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 8

Виконав:
Студент 5-го курсу
Групи ТМ-31мп
Заступайло Максим

Перевірив:
Демчишин А.А.

Київ 2023

Завдання

Тема роботи: Операції над текстурними координатами

Вимоги:

- Накласти текстуру 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 представляє собою метод, який додає реалістичність комп'ютерній графіці. Зображення накладається на поверхню, яка створюється в сцені, аналогічно до того, як наклейка приклеюється до плоскої поверхні. Це спрощує обчислення, потрібні для створення форм та текстур в сцені. Наприклад, можна створити сферу і застосувати текстуру обличчя, щоб уникнути необхідності детально працювати над формою носа та очей.

З розвитком потужності відеокарт теоретично відображення текстур для освітлення стає менш важливим, і на його місце приходить відображення рельєфу або збільшення кількості полігонів. Проте на практиці відзначається тенденція до використання більших і різноманітніших текстурних зображень, а також складніших методів поєднання кількох текстур для різних аспектів об'єкта. Це особливо актуально у графіці реального часу, де кількість одночасно відображуваних текстур залежить від доступної графічної пам'яті.

Метод обчислення пікселів на екрані з текселів текстури контролюється фільтрацією текстури. Найшвидший спосіб - використовувати один тексель для кожного пікселя, але існують більш складні методи.

У процесі 3D-текстурування необхідно розгорнути модель, що суттєво збігається з розгортанням 3D-сітки. Художники-фактуристи, отримавши готові моделі від відділу 3D-моделювання, створюють UV-карту для кожного 3D-об'єкта. UV-карта - це плоске зображення поверхні 3D-моделі, яке використовується для швидкого накладання текстур. Це єфективно пов'язує 2D-зображення (текстуру) з вершинами багатокутника, допомагаючи обертати 2D-зображення (текстуру) навколо 3D-об'єкта.

Програмні системи 3D мають кілька інструментів для розгортання 3D-моделей. Щодо створення UV-карт, це питання особистих уподобань. Якщо немає планів використовувати процедурні текстури, то, як правило, розгортати 3D-модель у компоненті текстурування. Ці текстури створюються за допомогою математичних методів, а не безпосередньо записаних даних у 2D або 3D.

Оптимізацією може бути перетворення деталей складної моделі з високою роздільною здатністю або вартістю обробки (наприклад, глобальне освітлення) в текстуру поверхні на моделі з низькою роздільною здатністю. Це відоме як рендер-мапінг і часто використовується для мап освітлення, нормальних мап і мап переміщень. Запікання також використовується як спосіб генерації рівня деталізації, коли складну сцену можна апроксимувати одним елементом з однією текстурою, що потім алгоритмічно зменшується для зниження вартості рендерингу та кількості відмов. Застосовується для отримання високодеталізованих моделей з програмного забезпечення для 3D-скульптування та сканування хмари точок, а потім апроксимації їх сітками для ефективної візуалізації в реальному часі.

Виконання завдання

В ході другої лабораторної роботи було створено поверхню “Cornucopia”. Отриману поверхню з освітленням можна побачити на рисунку 3.1.

