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 координатами текстур”

19 варіант

Виконав:
Студент групи ТР-42мп
Юрченко Василь Олександрович

Київ - 2024

РОЗДІЛ 1. Опис завдання

1. Реалізувати масштабування або обертання текстури навколо точки, яка визначається користувачем, використовуючи текстурування з контрольного завдання. Масштабування текстурних координат (для непарних варіантів).
2. Має бути можливість переміщати задану точку поверхнею у просторі (u,v) за допомогою клавіатури. Наприклад, клавіші **A** і **D** переміщують точку вздовж параметра *u*, а клавіші **W** і **S** — вздовж параметра *v*.
3. Підготувати цифровий звіт, який містить:
    a. титульну сторінку,
    b. розділ, що описує завдання (1 сторінка)
    c. розділ з теоретичними основами (2 сторінки)
    d. розділ з деталями реалізації (2 сторінки)
    e. інструкцію для користувача з прикладами та скриншотами (2 сторінки)
    f. зразок вихідного коду (2 сторінки).
4. Поділитися посиланням на ваш GitHub-репозиторій через відповідну форму. У репозиторії має бути розміщена стаття, яка представляє виконане завдання. Практичне завдання має знаходитися в гілці з назвою CGW.

РОЗДІЛ 2. Опис теоретичної частини

Текстурування — це процес нанесення зображення (текстури) на поверхню тривимірного об’єкта для підвищення візуальної реалістичності. У WebGL текстури визначаються у просторі координат (u, v), де *u* та *v* є нормалізованими значеннями у діапазоні [0, 1].

Ключові етапи текстурування включають у себе створення та завантаження текстури, зв’язування текстури з шейдерною програмою, налаштування відображення текстури на геометрії.

Операції з текстурними координатами включають масштабування, обертання, зміщення (translation) та віддзеркалення текстури.

- **Масштабування** текстури:

  Масштабування змінює розмір текстури шляхом множення текстурних координат *u* та *v* на заданий коефіцієнт:

$$u' = S_x \cdot (u - u_c) + u_c, \quad v' = S_y \cdot (v - v_c) + v_c$$

- **Обертання** текстури:

  Обертання реалізується шляхом застосування матриці обертання до текстурних координат:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} u - u_c \\ v - v_c \end{bmatrix} + \begin{bmatrix} u_c \\ v_c \end{bmatrix}$$

WebGL: Основи та Застосування

WebGL (*Web Graphics Library*) — це технологія для рендерингу 2D- та 3D-графіки у веб-браузерах. Вона базується на OpenGL ES 2.0 і дозволяє виконувати складні графічні операції, використовуючи апаратне прискорення графічного процесора (GPU). WebGL інтегрується з JavaScript, що дозволяє створювати інтерактивні графічні додатки без використання додаткових плагінів.

Для початку роботи необхідно отримати контекст WebGL із HTML-елемента <canvas>.

Шейдери — це невеликі програми, які виконуються на графічному процесорі. Вони поділяються на два типи:

- **Vertex Shader** (Вершинний шейдер): обробляє вершини геометрії.
- **Fragment Shader** (Фрагментний шейдер): відповідає за обчислення кольору кожного пікселя.

Дані для геометрії та текстур завантажуються у буфери WebGL. Це включає координати вершин, кольори та текстурні координати.

Інтерфейс програмування WebGL містить функції для налаштування шейдерів, передачі даних у GPU, керування текстурами та виконання рендерингу. Текстурування у WebGL дозволяє наносити зображення (текстури) на поверхню тривимірного об’єкта. Працює у будь-якому сучасному веб-браузері. Дозволяє використовувати потужність GPU для високопродуктивної графіки. Забезпечує швидкий відгук для графічних додатків, таких як ігри, симуляції та моделювання.

WebGL дозволяє реалізувати складні графічні операції, зберігаючи високу продуктивність навіть у браузерному середовищі.

Основні кроки реалізації:

1. Передача текстурних координат до шейдера:
   Вершинний шейдер передає координати (u, v) до фрагментного шейдера.
2. Реалізація трансформацій у фрагментному шейдері:
   Для виконання операцій з текстурними координатами у фрагментному шейдері використовуються матриці та обчислення у реальному часі.
3. Зміна центру трансформацій за допомогою клавіатури:
   Управління здійснюється через обробку подій клавіатури у JavaScript, значення параметрів передаються у шейдери як uniform-параметри.

