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 6

Виконав: Грицюк А.О

Студент групи ТР-21мп

Київ 2022

**Завдання**

**Тема роботи:** Операції над тектурними координатами

**Вимоги:**

1. Накласти текстуру на поверхню отриману в результаті виконання лабораторної роботи №2.
2. Імплементувати масштабування або обертання текстури(текстурних координат) згідно з варіантом: непарні - масштабування, парні - обертання.
3. 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.

**Теоритичні відомості**

**Текстурування** є важливим кроком у створенні і візуалізації 3D-моделі, що дозволяє надати поверхні об'ємного об'єкта фактурності, рельєфності і потрібного колірного забарвлення для додання моделі максимальної реалістичності. 3D-моделювання — це процес розробки математичного представлення будь-якої тривимірної поверхні об'єкта за допомогою спеціалізованого ПЗ. Продукт моделювання є 3D-модель. Вона може бути представлена у вигляді програмного коду або відображена у вюпорті чи вювері, як 3D-модель, а також за допомогою двовимірного зображення, що створюється за допомогою процесу рендерингу. 3D-моделі можуть створюватись вручну або автоматично, у тому числі за допомогою 3D-сканера. Виготовлення моделей вручну є подібним до створення скульптури в пластичному мистецтві. Тривимірна (3D) графіка - комп'ютерна графіка для зображення об'ємних об'єктів. Найбільше застосовується для створення зображень на площині екрану або аркуша друкованої продукції в архітектурній візуалізації, кінематографі, телебаченні, комп'ютерних іграх, друкованої продукції, а також в науці і промисловості.

Сучасне програмне забезпечення дозволяє використовувати незалежно від алгоритму моделювання різноманітні підходи для побудови моделі.

Примітиви — моделювання за допомогою простих геометричних фігур (кулі, циліндри, конуси тощо), які використовуються як цеглинки при побудові складніших об'єктів. Перевагою методу є швидка та легка побудова, а також те що моделі є математично визначені і точні. Підходить до технічного моделювання і менше для моделювання органіки. Деякі програми можуть рендерити з примітивів напряму, інші використовують примітиви тільки для моделювання, а пізніше конвертують для подальшої роботи або рендерингу.

3D-графіка призначена для імітації фотографування або відео зйомки тривимірних образів об’єктів, які попередньо створюються в пам’яті комп’ютера в такій послідовності: попередня підготовка, створення геометричної моделі сцени, настроювання освітлення і знімальних камер, підготовка і призначення матеріалів, візуалізація сцени. Таким чином створюється уявний світ, який часто називають віртуальним.

Попередня підготовка передбачає продумування складу сцени, розміщення об’єктів і їх деталей, які будуть видимими з передбачуваних напрямів спостереження.

На етапі створення геометричної моделі сцени за допомогою різноманітних інструментальних засобів будуються тривимірні геометричні моделі об’єктів сцени, після чого сцену можна розглядати і “фотографувати” з будь-якого потрібного ракурсу.

Правильний вибір джерел світла дозволяє виконувати імітацію фотографування сцени в будь-яких умовах освітленості, причому освітленість всіх об’єктів, тіні від них і бліки світла розраховуються автоматично. Моделі знімальних камер дають можливість розглядати тривимірну сцену і виконувати її знімання під будь-яким вибраним кутом зору.

На етапі підготовки і призначення матеріалів забезпечується надання сцені візуальної правдоподібності, що наближує якість зображення до реальної фотографії. Працюючи з матеріалами, можна настроювати такі їх якості, як сила блиску, прозорість, самосвічення, дзеркальність, рельєфність та інші. Реальні фотографії можна включати в склад матеріалів або використовувати для імітації фону.

Візуалізація сцени або рендерінг (rendering) полягає в проведенні програмою розрахунків і нанесення на зображення всіх тіней, бліків, взаємних відблисків об’єктів і т. п. і може тривати досить довго, що залежить від складності сцени і швидкодії комп’ютера.

