**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”

Варіант 17

Виконав студент групи ТР-31мп

Мітлицький Сергій Юрійович

Київ - 2023

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. Клавіші A і D переміщують точку вздовж параметра u, а клавіші W і S переміщують точку вздовж параметра v.

1. **Теорія**

WebGL

WebGL - це JavaScript API для створення інтерактивної 2D та 3D графіки у веб-браузері. Щоб реалізувати відображення текстур, масштабування, обертання та інтерактивне переміщення точок, WebGL API надає функції для обробки шейдерів, текстур та матричних перетворень.

Шейдери використовуються для визначення того, як обробляються вершини та фрагменти, дозволяючи включати текстурні координати та перетворення. Текстурні об’єкти створюються та прив’язуються до певних текстурних одиниць для вибірки в шейдерах. Матричні перетворення застосовуються для керування положенням, масштабуванням і обертанням 3D-моделі.

Шейдери WebGL

В конвеєрі візуалізації у WebGL шейдери відіграють ключову роль, забезпечуючи гнучкий та програмований спосіб визначення вигляду 3D-графіки. У WebGL існують два основних види шейдерів: вершинні та фрагментні. Вершинні шейдери відповідають за обробку кожної вершини 3D-моделі перед її відображенням на екрані. Ці шейдери отримують атрибути, такі як положення, колір і текстурні координати, з даних вершин і обробляють їх для створення виходу. Один із поширених методів використання вершинних шейдерів - застосування трансформацій до позицій вершин, що включає переміщення, обертання та масштабування, важливі для позиціонування об'єктів у 3D-просторі. Вихідні дані вершинного шейдера містять трансформовану позицію вершини та будь-які інтерпольовані значення, що передаються фрагментному шейдеру.

Фрагментні шейдери, іноді відомі як піксельні, опрацьовують кожен піксель, який буде намальований на екрані. Вони отримують інтерпольовані значення від вершинного шейдера, такі як координати кольору та текстури, і інші дані, такі як інформація про освітлення. Основне завдання фрагментного шейдера - визначити остаточний колір кожного пікселя, що може включати вибірку текстури, обчислення освітлення та інші ефекти. Кінцевий вихідний колір використовується для малювання пікселя на екрані.

Обидва види шейдерів - вершинні та фрагментні - написані мовою GLSL (OpenGL Shading Language) і потребують компіляції перед використанням. Після компіляції вони об'єднуються в програму шейдера, яку необхідно пов'язати перед відтворенням, щоб WebGL міг використовувати зазначені вершинні та фрагментні шейдери для конвеєра відтворення.

Uniforms - це значення, які залишаються сталими для всіх вершин або фрагментів під час візуалізації примітиву. Вони дозволяють передавати шейдерам зовнішні дані, такі як матриці трансформації або інформацію про глобальне освітлення.

Attributes - це дані для кожної вершини, які відрізняються між вершинами. Вони використовуються для передачі інформації, такої як положення вершин, нормалі та координати текстури.

Координати текстури зазвичай передаються як атрибути до вершинного шейдера, а потім інтерполюються через примітив для використання у фрагментному шейдері. У фрагментному шейдері координати текстури використовуються для вибірки кольорів із текстур, які потім використовуються для визначення остаточного кольору кожного пікселя.

Накладання текстур. Текстурні координати

Відображення текстури — це техніка в комп’ютерній графіці, яка дозволяє відтворювати реалістичні поверхні шляхом нанесення зображень або текстур на 3D-моделі. У контексті WebGL відображення текстур передбачає асоціювання кожної вершини 3D-об’єкта з координатами текстури (u, v), які потім використовуються для вибірки кольорів із зображення текстури. Це покращує візуальний вигляд об’єкта, надаючи детальну інформацію про поверхню.

Координати текстури (u, v) є важливими параметрами, які визначають спосіб нанесення текстури на поверхню. Вони варіюються від 0 до 1 і використовуються для звернення до конкретних точок на зображенні текстури. У контексті WebGL координати текстури пов’язані з кожною вершиною 3D-моделі та інтерполюються по всій поверхні під час візуалізації. Це гарантує, що текстура точно охоплює модель.

1. **Деталі розробки**

За варіантом мені було надано “Surface of revolution of a General Sinusoid”. При виконанні практичного завдання №3 було розроблено програму, що виводить поверхню у вигляді суцільних трикутників.



