Міністерство освіти і науки України

Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Кафедра цифрових технологій в енергетиці

Звіт з розрахунково-графічної роботи

з дисципліни “Візуалізація графічної та геометричної інформації”

на тему “Операції з текстурними координатами”

Варіант: 16

Виконав:

студент 5-го курсу групи ТР-32мп

Мисько Андрій Олексійович

Перевірив:

Демчишин А. А.

Київ – 2023

**Завдання:**

* Нанесіть текстуру на поверхню з практичного завдання №2.
* Реалізація масштабування текстури (координати текстури) масштабування/обертання навколо визначеної користувачем точки – непарні варіанти реалізують масштабування, парні варіанти реалізують обертання
* Повинна бути можливість переміщати точку вздовж простору поверхні (u, v) за допомогою клавіатури. Наприклад, клавіші A і D переміщують точку вздовж параметра u , клавіші W і S переміщують точку вздовж параметра v.

**Опис теорії**

WebGL (Web Graphics Library) - це технологія, яка дозволяє вбудовувати тривимірну та двовимірну графіку безпосередньо в веб-браузери. Вона базується на мові програмування JavaScript та використовує графічний контекст HTML5 для відображення складної графіки на веб-сторінках. WebGL використовує апаратну акселерацію графіки на стороні клієнта, що дозволяє створювати швидкі та ефективні візуальні ефекти та 3D-сцени без необхідності встановлення додаткових плагінів.

WebGL базується на стандарті OpenGL ES (OpenGL for Embedded Systems), який спрощує взаємодію із графічним апаратом. Використовуючи WebGL, можна створювати інтерактивні веб-сайти та додатки з високоякісною графікою, такі як візуалізації даних, ігри та інше. Загалом WebGL дозволяє використовувати апаратні можливості клієнтського пристрою для оптимізації відтворення складної графіки у реальному часі на веб-платформі.

У WebGL шейдери - це невід'ємна частина програмного забезпечення, використовуваного для обробки та відображення графіки в графічному контексті браузера за допомогою технології WebGL. Вони представляють собою малий програмний код, написаний мовою GLSL (OpenGL Shading Language), який використовується для визначення вигляду та обробки графічних об'єктів.

Існують два основних типи шейдерів: вершинний (vertex shader) та фрагментний (fragment shader). Вершинний шейдер відповідає за обчислення позицій вершин графічних об'єктів, визначення їх текстурних координат та інших властивостей. Фрагментний шейдер визначає кольори та інші характеристики пікселів (фрагментів), що утворюють графічний об'єкт на екрані.

Шейдери дозволяють розробникам здійснювати різноманітні візуальні ефекти, розраховувати освітлення, тіні та інші аспекти графіки у реальному часі. Вони сприяють створенню динамічних та інтерактивних веб-сцен, забезпечуючи високий рівень гнучкості та керованості у відображенні графічної інформації.

Вершинний шейдер відповідає за обробку вершин графічних об'єктів. Його основним завданням є обчислення нових координат вершин, визначення їх позицій у тривимірному просторі. Цей шейдер може виконувати операції, такі як трансформації координат, масштабування, обертання, інтерполяція текстурних координат та інші дії, які впливають на положення вершин.

Вершинний шейдер також може обробляти атрибути вершин, такі як колір чи нормалі, і передавати їх для використання у фрагментний. Крім того, він визначає фінальні координати вершин, які будуть передані фрагментному шейдеру для подальшого обчислення.

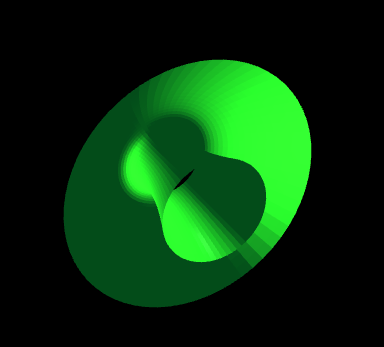
Фрагментний шейдер відповідає за обробку фрагментів (пікселів) графічного об'єкта. Його основне завдання - визначити кольори та інші атрибути кожного пікселя на екрані. Фрагментний шейдер може виконувати операції освітлення, тінь, визначення кольорів за текстурою, а також різноманітні візуальні ефекти. Він отримує на вхід атрибути, які були передані з вершинного шейдера, і використовує їх для розрахунків. Найважливішою операцією фрагментного шейдера є встановлення кольору, який буде відображений для кожного пікселя. Крім того, фрагментний шейдер може впливати на прозорість та інші характеристики візуалізації.

В сукупності вершинний та фрагментний шейдери дозволяють реалізувати широкий спектр графічних ефектів та динамічних візуальних змін у веб-графіці за допомогою WebGL.

**Опис деталей реалізації**

Фігура за варіантом: Surface of Revolution of a Parabola of Arbitrary Position

Рис. 1: Фігура за варіантом



Текстура для накладання була обрана наступною (рис. 2).

Рис. 2: Обрана текстура

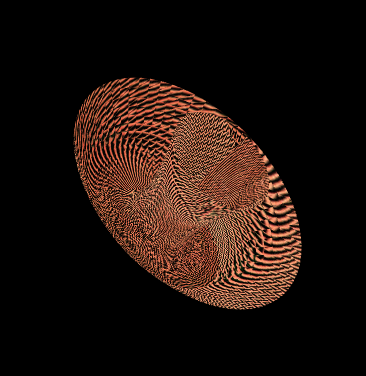


Текстуру було завантажено на Гітхаб перед накладанням для того, щоб використати у коді програми посилання на текстурне зображення відразу з цього середовища.

Було додано нові змінні для координат з метою накладання текстури на поверхню фігури, після чого було створено для них зв’язок з шейдером в коді.