До недоліків тривимірної графіки слід віднести:

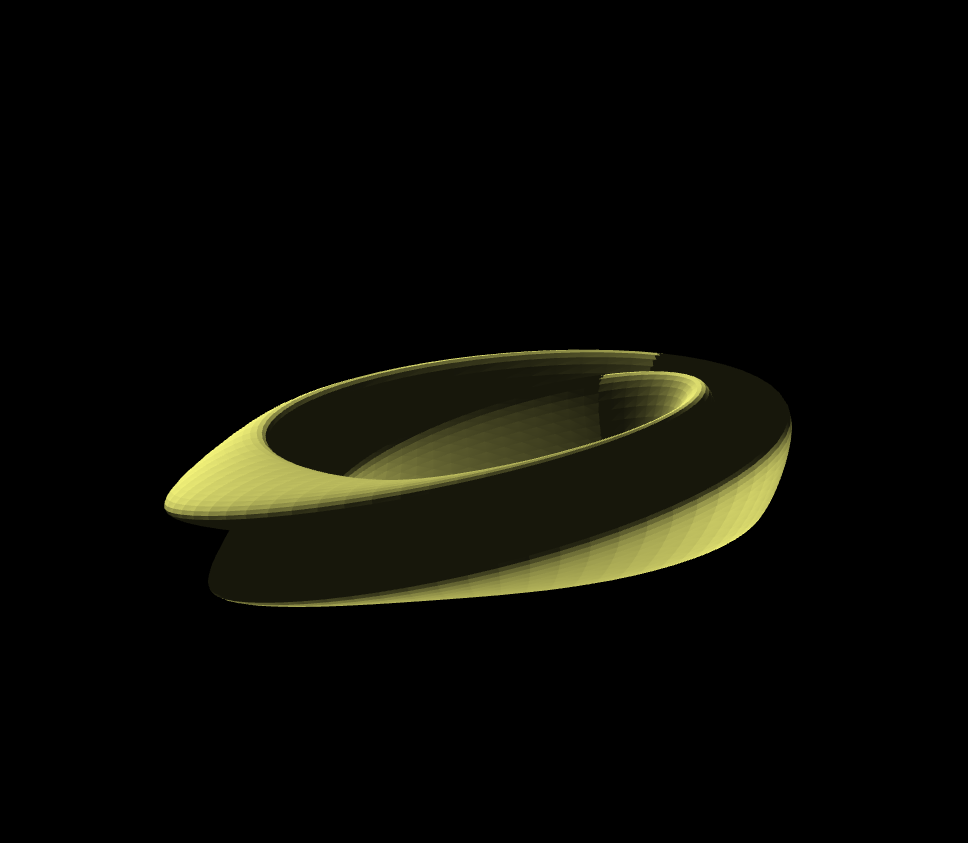
* підвищені вимоги до апаратної частини і пам’яті комп’ютера;
* необхідність проведення великої підготовчої роботи по створенню моделей всіх об’єктів сцени і призначення їм матеріалів;
* меншу, ніж в двовимірній графіці, свободу в формуванні зображень;
* необхідність контролю за взаємним положенням об’єктів в складі сцени;
* неправдоподібну ідеальність результатів візуалізації.

В тривимірній графіці оболонки об’єктів, незалежно від їх форми, складаються з трикутникових граней, що утворюють сітку. Кожна грань має три вершини і три ребра. Суміжні грані, що лежать в одній площині, утворюють багатокутник, або полігон, тому сітку часто називають полігональною. Ребра між гранями, які не лежать в одній площині, зображують на сітці суцільними лініями, а між гранями, які лежать в одній площині – пунктиром. Нормаль (перпендикуляр до поверхні грані) дозволяє визначити, чи буде дана грань видимою. Видимими вважаються тільки ті грані, нормалі яких направлені в бік спостерігача. Кожна грань задається координатами своїх вершин (Х,Y,Z).

Кожний об’єкт розміщується в так званому габаритному контейнері, який являє собою прямокутний паралелепіпед, описаний навкруги об’єкту. В момент створення об’єкту сторони габаритного контейнера орієнтуються паралельно координатним плоскостям глобальної системи координат, а при повертаннях об’єкту разом з ним повертається і його габаритний контейнер. Габаритні контейнери дозволяють програмі швидко визначати, чи закривають об’єкти один одного при спостеріганні сцени з певного напряму. Вони використовуються при підгонці розміру об’єкту під розмір вікна, визначають геометричний центр об’єктів складної форми (центр габаритного контейнера), дозволяють відобразити об’єкти в вигляді габаритного контейнера.

