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 2

Виконав студент групи ТР-32мп
Бортовський Олег Олексійович

Київ - 2024

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

2. Теорія

WebGL

WebGL - це інтерфейс програмування додатків на JavaScript, який використовується для створення веб-браузерами інтерактивної 2D- та 3D-графіки. Для відображення текстур, їх масштабування, обертання та інтерактивного переміщення точок, WebGL API містить функції для обробки шейдерів, текстур і матричних перетворень.

1. Шейдери використовуються для визначення, як обробляються вершини та фрагменти, надаючи можливість включення текстурних координат та різних перетворень.
2. Створення та пов'язування текстурних об'єктів з конкретними текстурними одиницями для використання у шейдерах.
3. Застосування матричних перетворень для управління положенням, масштабуванням і обертанням 3D-моделі.

Шейдери WebGL

В WebGL, роль шейдерів у візуалізації визначається як ключова, забезпечуючи гнучкий та програмований метод визначення вигляду 3D-графіки. Цей інтерфейс використовує два основних типи шейдерів: вершинні та фрагментні.

Вершинні шейдери відповідають за обробку кожної вершини 3D-моделі перед її відображенням на екрані. Ці шейдери обробляють атрибути вершин, такі як положення, колір і координати текстури, для створення вихідних даних. Застосування трансформацій до позицій вершин, таких як переміщення, обертання та масштабування, є поширеним використанням вершинних шейдерів. Результати вершинного шейдера включають перетворену позицію вершини та інтерпольовані значення для передачі фрагментному шейдеру.

Фрагментні шейдери, також відомі як піксельні шейдери, працюють з кожним пікселем, що буде відображений на екрані. Вони отримують інтерпольовані дані від вершинного шейдера, такі як координати кольору та текстури, та інші дані про освітлення. Основне завдання фрагментного шейдера - визначення кольору кожного пікселя, включаючи вибірку текстур, обчислення освітлення та інші ефекти.

Обидва типи шейдерів, вершинні та фрагментні, написані мовою GLSL (OpenGL Shading Language) і підлягають компіляції перед використанням. Після компіляції вони об'єднуються в програму шейдера, яка повинна бути зв'язана перед відтворенням, щоб WebGL міг використовувати їх у конвеєрі відтворення.

Uniforms - це константні значення для всіх вершин або фрагментів примітиву, які дозволяють передавати зовнішні дані, такі як матриці трансформації чи інформацію про глобальне освітлення, шейдерам.

Attributes - це дані для кожної вершини, які відрізняються між ними, і використовуються для передачі інформації, такої як положення вершин, нормалі та координати текстури. Координати текстури, зазвичай передані як атрибути до вершинного шейдера, інтерполюються для використання у фрагментному шейдері, де вони використовуються для вибірки кольорів з текстур та визначення остаточного кольору кожного пікселя.

Накладання текстур. Текстурні координати

Відображення текстур - це метод у комп'ютерній графіці, який дозволяє створювати реалістичні поверхні, застосовуючи зображення або текстури до 3D-моделей. У веб-графіці, зокрема у WebGL, відображення текстур включає асоціювання кожної вершини 3D-об'єкта з координатами текстури (u, v). Ці координати використовуються для вибірки кольорів із текстурного зображення, щоб поліпшити візуальний вигляд об'єкта та надати детальну інформацію про його поверхню.

Координати текстури (u, v) є ключовими параметрами, які визначають спосіб накладення текстури на поверхню. Значення цих координат знаходяться у діапазоні від 0 до 1 і використовуються для визначення конкретних точок на текстурному зображенні. У контексті WebGL ці координати пов'язані з кожною вершиною 3D-моделі та інтерполюються по всій поверхні під час візуалізації. Це забезпечує точне охоплення текстури моделлю для забезпечення правильного візуального представлення.

3. Деталі розробки

За варіантом мені було надано Corrugated Sphere. При виконанні практичного завдання №2 було розроблено програму, що виводить поверхню у вигляді суцільних трикутників.