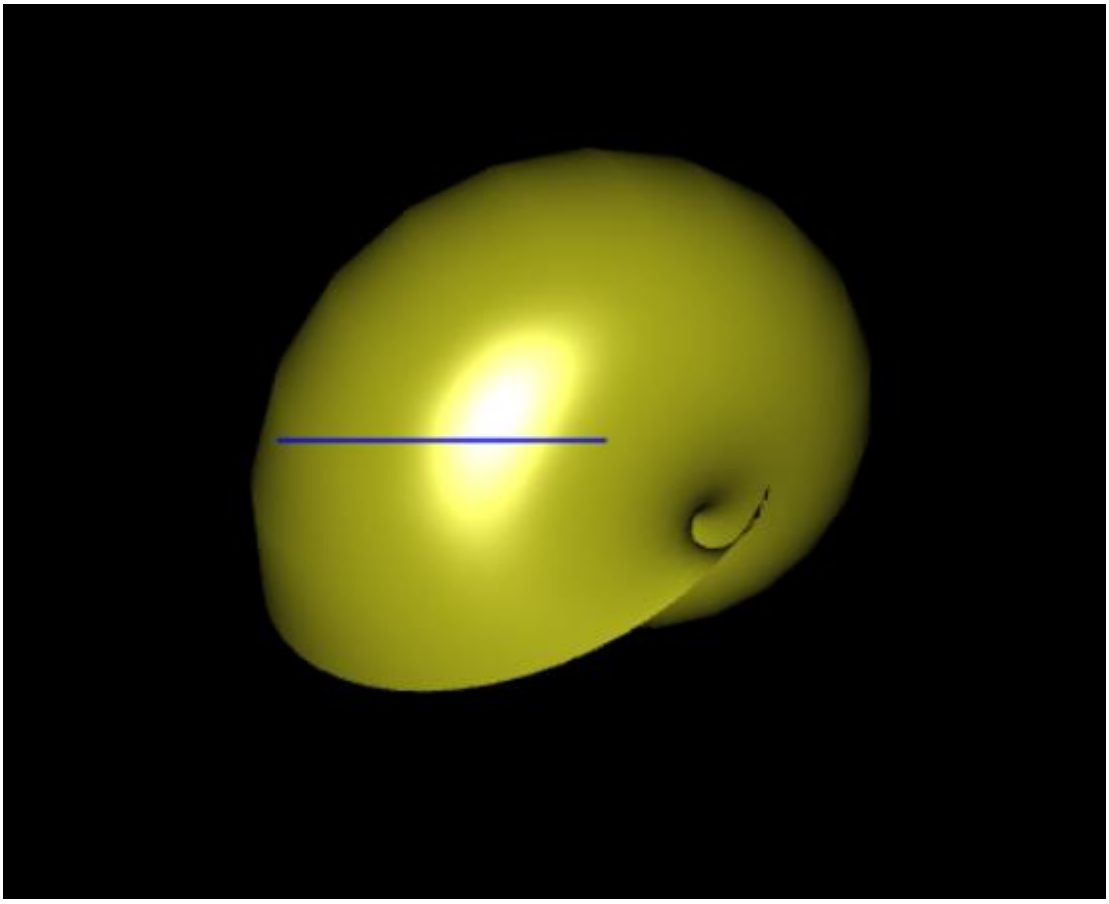

Рис 3.1 Поверхня «Cornucopia» з освітленням

Зображення було завантажено з Інтернету у форматі «jpg» і розміщено на GitHub для подальшого використання через посилання, уникнення проблем з політикою Cross-Origin Resource Sharing (CORS). У графічному редакторі були встановлені розміри зображення так, щоб ширина і висота були однакові, і щоб кожна сторона мала розмір 2^n в пікселях.

Для накладання текстури на поверхню спочатку було створено декілька змінних у коді шейдера, а потім створено посилання на них в коді програми. Також були розроблені функції для генерації текстурних даних.

