Міністерство освіти і науки України

Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Кафедра інженерії програмного забезпечення в енергетиці

Розрахункова графічна робота

з дисципліни «Візуалізація графічної та геометричної інформації»

Варіант №26

Виконав студент 5-го курсу

ІАТЕ групи ТР-32мп

Фіцай Володимир Романович

Перевірив: Демчишин Анатолій Анатолійович

Київ – 2023

**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.

**Теорія**

WebGL (Web Graphics Library) - це програмна бібліотека для мови JavaScript, призначена для візуалізації інтерактивної тривимірної та двовимірної графіки в межах сумісності веб-браузера без використання плагінів. WebGL вносить тривимірну графіку в Інтернет, представляючи API, яке ґрунтується на OpenGL ES 2.0, що дозволяє використовувати його в елементах canvas HTML5.

Однією з ключових особливостей WebGL є використання шейдерів - невеликих програм, що виконуються на відеокарті для обробки графічних даних. Є два типи шейдерів: вершинний і фрагментний, які контролюють обчислення координат вершин та кольорів пікселів. Основні параметри включають контекст, який підтримує рендеринг, об'єкти буферів для зберігання даних, текстури для відображення зображень, та програми, які об'єднують шейдери для обчислень. Керування матрицями трансформацій, такими як матриці моделі, виду та проекції, також важливо для коректного відображення сцени. WebGL використовує систему координат OpenGL, де точка (0, 0, 0) розташована в центрі екрана, а вісь Z вказує на спостереження вглиб. Це важливо при визначенні позиції та орієнтації об'єктів.

Відображення текстури з використанням текстурних координат (u, v) - це процес взаємодії між текстурою та геометричним об'єктом на екрані, де кожній точці поверхні об'єкта призначається конкретна частина текстури за допомогою текстурних координат.

Коли ми хочемо прикріпити картинку (текстуру) до якогось об'єкта на екрані, нам потрібно сказати комп'ютеру, яка частина цієї картинки відповідає кожній точці об'єкта. Це робиться за допомогою так званих "текстурних координат" (часто позначаються як u і v). Кожній точці об'єкта присвоюються ці координати, які показують, яка частина картинки відповідає цій точці. Комп'ютер інтерполює ці координати для кожної точки об'єкта, тобто знаходить проміжне значення між визначеними на краях об'єкта. Потім використовуючи ці координати, він визначає, як частину картинки відобразити в кожній точці об'єкта, створюючи таким чином реалістичний вигляд.

Ось короткий список головних компонентів, що необхідні для роботи з графікою:

* буфери - це масиви бінарних даних, які завантажуються на графічний процесор. Зазвичай у буферах зберігаються речі, такі як положення вершин, нормалі, текстурні координати, кольори вершин і т.д., хоча ви можете поміщати в них будь-що.
* атрибути визначають, як дані з ваших буферів передаються у вершинний шейдер. Наприклад, ви можете помістити положення вершин в буфер як три 32-бітні числа з рухомою комою на одне положення. Ви вказуєте конкретному атрибуту, звідки брати положення вершин, який тип даних використовується (три 32-бітні числа з рухомою комою), з якого індексу в буфері починаються положення вершин і скільки байтів потрібно взяти від одного положення до наступного.
* Uniform-змінні - це глобальні змінні, які встановлюються перед виконанням програми шейдера.
* текстури - це масиви даних, до яких є довільний доступ у програмі шейдера. Зазвичай в текстуру поміщається зображення, але текстура - це просто набір даних, і ви можете вставити в неї щось інше, крім набору кольорів.
* varying-змінні дозволяють передавати дані з вершинного шейдера фрагментному шейдеру. У фрагментному шейдері ми отримаємо інтерпольовані значення з вершинного шейдера - це залежить від того, чи ми відображаємо точки, лінії чи трикутники.

**Деталі розробки**

За варіантом мені було надано Twice Oblique Trochoid Cylindroid. При виконанні практичного завдання №2 було розроблено програму, що виводить поверхню у вигляді суцільних трикутників.