**Хід виконання завдання**

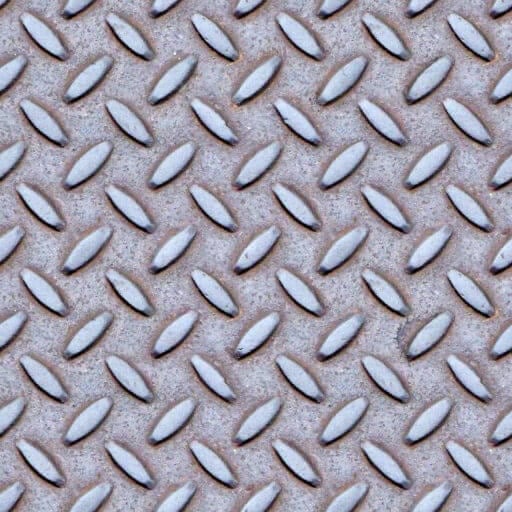
В ході виконання лабораторної роботи №2, була створена фігура The Klein Bottle.



Для виконання завдання з накладанням текстури було застосовано картинку з інтернету формату jpg. Слідом, вигружено на github, щоб безперешкодно використовувати в подальшому без багів і проблем з Cross-Origin Resource Sharing policy.

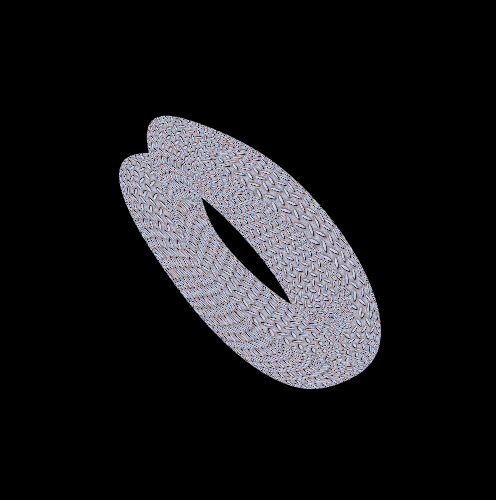
В ході виконання було створено змінні в коді шейдера, створенні посилання в коді програмі для належної роботи.

Зображення картинки текстури яку ми будемо розміщувати на фігурі The Klein Bottle.



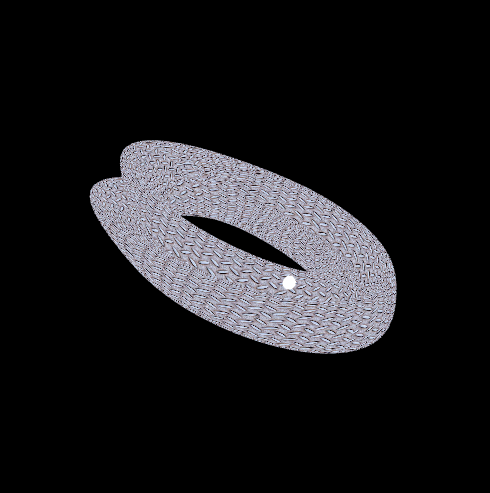
Вибрана текструра для накладання на фігуру The Klein Bottle

На цьому етапі ми можемо побачити The Klein Bottle вже з накладеною текстурою.



Було створенно умовну точку за допомогою якої буде відображатися рух фігури, для створення цього завдання було розроблено відповідні функції в коді програми. Для відорбраження сфери необхідно було створити функцію, яка б створювала геометрію для неї.

На цьому етапі виконання розрахункової роботи можемо побачити точку на фігурі з текстурою.



Для роботи з самою текстурою було створено додаткові функції та змінні в коді шейдера, такі як:

* Обертання фігури
* Розташування умовної точки на координатах
* Змінну для розташування сфери на відповідне місце поверхні

Реалізацію переміщення точки на поверхні та обертання фігури було розроблено за допомогою відповідних допоміжних функцій у коді програми у відповідь на вхідні дані користувача

**Посібник користувача**

Реалізовано можливість користувачу переміщувати точку на поверхні та обертати текстуру відносно умовної точки. Переміщення умовної точки реалізовано за допомогою введення з клавіатури:

* W та S здійснюють переміщення точки за параметрами в додатньому та від’ємному напрамках відповідно, клавіші
* A та D здійснюють переміщення точки за параметром u у від’ємному та додатньому напрямках відповідно.

