Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Інститут атомної та теплової енергетики
Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота

З дисципліни «Візуалізація графічної та геометричної інформації»
Варіант 2

Виконав: Булига Микола
Студент групи ТР-31мп

Київ 2023

Завдання

Тема роботи: Операціїї над тектурними координатами

Вимоги:

- Накласти текступу на поверхню отриману в результаті виконання лабораторної роботи №2.
- Імплементувати масштабування або обертання текстури(текстурних координат) згідно з варіантом: непарні - масштабування, парні - обертання.
- Запровадити можливість переміщення точки відносно якої відбувається трансформація текстури по поверхні за рахунок зміни параметрів в просторі текстури. Наприклад, клавіші A та D для переміщення по осі абсцис, змінюючи параметр u текстури, а клавіші W та S по осі ординат, змінюючи параметр v.

Теоретичні відомості

Текстурування - це метод додавання реалістичності до комп'ютерної графіки. Зображення відображається на поверхні, яка генерується в сцені, як наклейка, наклеєна на плоску поверхню. Це зменшує обсяг обчислень, необхідних для створення форм і текстур у сцені. Наприклад, можна згенерувати сферу і нанести на неї текстуру обличчя, щоб усунути необхідність обробки форми носа та очей.

Оскільки відеокарти стають потужнішими, теоретично, відображення текстур для освітлення стає менш необхідним, і на зміну йому приходить відображення рельєфу або збільшення кількості полігонів. Однак на практиці останнім часом спостерігається тенденція до більших і різноманітніших текстурних зображень, а також дедалі складніших способів поєднання кількох текстур для різних аспектів одного об'єкта. (Це є більш значущим у графіці реального часу, де кількість текстур, які можуть відображатися одночасно, є функцією доступної графічної пам'яті).

Спосіб обчислення результуючих пікселів на екрані з текселів (пікселів текстури) регулюється фільтрацією текстури. Найшвидшим методом є використання рівно одного текселя для кожного пікселя, але існують і більш складні методи.

Для процесу 3D-текстурування, необхідно спочатку розгорнути модель, що, по суті, те саме, що розгортання 3D-сітки. Коли художники-фактуристи отримають готові моделі від відділу 3D-моделювання, вони створять UV-карту для кожного 3D-об’єкта. UV-карта — це плоске зображення поверхні 3D-моделі, яке використовується для швидкого накладання текстур. Прямо пов’язуючи 2D-зображення (текстуру) з вершинами багатокутника, UV-відображення може допомогти обернути 2D-зображення (текстуру) навколо 3D-об’єкта, а згенеровану карту можна використовувати безпосередньо в процесі текстурування та затінення.

Програмні системи 3D мають кілька інструментів або підходів для розгортання 3D-моделей. Що стосується створення UV-карт, то це питання особистих уподобань. Якщо ви не плануєте використовувати процедурні текстури, у більшості випадків вам слід розгорнути вашу 3D-модель у компонент текстурування. Це текстури, створені за допомогою математичних методів (процесів), а не безпосередньо записаних даних у 2D або 3D.

В якості оптимізації можна перетворити деталі зі складної моделі з високою роздільною здатністю або дорогого процесу (наприклад, глобального освітлення) в текстуру поверхні (можливо, на моделі з низькою роздільною здатністю). Запікання також відоме як рендер-мапінг. Ця техніка найчастіше використовується для мап освітлення, але може також використовуватися для створення нормальних мап і мап переміщень. Деякі комп'ютерні ігри (наприклад, Messiah) використовують цю техніку. Оригінальний програмний рушій Quake використовував запікання на льоту для поєднання світлових і кольорових карт ("поверхневе кешування").

Запікання може використовуватися як форма генерації рівня деталізації, коли складна сцена з багатьма різними елементами та матеріалами може бути апроксимована одним елементом з однією текстурою, яка потім алгоритмічно зменшується для зниження вартості рендерингу та зменшення кількості відмов від рендерингу. Він також використовується для отримання високодеталізованих моделей з програмного забезпечення для 3D-скульптування та сканування хмари точок і апроксимації їх сітками, більш придатними для візуалізації в реальному часі.

Виконання завдання

В ході другої лабораторної роботи було створено поверхню під назвою «Corrugated Sphere». Отриману поверхню з освітленням можнапобачити на рисунку 3.1.