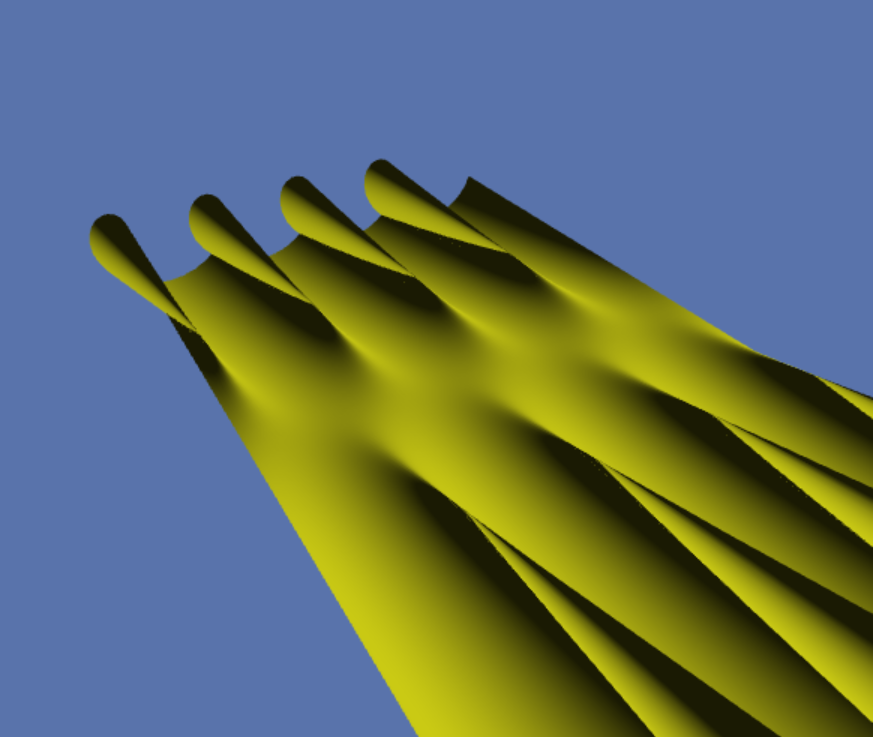


Рисунок 1 – Результат моделювання поверхні в практичній 2

В рамках виконання завдання, необхідно обрати зображення, що буде накладено на поверхню. Нижче виведено його вигляд. Формат зображення .jpg (512х512).



Рисунок 2 – Зображення, що використано як текстуру

При виконанні завдання вказану текстуру було накладено на поверхню. Для накладання текстури на поверхню було підготовлено буфер текстурних координат, кожна з яких відповідає елементу масива з буферу вершин. В рамках завдання було реалізовано обертання текстури(парний варіант). Також, було створено uniform, що визначає який визначатиме кут обертання. Для визначення кольору пікселів при накладанні текстури, було використано texture2D(). Нижче представлено результат накладання обраної текстури на площину.

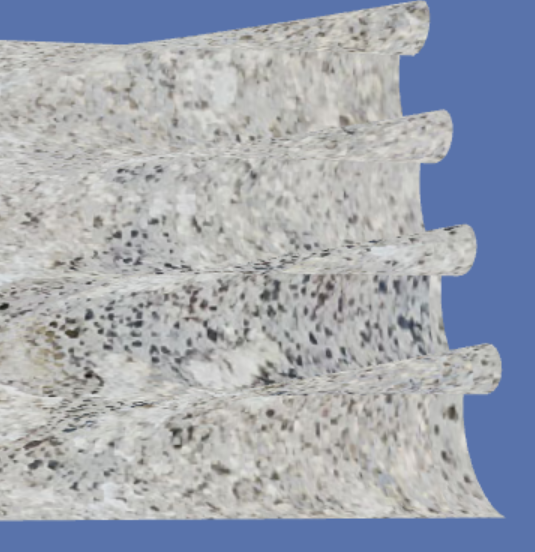


Рисунок 3 – Поверхня з текстурою

Для відображення точки відносно якої змінюватиметься відображення текстури було створено нову модель у вигляді сфери. На фото нижче її вигляд.

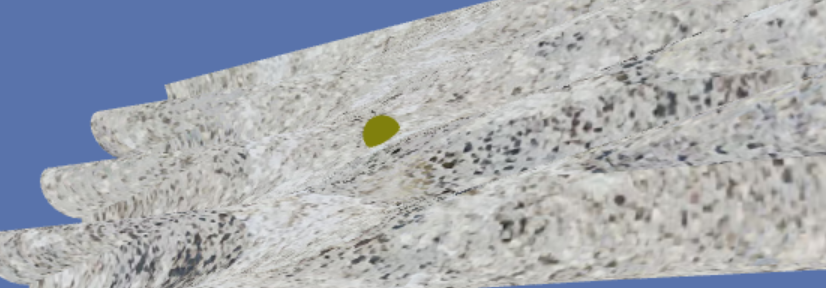


Рисунок 4 – Вигляд точки

**Інструкції**

Для оберту фігури необхідно скористатися мишкою, натиснувши будь-яку клавішу миші і потягнувши миш в сторону.