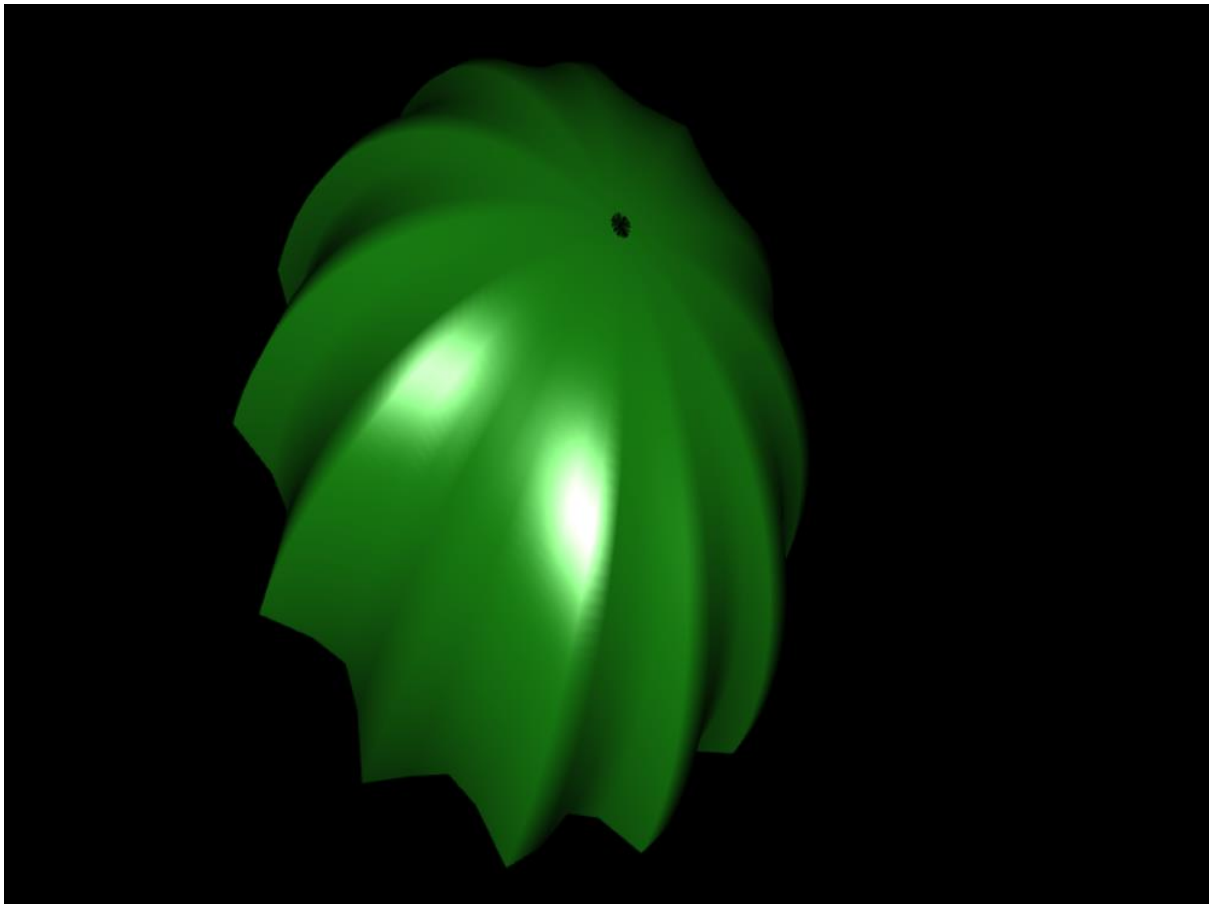

Рисунок 1 - Corrugated Sphere

Я обрав наступне зображення для подальшого виконання розрахунково-графічної роботи. Для підтримки на більшій кількості операційних систем та браузерів було обрано зображення розміром 512x512 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 обертатися, тому було створено відповідний uniform, який визначатиме кут обертання.

Рисунок 3 – Текстура накладена на поверхню

Було створено новий об’єкт класу Model для відображення точки відносно якої буде здійснюватись трансформація текстури. Об’єкт графічно відображається як сфера. Сфера перебуває на поверхні.