РОЗДІЛ 3. Опис аспектів розробки

Під час виконання практичного завдання розроблено програму, яка відображає поверхню у вигляді суцільних трикутників.

Рисунок 3.1 - *Візуалізація поверхні*

Для подальшого виконання розрахунково-графічної роботи було обрано відповідне зображення. Щоб забезпечити сумісність із більшістю операційних систем та браузерів, використано зображення розміром 512×512 пікселів.

Рисунок 3.2 - *Обране зображення текстури*

На поверхню було накладено текстуру. Для цього створено буфер текстурних координат, де кожна координата відповідає відповідному елементу масиву буфера вершин. Згідно з умовами завдання, текстура повинна масштабуватися, тому створено відповідний uniform-параметр, що задає кут обертання. У програмі шейдера для визначення кольору пікселя на текстурованій поверхні використовується функція texture2D(). Ця функція приймає два аргументи: перший — об'єкт класу sampler2D, який зберігає дані зображення, другий — текстурні координати.

Рисунок 3.3 - Накладена текстура на поверхню

Для відображення точки, відносно якої здійснюється трансформація текстури, створено новий об'єкт класу Model. Ця точка графічно представлена у вигляді сфери, яка розташована на поверхні.

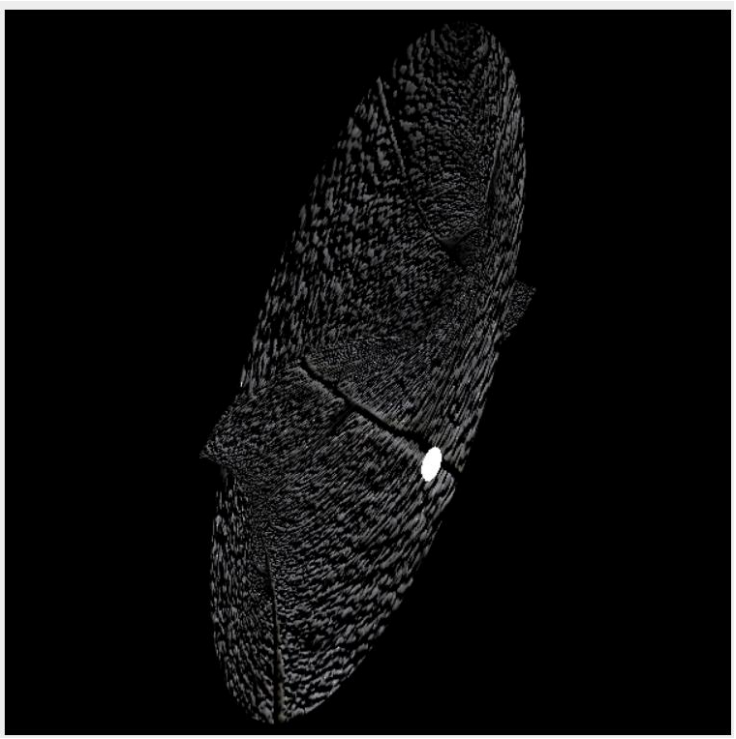

Рисунок 3.4 - Точка трансформації на текстурованій поверхні

Згідно з варіантом, було реалізовано масштабування текстури відносно визначеної точки на поверхні.

РОЗДІЛ 4. Інструкція для користувача зі скріншотами

Фігуру можна обертати, затиснувши ліву клавішу миші та переміщуючи її в напрямку бажаного обертання.