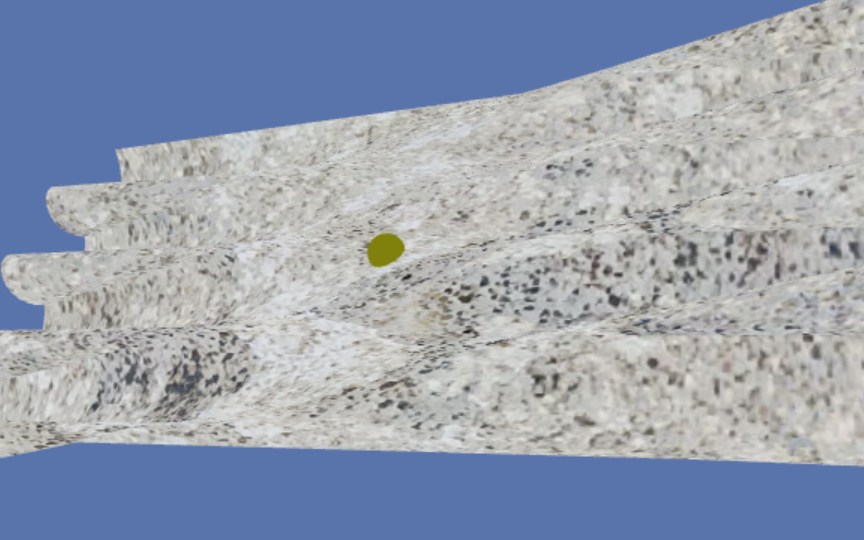


Рисунок 5 – До обертання поверхні

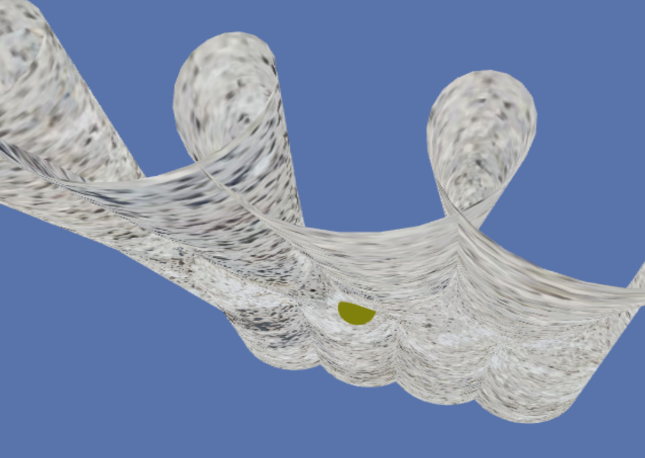


Рисунок 6 – Після обертання поверхні

Для виконання обертання текстури, необхідно скористатися клавішами WASD. Варто зазначити, що обертання обмежене краєм текстури. Крім цього, обертання можливе лише в одному напрямку в один момент часу і воно не залежить від положення камери.

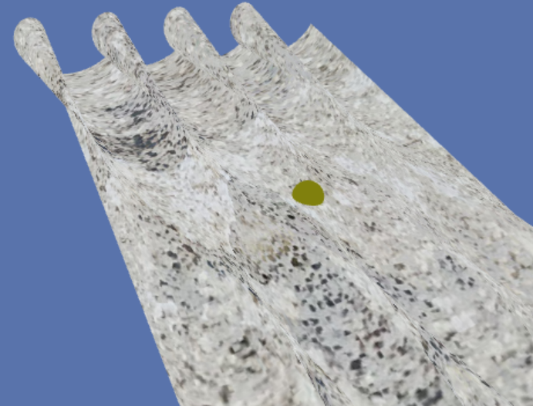


Рисунок 7 – До обертання текстури

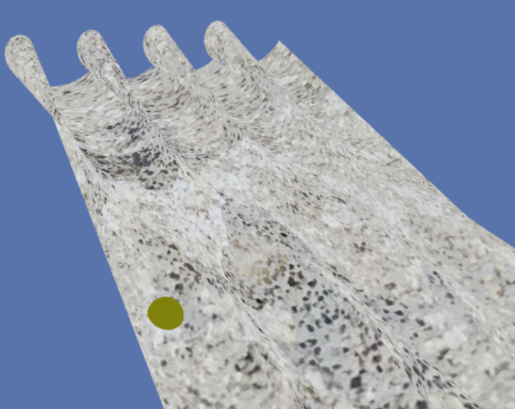


Рисунок 8 – Після обертання текстури

Для збільшення куту обертання необхідно скористатися повзунком з назвою Angle. Чим більше значення, тим більший кут обертання. За замовчуванням, його значення мінімальне (0,1). Індикатором переміщення може слугувати як сама текстура, так і точка на поверхні, що рухається разом з нею.