Орієнтація поверхні в просторі, а також трансформація текстури здійснюється за допомогою введення з миші: необхідно затиснути лівою клавішею миші у області відображення поверхні та потягнути в будь-яку сторону. Варто зауважити, що обертання текстури залежить від переміщення миші по горизонталі, тобто, при зміщенні затиснутої мишу тільки вгору або тільки вниз буде здійснюватись лише зміна орієнтації поверхні в просторі

**Код програми**

**shader.gpu**

// Vertex shader

const vertexShaderSource = `

attribute vec3 vertex;

attribute vec3 normal;

attribute vec2 texCoord;

varying vec3 normalInterp;

uniform mat4 ModelViewProjectionMatrix;

uniform mat4 NormalMatrix;

uniform vec3 lightPos;

varying vec4 color;

varying vec2 v\_texcoord;

uniform vec3 translateUP;

uniform vec2 userPoint;

uniform float rotA;

varying float col;

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 zrotation(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() {

col=rotA;

mat4 rotate = zrotation(rotA);

mat4 translate1 = translation(userPoint.x,userPoint.y,0.0);

mat4 translate2 = translation(-userPoint.x,-userPoint.y,0.0);

vec4 texCoordTr1 = vec4(texCoord,0.0,0.0)\*translate1;

vec4 texCoordRot = texCoordTr1\*rotate;

vec4 texCoordTr2 = texCoordRot\*translate2;

v\_texcoord = vec2(texCoordTr2.x,texCoordTr2.y);;

vec4 nV = vec4(vertex,1.0);

if (rotA>137.0)

{

nV = translation(translateUP.x,translateUP.y,translateUP.z)\*nV;

}

gl\_Position = ModelViewProjectionMatrix \* nV;

}

`;

// Fragment shader

const fragmentShaderSource = `

#ifdef GL\_FRAGMENT\_PRECISION\_HIGH

precision highp float;

#else

precision mediump float;

#endif

varying vec4 color;

varying vec2 v\_texcoord;

varying float col;

uniform sampler2D tmu;

void main() {

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

if (col>137.0)

{

texColor = vec4(1.0,1.0,1.0,1.0);

}

gl\_FragColor = texColor;

}`;

main.js

"use strict";

let gl; // The webgl context.

let surface1; // A surface model

let surface2; // A surface model

let sphere; //user point visualization

let shProgram; // A shader program

let spaceball; // A SimpleRotator object that lets the user rotate the view by mouse.

let userPointCoord;

let userRotAngle;

function deg2rad(angle) {

return (angle \* Math.PI) / 180;

}

// Constructor

function Model(name) {

this.name = name;

this.iVertexBuffer = gl.createBuffer();

this.iNormalBuffer = gl.createBuffer();

this.iTextureBuffer = gl.createBuffer();

this.count = 0;

this.texCount = 0;

this.BufferData = function (vertices) {

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

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

this.count = vertices.length / 3;

};

this.NormalBufferData = function (normals) {

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

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

this.count = normals.length / 3;

};

this.TextureBufferData = function (points) {

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

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

this.texCount = points.length / 2;

};

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.iNormalBuffer);

gl.vertexAttribPointer(shProgram.iAttribNormal, 3, gl.FLOAT, false, 0, 0);

gl.enableVertexAttribArray(shProgram.iAttribNormal);

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

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

gl.enableVertexAttribArray(shProgram.iAttribTexture);

gl.drawArrays(gl.TRIANGLE\_STRIP, 0, this.count);

};

this.DrawPoint = 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.LINE\_STRIP, 0, this.count);

};

}

// Constructor

function ShaderProgram(name, program) {

this.name = name;

this.prog = program;

// Location of the attribute variable in the shader program.

this.iAttribVertex = -1;

this.iAttribNormal = -1;

this.iAttribTexture = -1;

// Location of the uniform specifying a color for the primitive.

this.iColor = -1;

// Location of the uniform matrix representing the combined transformation.

this.iModelViewProjectionMatrix = -1;

this.iNormalMatrix = -1;