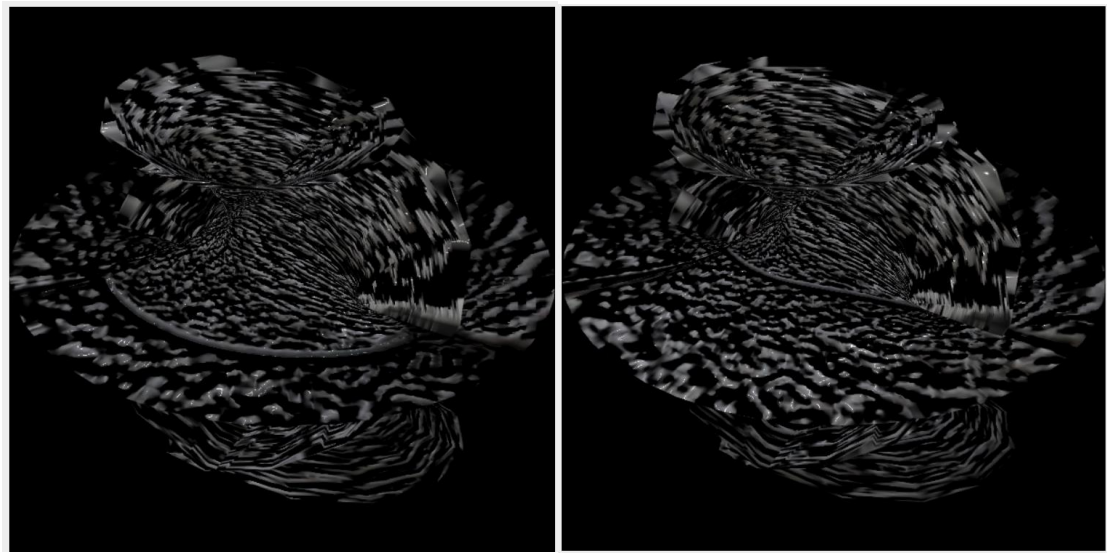

Рисунок 4.1 - Вигляд фігури до та після масштабування

Переміщення точки, відносно якої здійснюється обертання, можна здійснювати за допомогою клавіш WASD. Кожне натискання переміщає точку на поверхні на певний крок, і переміщення продовжується до заданої межі.

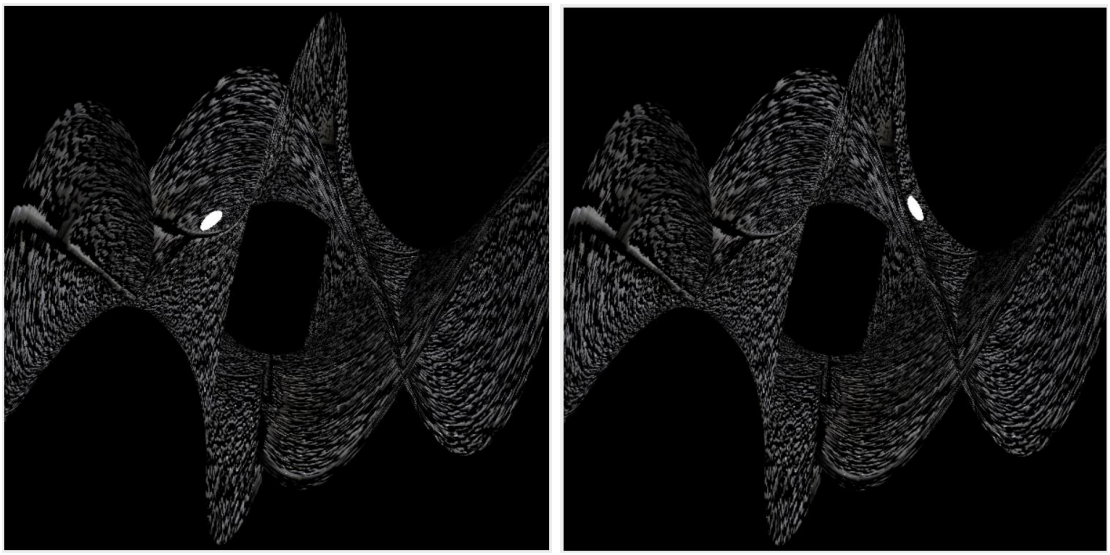

Рисунок 4.2 - Демонстрація переміщення точки відносно статичної фігури

Зміна кута обертання відносно умовної точки на поверхні здійснюється за допомогою повзунка. Якщо значення кута обертання не дорівнює нулю, при переміщенні точки можна помітити, що зміщується й текстура. Це пояснюється тим, що обертання відбувається відносно іншої точки на поверхні, яка відповідає іншій текстурній координаті.