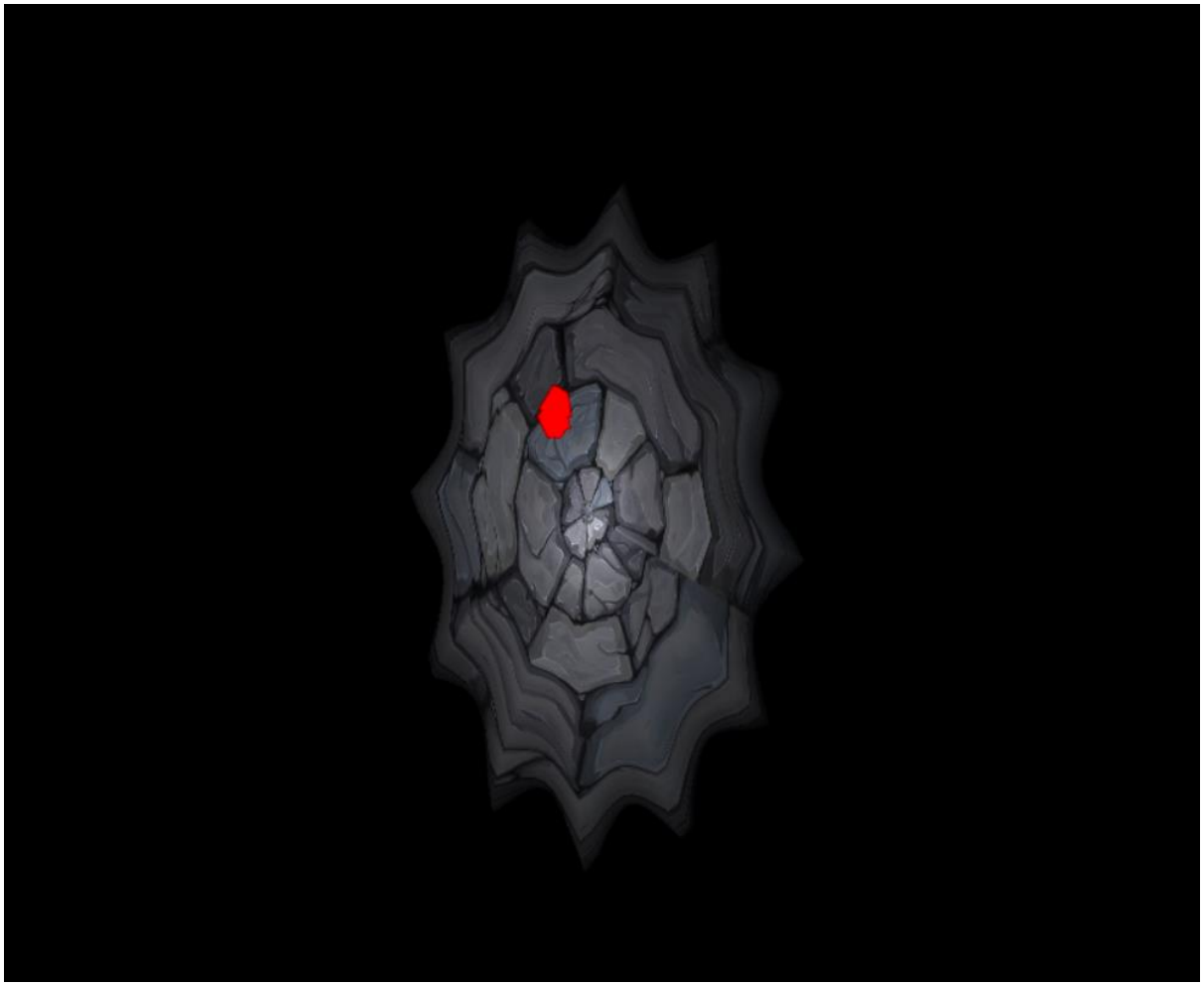

Рисунок 4 – Точка на поверхні

Згідно варіанту було імплементовано обертання текстури відносно точки на поверхні.

4. 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.

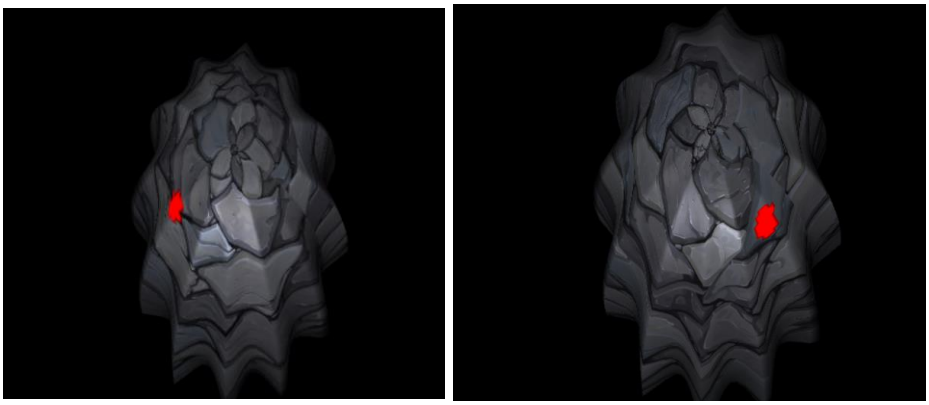

Рисунок 6 - Демонстрація переміщення точки
відносно нерухомої фігури

Можна помітити, що при переміщенні точки зміщується і текстура, адже обертання відбувається відносно іншої точки на поверхні , яка в свою чергу відповідає іншій текстурній координаті.