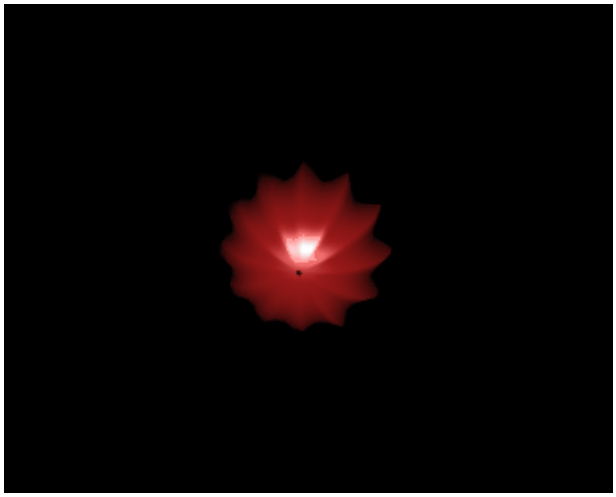

Рис. 3.1 «Corrugated Sphere» з освітленням

Текстура була завантажена як картинка з інтернету формату «jpg». Вона була завантажена на github, щоб в подальшому використовувати посилання на неї і не стикатися з проблемою Cross-Origin Resource Sharing policy.

В графічному редакторі було налаштувано розмір картинки так, щоб ширина і висота були рівні, а також, аби сторона мала розмір 2^n в пікселях.

З метою накладання текстури на поверхню, в першу чергу було створено декілька змінних в коді шейдера. Після чого були створення

посилання на них в коді програми. Були також створені функції для генерації даних текстури.

Обрану картинку можна побачити на рисунку 3.2.

Рис. 3.2 Обрана текстура

Поверхню з накладеною текстурою можна побачити на рисунку 3.3.

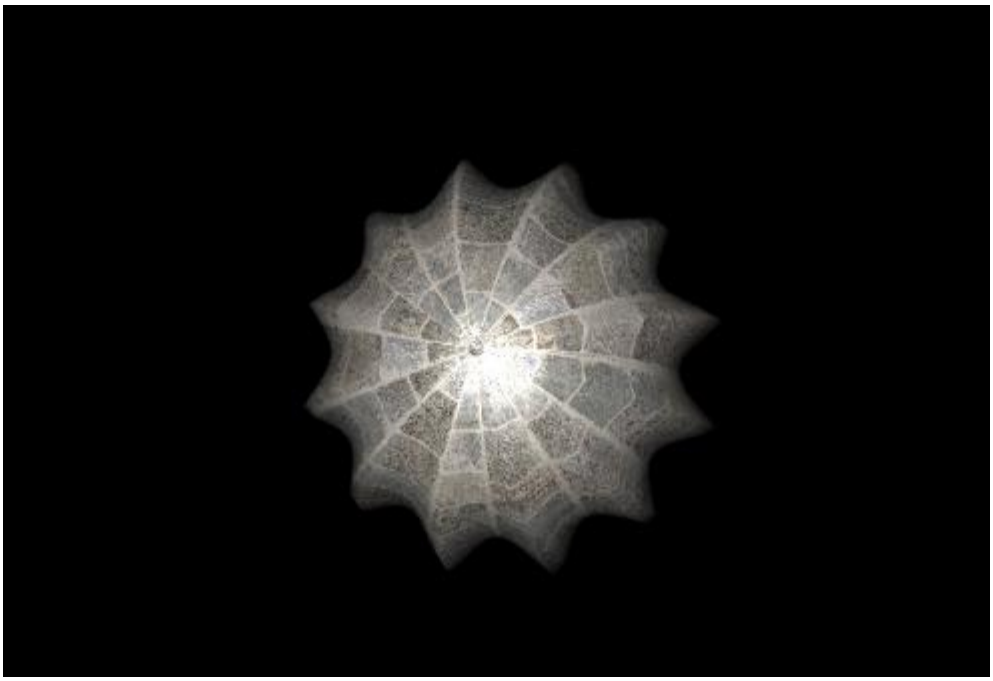

Рис. 3.3 «Corrugated Sphere» з накладеною текстурою

За допомогою шейдера була створена динамічна точка. Поверхню з точкою можна побачити на рис 3.4