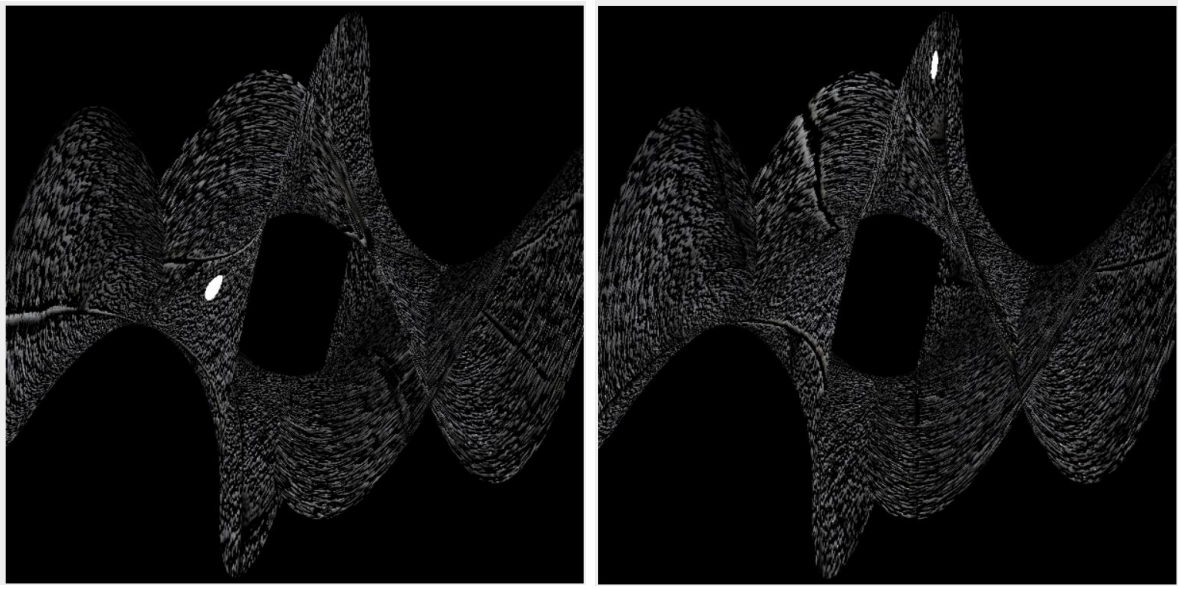

Рисунок 4.3 - Демонстрація зміни кута обертання текстури

При перезавантаженні сторінки обертання фігури та масштабування скидаються до значень за замовчуванням. Положення точки на поверхні також відновлюється до значення за замовчуванням.

1. Зразок коду програми

```
1.  function draw() {
2.          gl.clearColor(0, 0, 0, 1);
3.          gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
4.
5.          let projection = m4.orthographic(-2, 2, -2, 2, 8, 12);
6.
7.          let modelView = spaceball.getViewMatrix();
8.
9.          let rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0.7);
10.         let translateToPointZero = m4.translation(0, 0, -10);
11.
12.         let matAccum0 = m4.multiply(rotateToPointZero, modelView);
13.         let matAccum1 = m4.multiply(translateToPointZero, matAccum0);
14.
15.         let modelViewProjection = m4.multiply(projection, matAccum1);
16.
17.         let inversion = m4.inverse(modelViewProjection);
18.         let transposedModel = m4.transpose(inversion);
19.
20.         gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false,
modelViewProjection);
21.         gl.uniformMatrix4fv(shProgram.iModelMatrixNormal, false,
transposedModel);
22.         gl.uniform3fv(shProgram.iLightPosition, [0.0, 1.0, 0.0]);
23.         gl.uniform1f(shProgram.iS,
parseFloat(document.getElementById("paramS").value));
24.         gl.uniform2fv(shProgram.iT, p);
25.         const v = equations(
26.             map(p[0], 0, 1, 0.25,
parseFloat(document.getElementById("paramR").value)),
27.             map(p[1], 0, 1, 0, 2 * Math.PI),
28.         );
29.         gl.uniform3fv(shProgram.iCP, [v.x, v.y, v.z]);
30.
31.         surface.Draw();
32.     }
```