Вибране зображення для текстури можна побачити на рис 3.2

Рис 3.2 Вибрана текстура

Поверхню “Cornucopia” з накладеною текстурою продемонстровано на рис 3.3

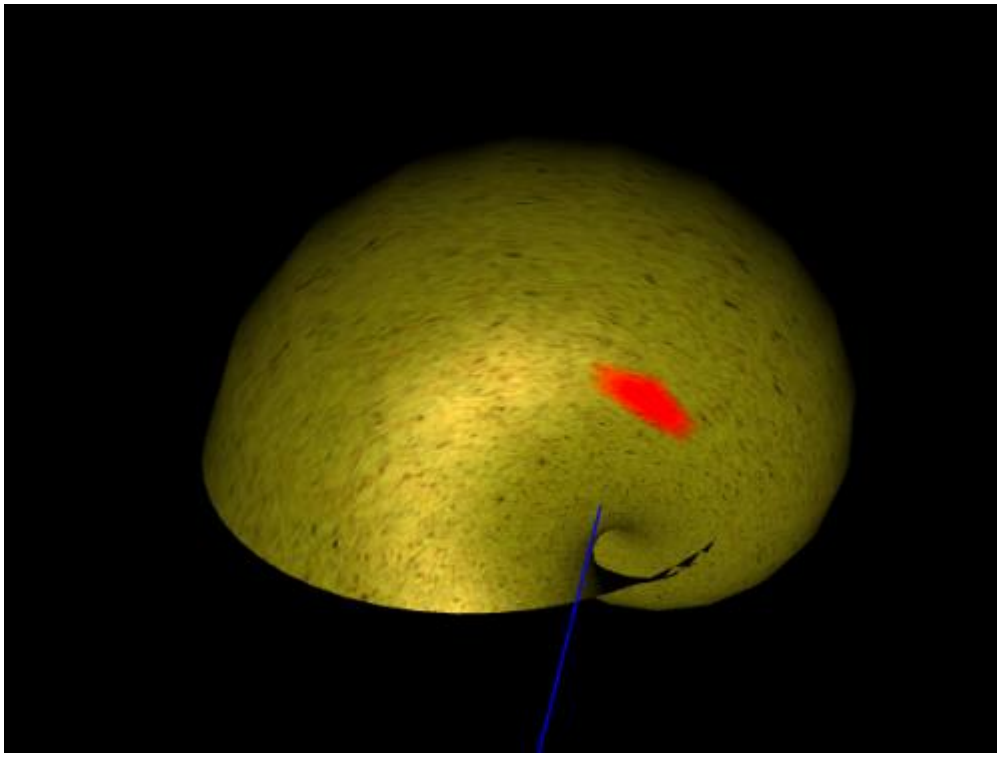

Рис 3.3 Поверхня “Cornucopia” з накладеною текстурою

Для опрацювання текстури були додані додаткові змінні у коді шейдера, а саме: параметри обертання текстури, визначення положення умовної точки у координатах (u, v), та змінну для позиціонування сфери на відповідному місці поверхні в тривимірному просторі.

Взаємодія користувача

Була додана можливість для користувача, керувати переміщенням умовних точок по поверхні, поворотом текстур щодо умовних точок і орієнтацією поверхні в просторі.

Переміщення умовної точки реалізовано за допомогою введення з клавіатури: 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.

Переміщення умовної точки зображено на рис 4.1

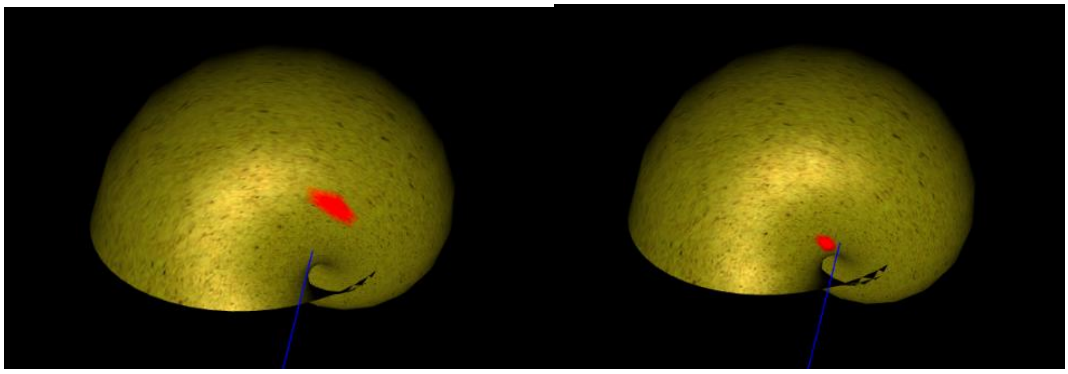

Рис. 4.1 Переміщення умовної точки

Напрямок поверхні в просторі контролюється за допомогою миші. На рис. 4.2 видно, що положення точки та текстури залишаються незмінними відносно поверхні. Змінилося лише напрямок орієнтації поверхні в просторі.