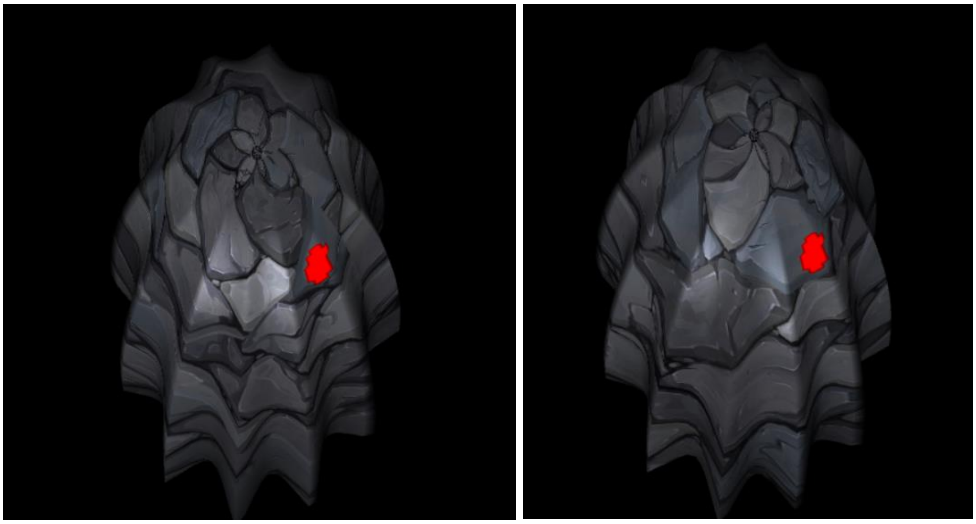

Рисунок 7 - Демонстрація зміни кута обертання текстури

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

5. Код програми

Код функцій астини програми на javascript

```
function draw() {
    gl.clearColor(0, 0, 0, 1);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    let projectionMatrix = m4.perspective(scale, 2, 1, 40);
    const viewMatrix = m4.lookAt(CameraPosition, WorldOrigin, [0, 1, 0]);
    const camRotation = m4.axisRotation([0, 1, 0], 179);
    const projectionViewMatrix = m4.multiply(projectionMatrix,
m4.multiply(viewMatrix, camRotation));

    if(ShowPath){
        segmentProgram.Use();
        segment.Draw(projectionViewMatrix);
    }

    shProgram.Use()
    surface.Draw(projectionViewMatrix);
}

function GetDirLightDirection(){
    let test = m4.scaleVector(m4.normalize(LightPosition), -1);
    return test;
}
```

Код функції частни програми на GLSL

```
// Vertex shader
const vertexShaderSource = `
attribute vec3 position;
attribute vec2 textureCoord;
attribute vec3 normal;
uniform vec3 matAmbientColor;
uniform vec3 matDiffuseColor;
uniform vec3 matSpecularColor;
uniform float matShininess;
```

```
uniform vec3 lsAmbientColor;
uniform vec3 lsDiffuseColor;
uniform vec3 lsSpecularColor;
uniform vec3 LightDirection;
uniform vec3 CamWorldPosition;
uniform mat4 WorldInverseTranspose;
uniform mat4 ModelViewProjectionMatrix;
uniform vec2 rotationPoint;
uniform float rotationValue;
uniform vec3 pointVizualizationPosition;
varying vec4 color;
varying vec2 v_texcoord;
varying vec4 pointColorSaturation;

void main() {
   vec3 normalInterp = (WorldInverseTranspose * vec4(normalize(normal),
0.0)).xyz;
   vec3 N = normalize(normalInterp);
   vec3 L = normalize(LightDirection);
   gl_Position = ModelViewProjectionMatrix * vec4(position,1.0);

   vec3 reflectLighDir = normalize(reflect(-L, N));
   vec3 camViewDir = normalize(CamWorldPosition - position);

   pointColorSaturation = vec4(0.0, 0.0, 0.0, 0.0);
   if(distance(pointVizualizationPosition, position) < 0.07){
      pointColorSaturation.r = 1.0;
      pointColorSaturation.g = -1.0;
      pointColorSaturation.b = -1.0;
   }
   vec3 ambient = matAmbientColor * lsAmbientColor;
   vec3 diffuse = matDiffuseColor * lsDiffuseColor * max(0.0, dot(N, L));
   vec3 specular = matSpecularColor * lsSpecularColor * pow(max(0.0,
dot(camViewDir, reflectLighDir)), matShininess);
   color = vec4(ambient + diffuse + specular, 1.0);
   v_texcoord = textureCoord;
}`;
```