Міністерство освіти й науки України

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 над текстурними координатами”

Варіант 9

Виконав:

Студент 1-го курсу

групи ТР-32мп

Журавель В.І.

Перевірив:

Демчишин А.А.

Київ-2024

**Завдання**

* Нанести текстуру на поверхню з практичного завдання №2.
* Реалізувати масштабування текстури (координат) 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.

**Теоритичні відомості**

**WebGL**

WebGL - це програмна бібліотека для мови JavaScript, призначена для візуалізації інтерактивної тривимірної та двовимірної графіки у межах веб-браузера без використання плагінів. WebGL вносить тривимірну графіку до веб-середовища, впроваджуючи API, побудоване на основі OpenGL ES 2.0, що дозволяє використовувати WebGL в елементах canvas HTML5.

Однією з переваг WebGL є взаємодія з апаратним забезпеченням пристрою, використовуючи функціональність, яка доступна на рівні низькорівневого програмування графічних процесорів.

Для реалізації поставленого завдання, WebGL API надає функції для текстур і перетворень матриці. Шейдери використовуються для визначення, як обробляються вершини та фрагменти, дозволяючи включати такі елементи, як текстурні координати. Текстурні об'єкти створюються та прив'язуються до конкретних текстурних одиниць для використання в шейдерах при вибірці. Застосовуються матричні перетворення для управління положенням та масштабуванням тривимірної моделі.

**Накладання текстур. Текстурні координати**

Відображення текстури — це метод у комп'ютерній графіці, що дозволяє реалістично відтворювати поверхні шляхом застосування зображень або текстур на 3D-моделі. У веб-графіці за допомогою WebGL цей процес включає асоціювання кожної вершини 3D-об'єкта з текстурними координатами (u, v), які використовуються для вибірки кольорів із зображення текстури. Це поліпшує візуальний вигляд об'єкта, додаючи деталі та інформацію про його поверхню. Текстурування є дуже важливою частиною процесу 3D-моделювання. Усі дрібніші візуальні характеристики у 3D-моделюванні, такі як зморшки та окремі нитки килима, є продуктом текстури, нанесеної 3D-художником

Текстурні координати - це пари чисел (зазвичай у вигляді (u, v)), які вказують на конкретну точку на текстурі, яка відповідає конкретній точці на поверхні об'єкта. Зазвичай координати лежать в інтервалі [0, 1], де (0, 0) - це верхній лівий кут текстури, а (1, 1) - нижній правий кут. Таким чином, текстурні координати визначають, як розмістити частину текстури на поверхні об'єкта.

У WebGL для накладання текстур використовуються шейдери - фрагментний та вершинний. У вершинному шейдері визначаються атрибути для текстурних координат, а у фрагментному шейдері використовується ця інформація для визначення кольору, який відображатиме текстуру на поверхні об'єкта. Нижче розглянуто основнні характеристики та відмінності.

Вершинний шейдер (Vertex Shader):

* Вершинний шейдер викликається для кожної вершини 3D об'єкта.
* Відповідає за обчислення та перетворення координат вершин, заданих у віртуальному просторі, у координати екрану.
* Зазвичай використовується для передачі додаткової інформації, такої як текстурні координати, фарби та інші параметри, у фрагментний шейдер.

Фрагментний шейдер (Fragment Shader):

* Фрагментний шейдер викликається для кожного пікселя, що відображається на екрані, після того як вершинний шейдер обробив всі вершини.
* Відповідає за визначення кольору кожного пікселя на основі різних параметрів, таких як текстурні координати та освітлення.
* Здатен створювати складні візуальні ефекти та текстурні зображення.

Основна робота шейдерів полягає в оптимізації та пришвидшенні процесу рендерингу графіки, а також у реалізації різноманітних візуальних ефектів. Вони програмуються мовою GLSL (OpenGL Shading Language), яка є спеціалізованою мовою для написання шейдерів. **Деталі розробки**

В ході другої лабораторної роботи було розроблено програму, що виводить поверхню під назвою «Snail Surface» використовуючи суцільні трикутники.