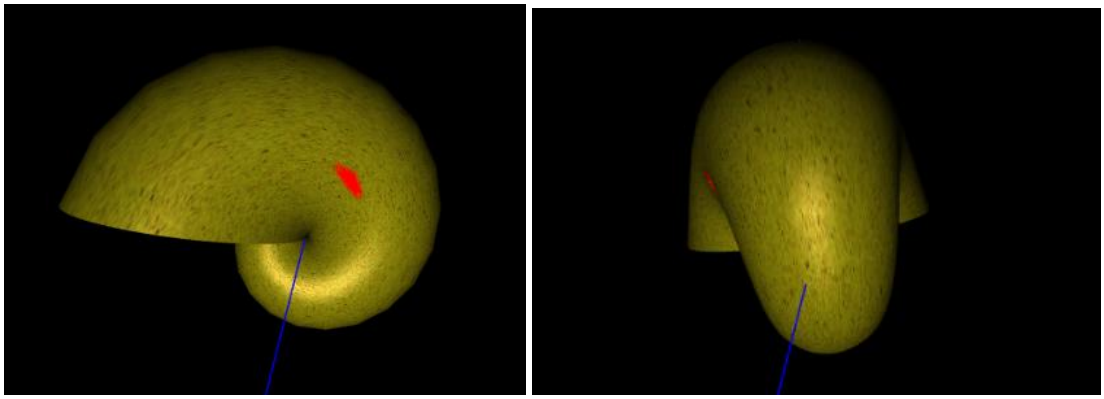

Рис 4.2 Зміна орієнтація поверхні в просторі

Також додана можливість повороту текстуру на певний градус з центром в умовній точці. Для цього на сторінку додано слайдер з допомогою чого можна це виконати.

На рис 4.3 продемонстровано поворот текстури (Через особливість текстури це не досить помітно).

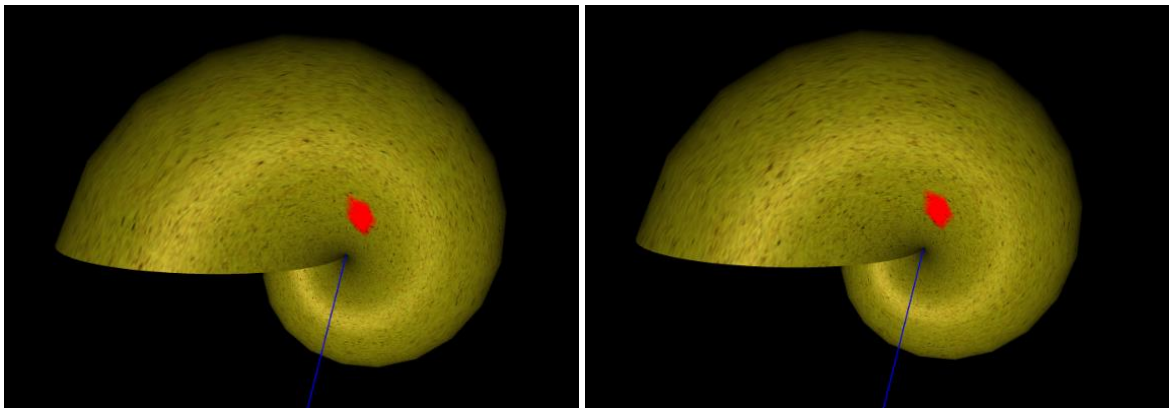

Рис 4.3 Поворот текстури

Висновки

У проведеній розрахунковій роботі було докладно вивчено концепції текстурування об'єкту, розгортки та UV-мапінгу. Під час дослідження ми зосередили увагу на вивченні того, як текстури взаємодіють з поверхнею об'єкту та яким чином вони можуть бути оптимально використані для досягнення бажаного візуального ефекту.