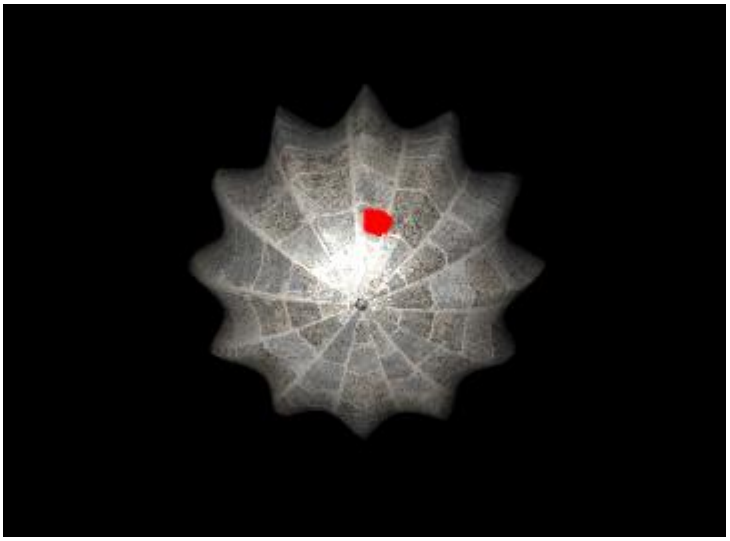

Рис. 3.4 «Corrugated Sphere» з накладеною текстурою та

Для роботи з текстурою було створено ще кілька змінних в коді шейдера:

обертання текстури, розташування умовної точки в (u,v) координатах, змінну для розташування сфери на відповідне місце поверхні в 3д-просторі.

Вказівки користувачу

Була додана можливість для користувача, керувати переміщенням умовних точок по поверхні, поворотом текстур щодо умовних точок і орієнтацією поверхні в просторі. Останні два пункти виконуються таким же чином.

Переміщення умовної точки реалізовано за допомогою введення з клавіатури(рисунок 4.1): клавіші W та S здійснюють переміщення точки за параметром v в додатньому та від’ємному напрамках відповідно, клавіші A та D здійснюють переміщення точки за параметром u у від’ємному та додатньому напрямках відповідно.

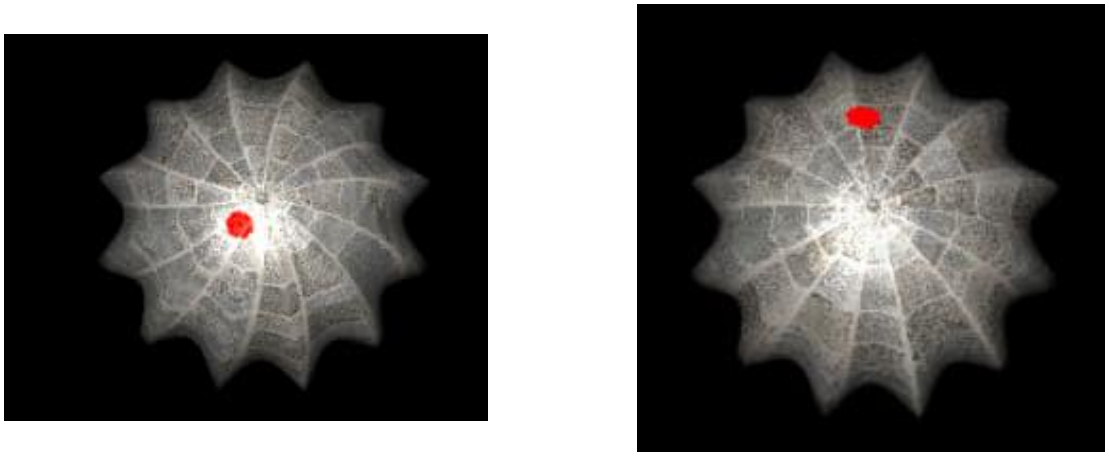

Рис. 4.1. Переміщення умовної точки

Орієнтація поверхні в просторі здійснюється за допомогою миші. На рисунку 4.2 можна помітити що точка та текстура залишились на одному і тому самому місці відносно поверхні. Змінилась лише орієнтації поверхні в просторі.

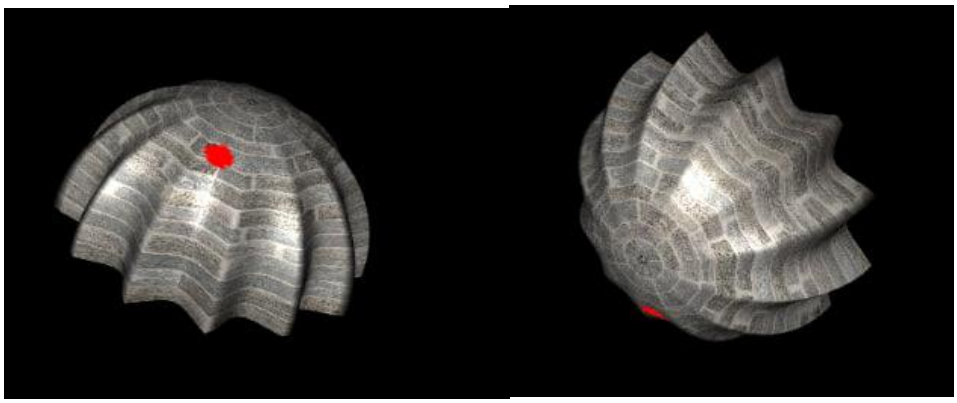

Рис. 4.2. Зміна орієнтація поверхні в просторі

Для зміни повороту текстури на сторінку доданий слайдер зображений на рис. 4.3.

