Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота

З дисципліни «Візуалізація графічної та геометричної інформації»

Варіант 4

Виконав: Узун А.С.

Студент групи ТР-21мп

Київ 2022

**Завдання**

* Накласти текстуру 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.
* Підготувати звіт

Теоретичні відомості

WebGL — це кросплатформний низькорівневий графічний API, який дозволяє розробникам створювати інтерактивну 2D і 3D графіку у веб-переглядачах. Однією з ключових особливостей WebGL є можливість використовувати GLSL (OpenGL Shading Language) для написання власних шейдерів, які можна запускати на графічному процесорі (графічному процесорі). Одним із поширених застосувань GLSL у WebGL є виконання матричних перетворень 2D та 3D об’єктів. Матричні перетворення використовуються для переміщення, обертання, масштабування та нахилу об’єктів у 3D-сцені. Ці перетворення можна комбінувати та об’єднувати для створення більш складних перетворень. Відображення текстури – це техніка, яка часто використовується в WebGL для додання реалізму 3D-моделям. Текстури — це зображення, які наносяться на поверхню тривимірного об’єкта, щоб надати йому певного вигляду, наприклад текстуру дерева або каменю. У WebGL текстури зазвичай завантажуються в графічний процесор і доступ до них здійснюється за допомогою UV (текстурних) координат. UV-координати використовуються для визначення положення точки на двовимірному зображенні текстури по відношенню до тривимірного об’єкта, до якого вона застосована. UV-координати використовуються для інтерполяції між пікселями в зображенні текстури, що дозволяє плавно застосовувати текстуру до 3D-об’єкта. Використовуючи GLSL, можна маніпулювати UV-координатами текстури для досягнення різних ефектів, таких як масштабування, обертання або зміщення текстури. Це може бути корисним для створення анімації або додавання деталей до 3D-моделей. Таким чином, WebGL і GLSL надають потужний набір інструментів для створення інтерактивної графіки та анімації у веб-браузерах. Матричні перетворення та відображення текстури є двома важливими техніками, які можна використовувати для додання реалізму та деталізації 3D-моделей. Використовуючи графічний процесор для виконання цих операцій, можна досягти швидкого та ефективного рендерингу складної графіки в реальному часі.

Нанесення текстури – це техніка, яка зазвичай використовується в WebGL для додання реалізму та деталізації 3D-моделей. Текстура — це зображення, яке наноситься на поверхню тривимірного об’єкта, щоб надати йому певного вигляду, наприклад текстуру дерева або каменю. У WebGL текстури зазвичай завантажуються в графічний процесор і доступ до них здійснюється за допомогою UV (текстурних) координат. UV-координати використовуються для визначення положення точки на двовимірному зображенні текстури по відношенню до тривимірного об’єкта, до якого вона застосована. UV-координати використовуються для інтерполяції між пікселями в зображенні текстури, що дозволяє плавно застосовувати текстуру до 3D-об’єкта. Маніпулюючи UV-координатами, можна досягти різних ефектів, таких як масштабування, обертання або зміщення текстури. Це може бути корисним для створення анімації або додавання деталей до 3D-моделей. Крім того, кілька текстур можна комбінувати за допомогою техніки змішування для створення більш складних зовнішніх зображень.

Виконання завдання

В ході другої лабораторної роботи було створено поверхню. Отриману поверхню з освітленням можна побачити на рисунку 1.

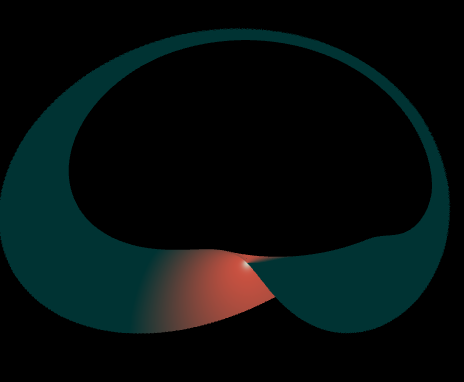


Рис.1 Отримана поверхня з освітленням

Для текстури було обрано картинку з інтернету. Після чого

завантажив її на github, щоб в подальшому використовувати посилання на неї.

Обрану картинку можна побачити на рисунку 2.

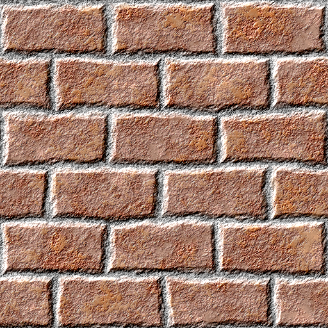


Рис. 2 Обрана текстура

Для відображення умовної точки відносно якої буде виконватися

трансформація текстури, в класі моделі було сворено відповідну функцію.

Замість відображення точки було прийнято рішення відобжати сферу,

адже працюємо в 3д-просторі. Для відорбраження сфери необхідно було

створити функцію, яка б створювала геометрію для неї. Модель з

умовною точкою зображено на рисунку 3.