Рисунок 1 - Вигляд поверхні

Було обрано наступне зображення для подальшого виконання розрахунково-графічної роботи. Для підтримки на більшій кількості операційних систем та браузерів було обрано зображення розміром 512х512 пікселів. Формат зображення .jpg



Рисунок 2 - Зображення текстури

Було накладено текстуру на поверхню. Для накладання текстури на поверхню було підготовлено буфер текстурних координат, кожна з яких відповідає елементу масива з буферу вершин. Згідно варіанту текстура має масштабуватися, тому було створено відповідний uniform, який визначатиме фактор масштабування. 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

Було створено новий обʼєкт класу Model для відображення точки відносно якої буде здійснюватись трансформація текстури. Обʼєкт графічно відображається як сфера. Сфера перебуває на поверхні.



Рисунок 4 - Точка на текстурованій поверхні

Згідно варіанту було імплементовано масштабування текстури відносно точки на поверхні.

1. **Інструкції користувача**

Фігуру можна обертати відносно центру затиснувши ліву клавішу миші та потягнувши в сторону бажаного обертання.

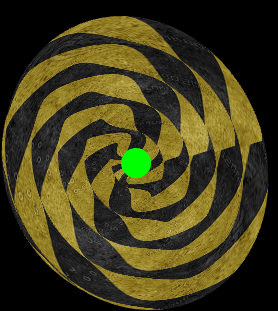


Рисунок 5 - Вигляд фігури до та після обертання

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 “Зміна фактору масштабування” можна змінювати фактор масштабування відносно умовної точки на поверхні. Якщо значення фактора мастабування не нульове можна помітити, що при переміщенні точки зміщується і текстура, адже масштабування відбувається відносно іншої точки на поверхні , яка в свою чергу відповідає іншій текстурній координаті.

Рисунок 7 - Демонстрація зміни кута фактору масштабування текстури

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

1. **Код програми**

function draw() {

    gl.clearColor(0, 0, 0, 1);

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

    /\* Set the values of the projection transformation \*/

    let frustrum = 2

    let projection = m4.orthographic(-frustrum, frustrum, -frustrum, frustrum, -frustrum, frustrum);

    /\* Get the view matrix from the SimpleRotator object.\*/

    let modelView = spaceball.getViewMatrix();

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

    let translateToPointZero = m4.translation(0, 0, 0);

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

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

    /\* Multiply the projection matrix times the modelview matrix to give the

       combined transformation matrix, and send that to the shader program. \*/

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

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

    /\* Draw the six faces of a cube, with different colors. \*/

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

    let dC = document.getElementById('color').value

    let sC = document.getElementById('colorS').value

    gl.uniform3fv(shProgram.iDiffuseColor, [-100, 0, 0]);

    gl.uniform3fv(shProgram.iSpecularColor, hexToRgb(sC));

    gl.uniform2fv(shProgram.iTexball, texball);

    gl.uniform1f(shProgram.iScaler, document.getElementById('scaler').value);

    surface.Draw();

    gl.uniform3fv(shProgram.iDiffuseColor, hexToRgb(dC));

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

        m4.translation(...CreateVertex(texball[0], deg2rad(texball[1] \* 360)))

    ));

    ball.Draw();

}

window.onkeydown = (e) => {

    if (e.keyCode == 87) { //w

        texball[0] = Math.min(texball[0] + 0.01, 1);

    }

    else if (e.keyCode == 65) { //a

        texball[1] = Math.max(texball[1] - 0.01, 0);

    }

    else if (e.keyCode == 83) { //s

        texball[0] = Math.max(texball[0] - 0.01, 0);

    }

    else if (e.keyCode == 68) { //d

        texball[1] = Math.min(texball[1] + 0.01, 1);

    }

    draw();

}

function hexToRgb(hex) {

    var result = /^#?([a-f\d]{2})([a-f\d]{2})([a-f\d]{2})$/i.exec(hex);

    return [

        parseInt(result[1], 16) / 255,

        parseInt(result[2], 16) / 255,

        parseInt(result[3], 16) / 255

    ]

}

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 = "https://raw.githubusercontent.com/Smit0001/VGGI/CGW/texture.jpg";

    image.onload = () => {

        gl.bindTexture(gl.TEXTURE\_2D, texture);

        gl.texImage2D(

            gl.TEXTURE\_2D,

            0,

            gl.RGBA,

            gl.RGBA,

            gl.UNSIGNED\_BYTE,

            image

        );

        console.log("imageLoaded")

        draw()

    }

}