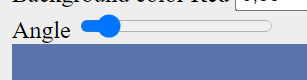


Рисунок 9 – Повзунок для зміни куту

Крім цього, можливо змінити колір заднього фону і колір точки індикатора положення текстури. Нижче продемонстровано результат цих змін.

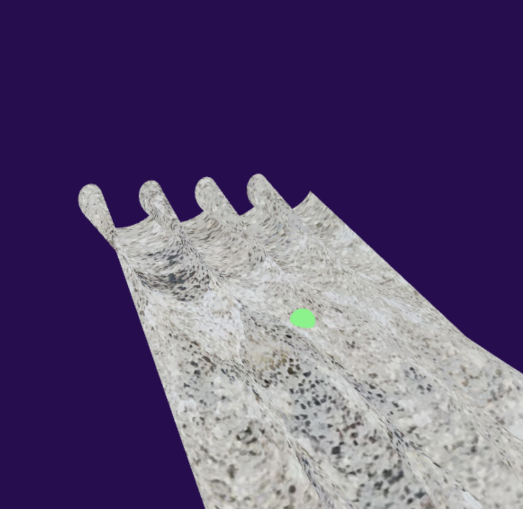


Рисунок 10 – Зміна кольорової палітри

**Код програми:**

function cassiniVertex(u, v) {

omega = p \* u

let x = (c \* u + v \* (Math.sin(fi) + Math.tan(a) \* Math.cos(fi) \* Math.cos(omega))),

y = (v \* Math.tan(a) \* Math.sin(omega)),

cV = (H + v \* (Math.tan(a) \* Math.sin(fi) \* Math.cos(omega) - Math.cos(fi)));

return [scaler \* x, scaler \* y, scaler \* cV];

}

function draw() {

let b\_red = document.getElementById('b\_red').value

let b\_green = document.getElementById('b\_green').value

let b\_blue = document.getElementById('b\_blue').value

gl.clearColor(b\_red, b\_green, b\_blue, 1);

gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

let fov = Math.PI / 4;

let aspectRatio = 1;

let near = 1;

let far = 50;

let projection = m4.perspective(fov, aspectRatio, near, far);

let modelView = spaceball.getViewMatrix();

let rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0.7);

let translateToPointZero = m4.translation(-2, -1, -10);

let matAccum0 = m4.multiply(rotateToPointZero, modelView);

let matAccum1 = m4.multiply(translateToPointZero, matAccum0);

let modelViewProjection = m4.multiply(projection, matAccum1);

const normalMat = m4.identity();

m4.inverse(modelView, normalMat);

m4.transpose(normalMat, normalMat);

gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, modelViewProjection);

gl.uniformMatrix4fv(shProgram.iNormalMat, false, normalMat);

gl.uniform4fv(shProgram.iColor, [1, 1, 0, 1]);

let m\_red = document.getElementById('m\_red').value

let m\_green = document.getElementById('m\_green').value

let m\_blue = document.getElementById('m\_blue').value

gl.uniform3fv(shProgram.iDiffuseColor, [m\_red, m\_green, m\_blue]);

gl.uniform2fv(shProgram.iPointCoords, pointCoords);

gl.uniform1f(shProgram.iAngle, document.getElementById('a').value);

surface.Draw();

gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, m4.multiply(modelViewProjection,

m4.translation(...cassiniVertex(pointCoords[0], map(pointCoords[1], 0, 1, -5, 5)))));

gl.uniform1f(shProgram.iAngle, -100.0);

point.Draw();

}

function init() {

document.getElementById('b\_red').onchange = (e) => {

draw()

}

document.getElementById('b\_green').onchange = (e) => {

draw()

}

document.getElementById('b\_blue').onchange = (e) => {

draw()

}

document.getElementById('m\_red').onchange = (e) => {

draw()

}

document.getElementById('m\_green').onchange = (e) => {

draw()

}

document.getElementById('m\_blue').onchange = (e) => {

draw()

}

document.getElementById('a').onchange = (e) => {

draw()

}

window.onkeydown = (e) => {

if (e.keyCode == 87) {

pointCoords[0] = Math.min(pointCoords[0] + 0.02, 1);

}

else if (e.keyCode == 65) {

pointCoords[1] = Math.max(pointCoords[1] - 0.02, 0);

}

else if (e.keyCode == 83) {

pointCoords[0] = Math.max(pointCoords[0] - 0.02, 0);

}

else if (e.keyCode == 68) {

pointCoords[1] = Math.min(pointCoords[1] + 0.02, 1);

}

draw()

}