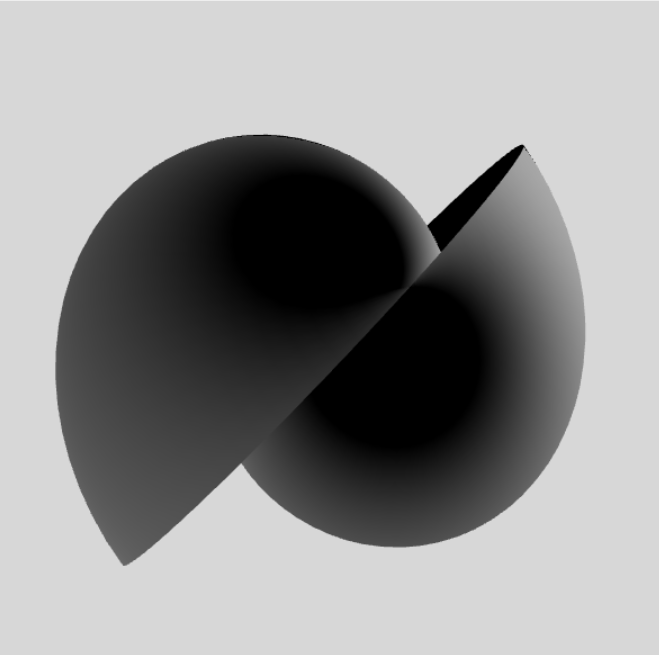


Рис. 1 - «Snail Surface».

Для текстури було обрано зображення розміром 512\*512 формату «.jpg». Зображення було завантажено на GitHub з метою подальшого використання посилань на неї та уникнення проблеми політики обміну ресурсів між джерелами (Cross-Origin Resource Sharing).



Рис. 2 – Обране зображення.

Було накладено текстуру на поверхню. Згідно варіанту текстура має масштабуватися. З метою накладання текстури на поверхню, в першу чергу було створено декілька змінних в коді шейдера. Після чого були створені посилання на змінні в коді програми. Поверхню за накладеною текстурою можна побачити на рисунку 3.

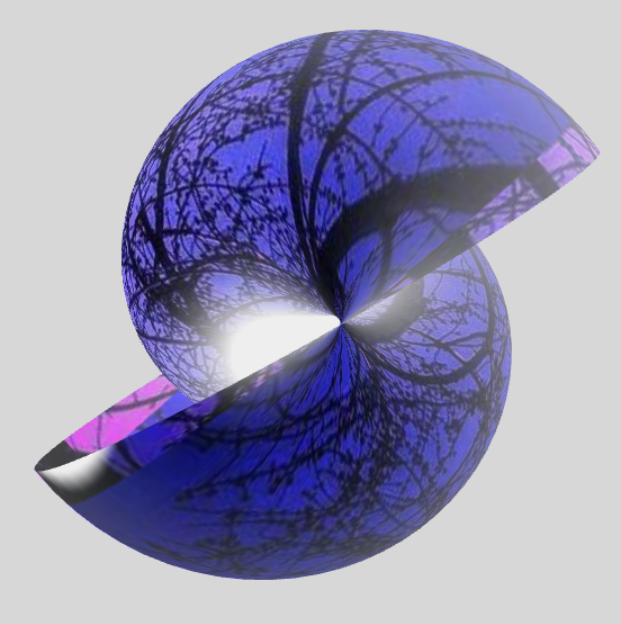


Рис. 3 – Поверхня з накладеною текстурою.

Для відображення точки відносно якої буде здійснюватись трансформація текстури було прийнято рішення використовувати сферу, що знаходиться на поверхні, оскільки наша робота відбувається в 3D-просторі. Модель з умовною точкою зображено на рисунку 4.

