використовувати посилання на неї.

З метою накладання текстури на поверхню було створено декілька змінних в коді шейдера. Після чого були створенні посилання на них в коді програми. Були також створені функції для генерації буфера даних текстури.

Обрану картинку можна побачити на рисунку 3.2.



Рис. 3.2 Обрана текстура

Для відображення умовної точки відносно якої буде виконуватися трансформація текстури, в класі моделі було сворено відповідний метод. Замість відображення точки було прийнято рішення відобжати сферу. Для відорбраження сфери було реалізовано функцію, яка створює геометрію для неї. Поверхню з умовною точкою зображено на рисунках 3.4.

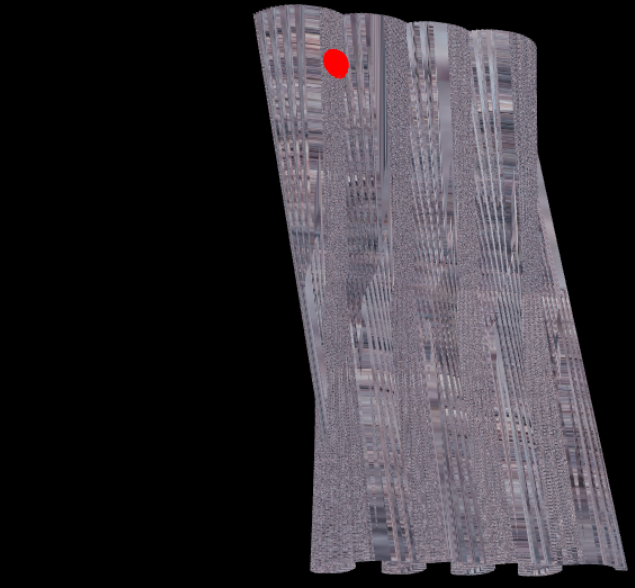
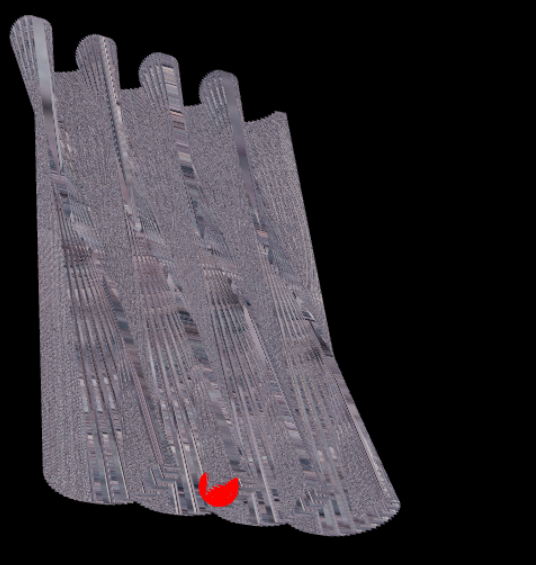


Рис. 3.4 Поверхня з умовною точкою

Для роботи з текстурою було створено ще кілька змінних в коді шейдера: обертання текстури, розташування умовної точки в (u,v) координатах, змінну для розташування сфери на відповідне місце поверхні в 3д-просторі.

Для реалізації переміщення точки по поверхні та обертання текстури було додано відповідні функції на відповідні вхідні дані від користувача.

Переміщення умовної точки реалізовано за допомогою введення з клавіатури: клавіші W та S здійснюють переміщення точки за параметром v в додатньому та від’ємному напрамках відповідно, клавіші A та D здійснюють переміщення точки за параметром u у від’ємному та додатньому напрямках відповідно. Приклади переміщення точки зображено на рисунках 3.5-3.6

 A close-up of a planet

Description automatically generated with low confidence

Рис. 3.5 Рис. 3.6

**Код**

// Vertex shader

const vertexShaderSource = `

attribute vec3 vertex;

attribute vec3 normal;

uniform mat4 ModelViewProjectionMatrix, normalMat;

varying vec3 normalInterp;

varying vec3 vertPos;

attribute vec2 texture;

varying vec2 v\_texcoord;

uniform vec3 translateUP;

uniform vec2 userPoint;

uniform float rotA;

mat4 translation(float tx, float ty, float tz) {

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] = tx;

dst[3][ 1] = ty;

dst[3][ 2] = tz;

dst[3][ 3] = 1.0;

return dst;

}

mat4 rotation(float angleInRadians) {

mat4 dst;

float c = cos(angleInRadians);

float s = sin(angleInRadians);

dst[0][0] = c;

dst[0][ 1] = s;

dst[0][ 2] = 0.0;

dst[0][ 3] = 0.0;

dst[1][ 0] = -s;

dst[1][ 1] = c;

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] = 0.0;

dst[3][ 1] = 0.0;

dst[3][ 2] = 0.0;

dst[3][ 3] = 1.0;

return dst;

}

void main() {

mat4 rotMatrix = rotation(rotA);

mat4 transMatrix1 = translation(userPoint.x,userPoint.y,0.0);

mat4 transMatrix2 = translation(-userPoint.x,-userPoint.y,0.0);

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

vec4 texture1 = vec4(texture,0.0,0.0)\*transMatrix1;

vec4 textureRotated = texture1\*rotMatrix;

vec4 texture2 = textureRotated\*transMatrix2;

v\_texcoord = vec2(texture2.x,texture2.y);

vertPos = vec3(vertPos4) / vertPos4.w;

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

gl\_Position = vertPos4;

if(rotA>1000.0){

vec4 translatePoint = translation(translateUP.x,translateUP.y,translateUP.z)\*vec4(vertex,1.0);

gl\_Position = ModelViewProjectionMatrix\*translatePoint;

}

}`;

// Fragment shader

const fragmentShaderSource = `

#ifdef GL\_FRAGMENT\_PRECISION\_HIGH

precision highp float;

#else

precision mediump float;

#endif

varying vec2 v\_texcoord;

uniform sampler2D tmu;

uniform float rotA;

varying vec4 color;

precision mediump float;

varying vec3 normalInterp; // Surface normal

varying vec3 vertPos; // Vertex position

uniform float ambientCoefficient; // Ambient reflection coefficient

uniform float diffuseCoefficient; // Diffuse reflection coefficient

uniform float specularCoefficient; // Specular reflection coefficient

uniform float shininess; // Shininess

uniform vec3 ambientColor;

uniform vec3 diffuseColor;

uniform vec3 specularColor;

uniform vec3 lightPosition;

void main() {

vec3 vNormal = normalize(normalInterp);

vec3 light = normalize(lightPosition);

float dotProduct = max(dot(vNormal, light), 0.0);

float specular = 0.0;

if(dotProduct > 0.0) {

vec3 reflect = reflect(-light, vNormal);

vec3 n\_vertPos = normalize(-vertPos);

float specAngle = max(dot(reflect, n\_vertPos), 0.0);

specular = pow(specAngle, shininess);

}

gl\_FragColor = vec4(ambientCoefficient \* ambientColor

+ diffuseCoefficient \* dotProduct \* diffuseColor

+ specularCoefficient \* specular \* specularColor, 1.0);

vec4 texColor = texture2D(tmu, v\_texcoord);

gl\_FragColor = texColor;

if(rotA>1000.0){

gl\_FragColor = vec4(0.,0.,0.,0.);

}

}`;