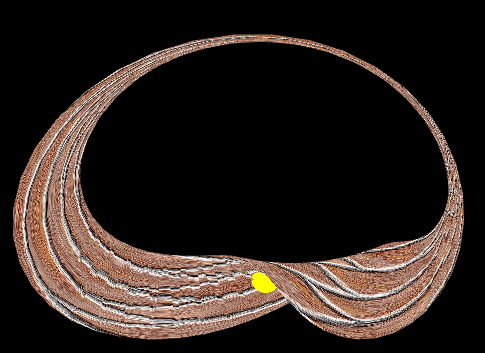


Рис. 3 Поверхня з умовною точкою

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.

Вказівки користувачу

Переміщення умовної точки реалізовано за допомогою введення з клавіатури(рисунок 4): клавіші W та S здійснюють переміщення точки за параметром v в додатному та від’ємному напрямках відповідно, клавіші A та D здійснюють переміщення точки за параметром u у від’ємному та додатному напрямках відповідно.

Орієнтація поверхні в просторі здійснюється за допомогою введення з миші, так само як і в минулих роботах. Обертання текстури залежить від переміщення миші по горизонталі.

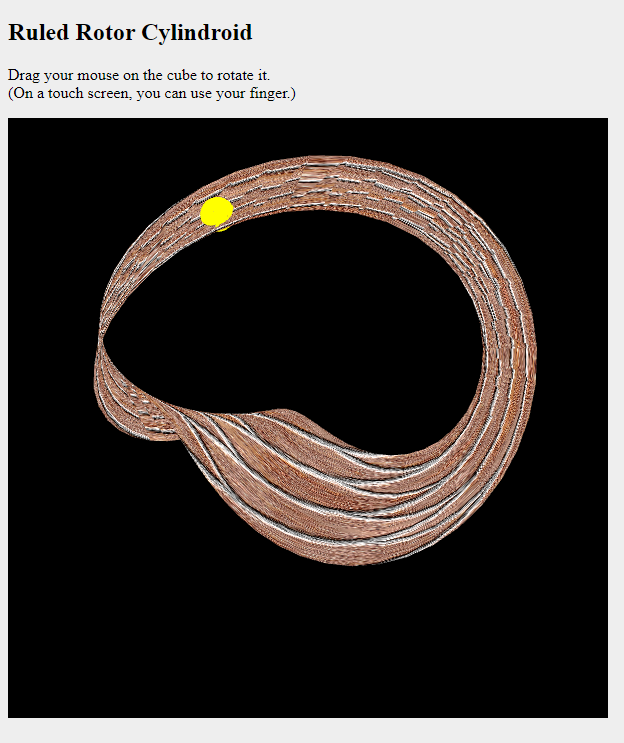


Рис. 4 Приклад роботи

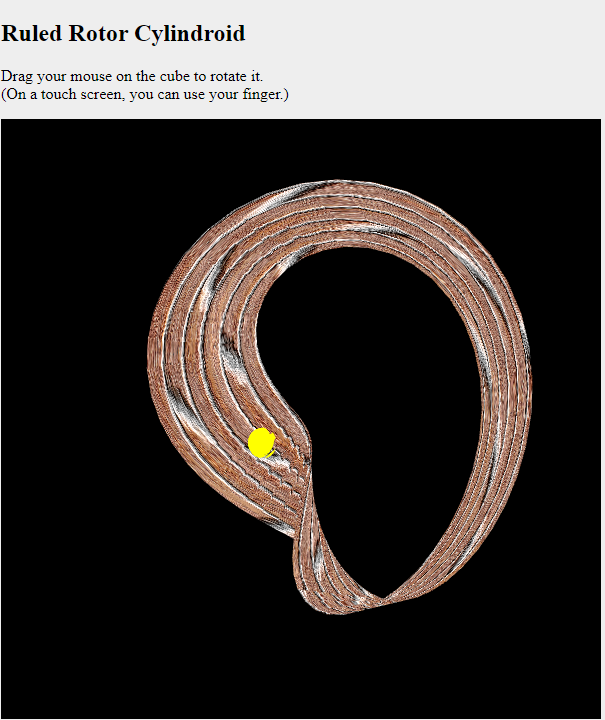


Рис. 5 Приклад роботи

Код

// Vertex shader

const vertexShaderSource = `

attribute vec3 vertex;

attribute vec2 texture;

uniform mat4 ModelViewProjectionMatrix;

varying vec2 v\_texcoord;

uniform vec3 translatePoint;

uniform vec2 texturePoint;

uniform float rotateValue;

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 rMatrix = rotation(rotateValue);

mat4 t1Matrix = translation(texturePoint.x,texturePoint.y,0.0);

mat4 t2Matrix = translation(-texturePoint.x,-texturePoint.y,0.0);

vec4 texture1 = vec4(texture,0.0,0.0)\*t1Matrix;

vec4 textureRotated = texture1\*rMatrix;

vec4 texture2 = textureRotated\*t2Matrix;

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

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

gl\_Position = vertPos4;

if(rotateValue>10.0){

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

gl\_Position = ModelViewProjectionMatrix\*tPoint;

}

}`;

// 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 rotateValue;

void main() {

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

gl\_FragColor = texColor;

if(rotateValue>10.0){

gl\_FragColor = vec4(0.,1.,0.,0.);

}

}`;