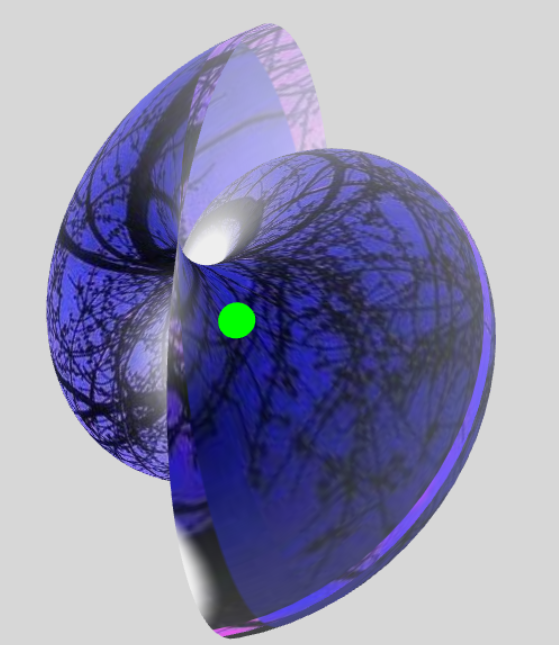


Рис. 4 – Поверхня з умовною точкою.

**Вказівки користувачу**

Користувач може керувати, обертанням текстури відносно центру, а також орієнтацією поверхні в просторі.

Переміщення умовної точки реалізовано за допомогою введення з клавіатури використовуючи клавіші WASD. Кожна з клавіш відповідає за один напрямок та переміщує умовну точку на визначений крок. Переміщення здійснюється до визначеної межі.

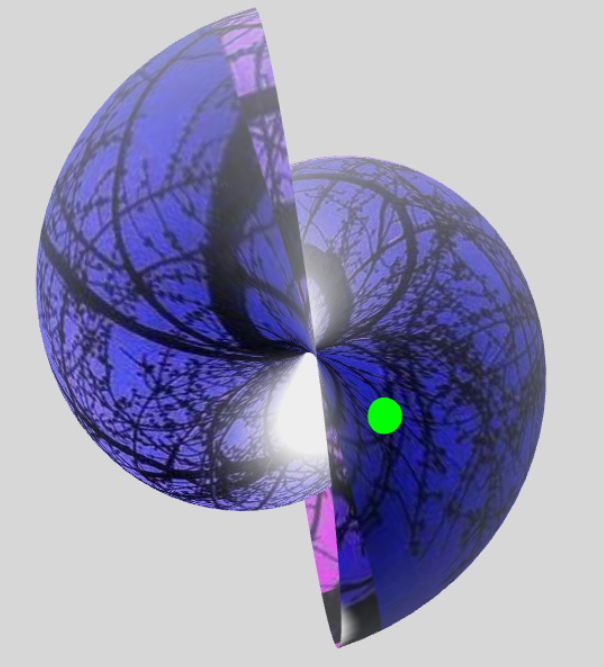
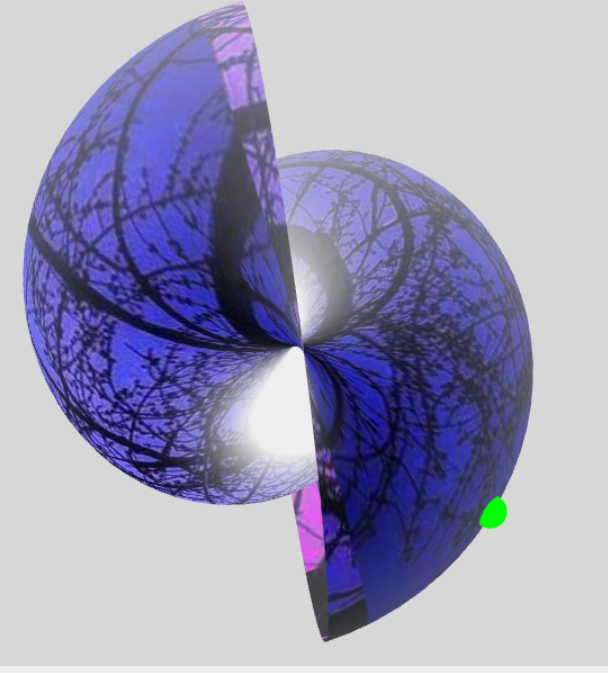
 

Рис. 5 – Переміщення умовної точки.