Рис. 4.3. Слайдер зміни повороту текстури

Поворот текстури 45 градусів та 90 градусів зображено на рисунку 4.4

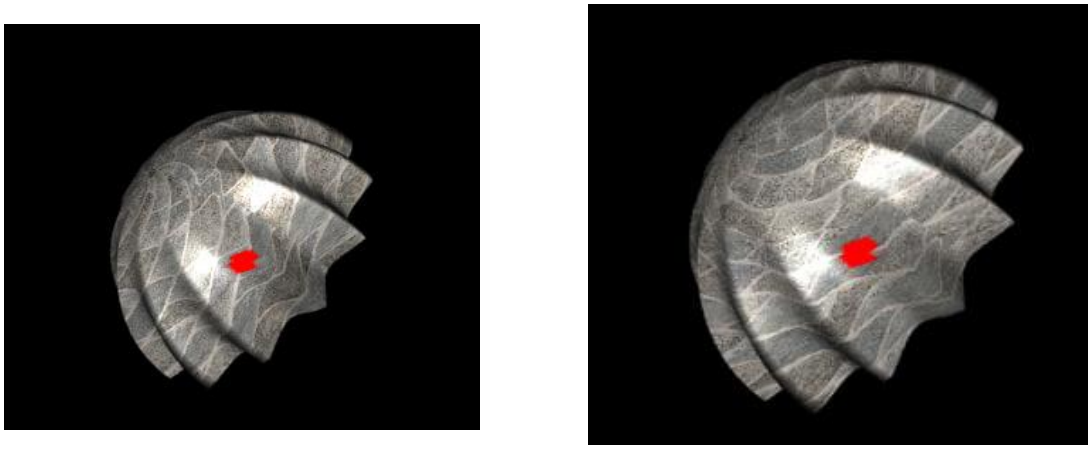

Рис. 4.4. Поворот текстури

Висновок

В даній розрахунковій роботі ми дослідили, що таке текстурування об'єкту, а також вивчили, що таке розгортка та UV-mapping. Було реалізовано обертання текстури навколо визначеної користувачем точки. Також є можливість переміщати точку вздовж поверхні. Матеріал засвоєний.

Код шейдерів:

```
// Vertex shader
const vertexShaderSource = `
attribute vec3 vertex;
attribute vec2 textureCoord;
attribute vec3 normal;

uniform mat4 WorldInverseTranspose;
uniform mat4 ModelViewProjectionMatrix;

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 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(vertex,1.0);

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

    pointColorSaturation = vec4(0.0, 0.0, 0.0, 0.0);
    if(distance(pointVizualizationPosition, vertex) < 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;
}`;


// Fragment shader
const fragmentShaderSource = `
#ifdef GL_FRAGMENT_PRECISION_HIGH
   precision highp float;
#else
   precision mediump float;
#endif

varying vec4 color;
varying vec4 pointColorSaturation;
uniform vec2 rotationPoint;
uniform float rotationValue;
uniform sampler2D texture;
varying vec2 v_texcoord;

mat3 getRotationMatrix(float angle) {
    float s = sin(angle);
    float c = cos(angle);
    return mat3(
        vec3(c, -s, 0.0),
        vec3(s, c, 0.0),
        vec3(0.0, 0.0, 1.0)
    );
}

void main() {
    vec3 texCoordHomogeneous = vec3(v_texcoord, 1.0);

    mat3 t1Matrix = mat3(
        vec3(1.0, 0.0, -rotationPoint.x),
        vec3(0.0, 1.0, -rotationPoint.y),
        vec3(0.0, 0.0, 1.0)
    );
```

```
    mat3 rotationMatrix = getRotationMatrix(rotationValue);
    vec3 rotatedTexCoord = texCoordHomogeneous * t1Matrix *
rotationMatrix;

    mat3 t2Matrix = mat3(
        vec3(1.0, 0.0, rotationPoint.x),
        vec3(0.0, 1.0, rotationPoint.y),
        vec3(0.0, 0.0, 1.0)
    );

    gl_FragColor = texture2D(texture, (rotatedTexCoord * t2Matrix).xy) *
color + pointColorSaturation;
}`;
```