У процесі роботи було ефективно реалізовано можливість обертання текстури навколо визначеної користувачем точки. Це дозволяє користувачу легко контролювати розташування текстури на поверхні об'єкта та досягати необхідного ефекту візуалізації.

Загалом, результати цієї роботи свідчать про успішне засвоєння та вдалу імплементацію концепцій текстурування та відображення поверхні об'єктів у графічних програмах.

Код завандаження зображення

```
function LoadTexture() {
    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/MaxZastupailo/VGGI_Labs/CGW/texture.jpg";
    image.onload = () => {
        gl.bindTexture(gl.TEXTURE_2D, texture);
        gl.texImage2D(
            gl.TEXTURE_2D,
            0,
            gl.RGBA,
            gl.RGBA,
            gl.UNSIGNED_BYTE,
            image
        );

        SetupSurface();
        BuildSurface();

        SetupLine();
        BuildLine();

        SetupSegment();
        BuildSegment();


        draw()
    }
}
```

Функція відмальовування поверхну

```
this.Draw = function (projectionViewMatrix) {
        let rotation = spaceball.getViewMatrix();

        let translation = m4.translation(World_X, World_Y, World_Z);

        let modelMatrix = m4.multiply(translation, rotation);

        let modelViewProjection = m4.multiply(projectionViewMatrix, modelMatrix);

        var worldInverseMatrix = m4.inverse(modelMatrix);
        var worldInverseTransposeMatrix = m4.transpose(worldInverseMatrix);
```

```
        gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false,
modelViewProjection);
        gl.uniformMatrix4fv(shProgram.iWorldInverseTranspose, false,
worldInverseTransposeMatrix);

        gl.uniform3fv(shProgram.iMatAmbientColor, AmbientColor);
        gl.uniform3fv(shProgram.iMatDiffuseColor, DiffuseColor);
        gl.uniform3fv(shProgram.iMatSpecularColor, SpecularColor);
        gl.uniform1f(shProgram.iMatShininess, Shininess);

        gl.uniform3fv(shProgram.iLSAmbientColor, [0.1, 0.1, 0.1]);
        gl.uniform3fv(shProgram.iLSDiffuseColor, [LightIntensity, LightIntensity,
LightIntensity]);
        gl.uniform3fv(shProgram.iLSSpecularColor, [1, 1, 1]);

        gl.uniform3fv(shProgram.iCamWorldPosition, CameraPosition);
        gl.uniform3fv(shProgram.iLightDirection, GetDirLightDirection());

        gl.uniform2fv(shProgram.iRotationPoint, texturePoint);

        let point = CalculateCornucopiaPoint(map(texturePoint[0], 0, 1,uMin, uMax),
map(texturePoint[1], 0, 1,vMin, vMax));
        gl.uniform3fv(shProgram.iPointVizualizationPosition, [point.x, point.y,
point.z]);
        gl.uniform1f(shProgram.iRotationValue, rotateValue);

        gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
        gl.vertexAttribPointer(shProgram.iAttribVertex, 3, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(shProgram.iAttribVertex);

        gl.bindBuffer(gl.ARRAY_BUFFER, this.iNormalBuffer);
        gl.vertexAttribPointer(shProgram.iNormalVertex, 3, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(shProgram.iNormalVertex);

        gl.bindBuffer(gl.ARRAY_BUFFER, this.iTextureBuffer);
        gl.vertexAttribPointer(shProgram.iTextureCoords2D, 2, gl.FLOAT, false, 0,
0);
        gl.enableVertexAttribArray(shProgram.iTextureCoords2D);
        gl.uniform1i(shProgram.iTexture, 0);
        gl.enable(gl.TEXTURE_2D);

        gl.drawArrays(gl.TRIANGLE_STRIP, 0, this.count);
    }
```