За допомогою повзунка можна змінювати масштабування відносно умовної точки на поверхні. Можна помітити, що при переміщенні точки зміщується і текстура, адже операція відбувається відносно іншої точки на поверхні , яка в свою чергу відповідає іншій текстурній координаті. Результати можна побачити на рисунках 6, 7.

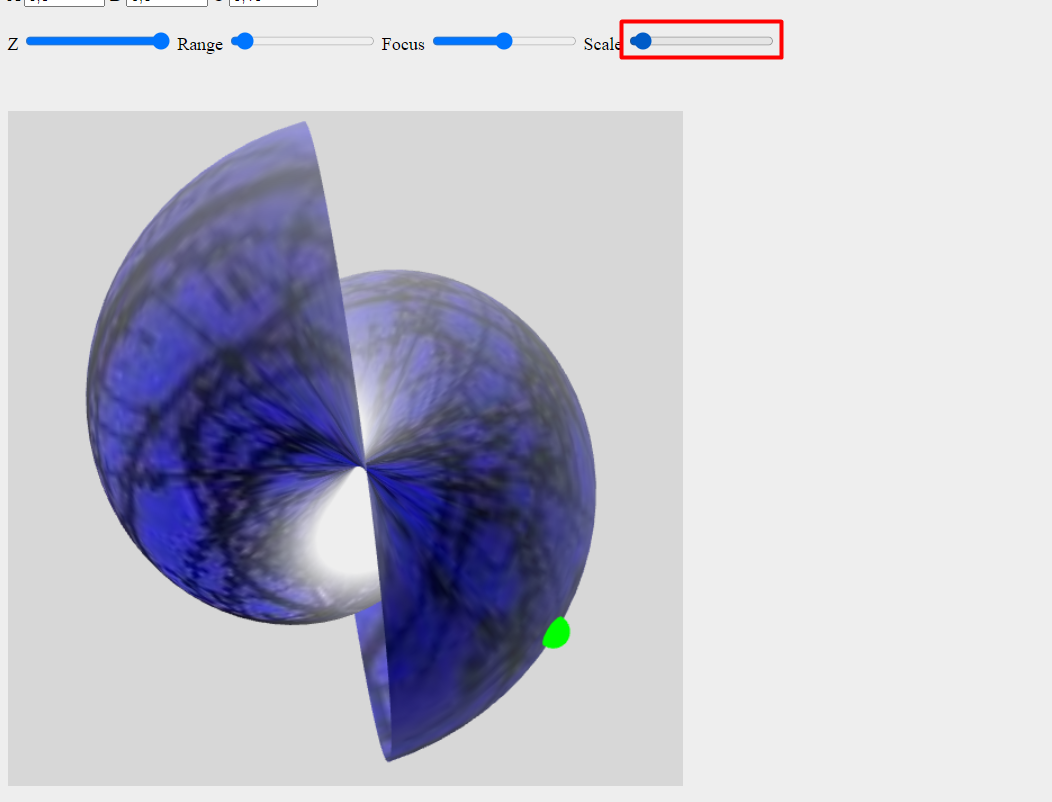


Рисунок 6 – Трансформація текстури

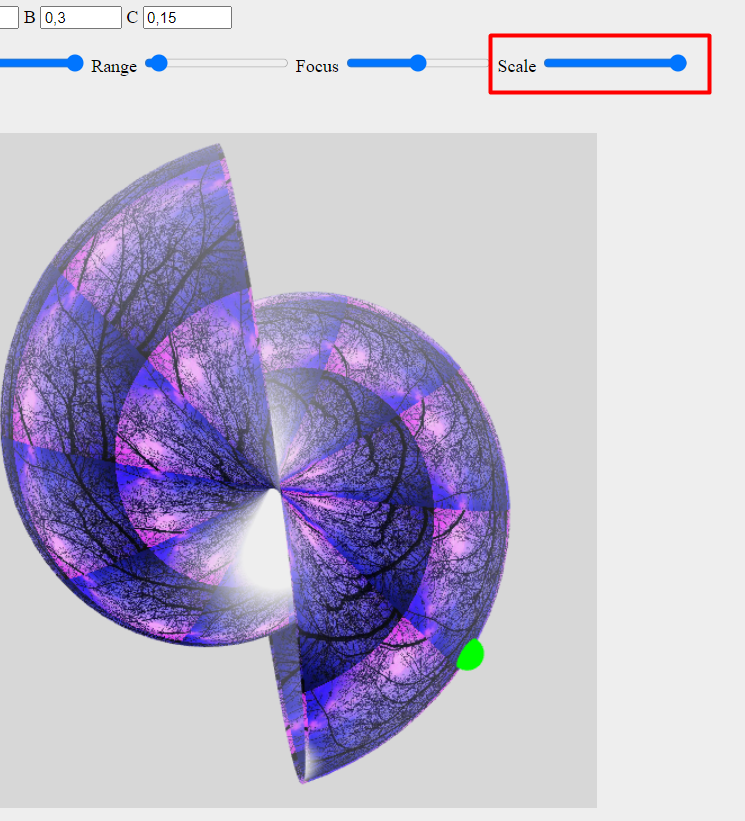


Рисунок 7 – Трансформація текстури

**Лістинг частини програми**

function addListeners() {

    let constants = ['a', 'b', 'c'];

    window.addEventListener('DOMContentLoaded', () => {

        constants.forEach((item) => {

            document.getElementById(item).addEventListener("change", () => {

                updateConstants();

                surface.BufferData(CreateSurfaceData().vertexList);

                draw();

            });

        });

    })

    window.addEventListener('keydown', (e) => {

        switch(e.code) {

            case "KeyA":

                moveV = Math.max(moveV - 0.01, 0);

                break;

            case "KeyD":

                moveV = Math.min(moveV + 0.01, 1);

                break;

            case "KeyW":

                moveU = Math.min(moveU + 0.01, 1);

                break;

            case "KeyS":

                moveU = Math.max(moveU - 0.01, 0);

                break;

            default:

                break;

        }

    })

}

addListeners();

function updateConstants() {