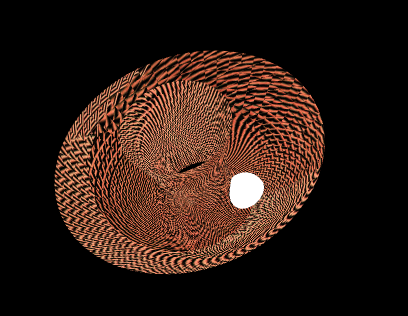
Результат наступний (рис. 3):

Рис. 3: Накладена текстура



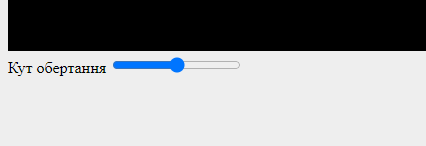
Було додано точку, навколо якої буде виконуватися обертання за варіантом. Для цього введено координати відносно поверхні фігури у нову змінну та додано необхідні модернізації у код функцій draw() та initGL(). Для переміщення фігури по клавіутурі використано специфікований код, що оновлює положення точки по координатам при натисканні зазначених у завданні кнопок.

Рис. 4: Фігура з точкою



Для обертання задано повзунок, дані з якого передаватимуться з htmlфайлу – в основний код програми з подальшим обертанням фігури по куту, зазначеному з повзунка, відносно нинішнього положення точки.

Рис. 5: Повзунок



**Інструкція користувача**

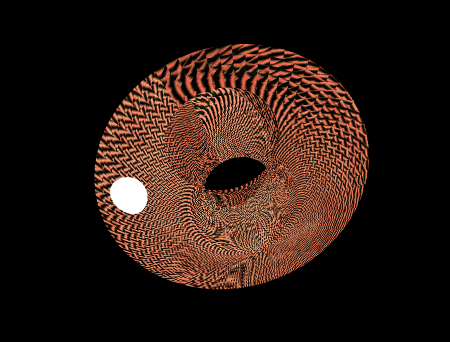
Через рух миші з натисканням її лівої кнопки можливо змінювати положення фігури (як в практичних роботах).

Рис. 6: Зміна положення.



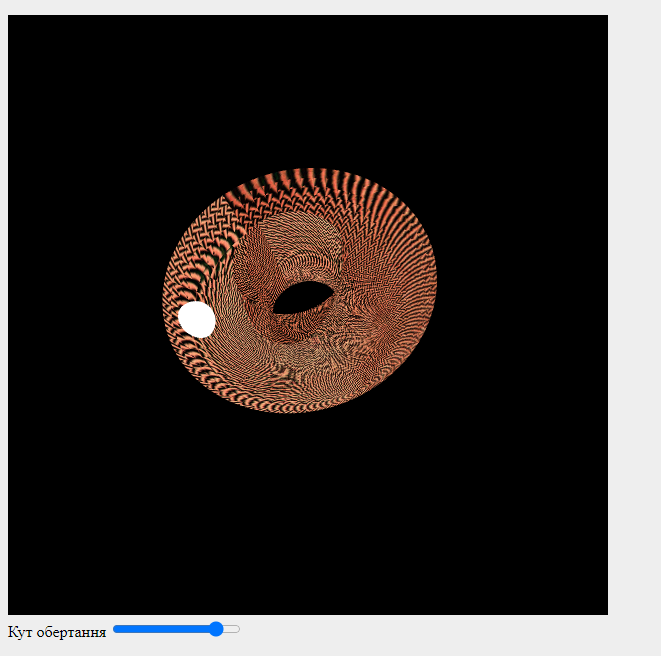
Через клавіші W, S, A, D реалізовано переміщення точки по фігурі – A та D по координаті u відповідно на зменшення та збільшення, W та S – по координаті vтак само на зменшення та збільшення. Разом з переміщенням точки обертається і текстура по поверхні фігури відносно точки.

Рис. 7: Переміщення точки



За допомогою повзунка можна обертати текстуру через зміну кута обертання.

Рис. 8: Зміна кута обертання



**Вихідний код (приклади)**

function draw() {

    gl.clearColor(0, 0, 0, 1);

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

    let projection = m4.orthographic(-3, 3, -3, 3, -3, 3)

    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);

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

    let normalMatrix = m4.identity();

    m4.inverse(modelView, normalMatrix);

    normalMatrix = m4.transpose(normalMatrix, normalMatrix);

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

    gl.uniformMatrix4fv(shProgram.iNormalMatrix, false, normalMatrix);

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

    gl.uniform2fv(shProgram.iballpos, ballpos);

    gl.uniform1i(shProgram.iOsv, false);

    surface.Draw();

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

    gl.uniform1i(shProgram.iOsv, true);

    gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, m4.multiply(modelViewProjection, m4.translation(...Object.values(vertex(ballpos[0],ballpos[1], 2)))));

    sphereSurface.Draw();

}

function init() {

    let canvas;

    try {

        canvas = document.getElementById("webglcanvas");

        gl = canvas.getContext("webgl");

        if (!gl) {

            throw "Browser does not support WebGL";

        }

    }

    catch (e) {

        document.getElementById("canvas-holder").innerHTML =

            "<p>Sorry, could not get a WebGL graphics context.</p>";

        return;

    }

    try {

        initGL();

    }

    catch (e) {

        document.getElementById("canvas-holder").innerHTML =

            "<p>Sorry, could not initialize the WebGL graphics context: " + e + "</p>";

        return;

    }

    spaceball = new TrackballRotator(canvas, draw, 0);

    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/TRPZ-Study/VGGI/main/512.jpg";

    image.onload = () => {

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

        gl.texImage2D(

            gl.TEXTURE\_2D,

            0,

            gl.RGBA,

            gl.RGBA,

            gl.UNSIGNED\_BYTE,

            image

        );

        draw()

    }

    circleD();

}