    A = document.getElementById('a').value;

    B = document.getElementById('b').value;

    C = document.getElementById('c').value;

}

function processSurfaceEquations(u, v) {

    const x = A \* deg2rad(u) \* Math.sin(deg2rad(u)) \* Math.cos(deg2rad(v));

    const y = B \* deg2rad(u) \* Math.cos(deg2rad(u)) \* Math.cos(deg2rad(v));

    const z = -C \* deg2rad(u) \* Math.sin(deg2rad(v));

    return { x, y, z };

}

function getNormal(j, i) {

    const innerStep = 0.0001;

    const { x, y, z } = processSurfaceEquations(j, i);

    const { x: x1, y: y1, z: z1 } = processSurfaceEquations(j + innerStep, i);

    const { x: x2, y: y2, z: z2 } = processSurfaceEquations(j, i + innerStep);

    const deltaU = [x - x1 / innerStep, y - y1 / innerStep, z - z1 / innerStep];

    const deltaV = [x - x2 / innerStep, y - y2 / innerStep, z - z2 / innerStep];

    const normal = m4.normalize(m4.cross(deltaU, deltaV));

    return normal;

}

function loadAndBindTexture() {

    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.setAttribute('Access-Control-Allow-Origin', '\*')

    image.crossOrigin = 'crossorigin'

    image.src = "https://raw.githubusercontent.com/zhura0well/geometric\_visualisation/CGW/assets/test.jpg";

    image.onload = () => {

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

        gl.texImage2D(

            gl.TEXTURE\_2D,

            0,

            gl.RGBA,

            gl.RGBA,

            gl.UNSIGNED\_BYTE,

            image

        );

        draw();

    }

}