МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра ЦТЕ

**ЗВІТ**

розрахунково-графічної роботи

з дисципліни: «Візуалізація графічної та геометричної інформації»

**Виконав:**

Ст. гр. ТР-22мп

Головакін М. А.

**Перевірив:**

Демчишин А. А.

Київ – 2023

# Постановка завдання

**Мета**: навчитись виконувати операцію масштабування з текстурними координатами у webGL на поверхні Неовіуса (варіант 5).

**Вимоги:**

* Нанести текстуру на поверхню з практичного завдання №2;
* Реалізувати масштабування текстури (координати текстури) навколо визначеної користувачем точки;
* Реалізувати переміщення точки вздовж простору поверхні *(u, v)* за допомогою клавіатури. Клавіші ***A*** та ***D*** переміщують точку вздовж параметра *u*, а клавіші ***W*** та ***S*** переміщують точку вздовж параметра *v.*

# Теоретичні відомості

У комп’ютерній графіці текстура (texture) – це растрове зображення, яке можна застосувати до поверхні тривимірного об’єкта. Текстури використовуються для додавання деталей, текстури поверхні та кольору до 3D-моделей і для створення ілюзії більш складного та реалістичного об’єкта.

Текстури можна створювати з файлів зображень або генерувати програмно. Зазвичай вони представлені як 2D-зображення, але їх також можна зберігати як 3D-зображення або як послідовність зображень (атлас текстур або аркуш спрайтів).

У 3D-графіці текстури використовуються для додавання деталей поверхні моделям шляхом відображення 2D-зображення на поверхні моделі. Цей процес називається відображенням текстури. Текстури можна використовувати для додавання кольору, карти рельєфу (які створюють ілюзію шорсткості поверхні), дзеркальні карти (які керують дзеркальним відблиском поверхні) і багато інших типів деталей поверхні.

**Texture Mapping** (Відображення текстури) — це процес застосування 2D-зображення (текстури) до 3D-моделі таким чином, щоб текстура виглядала як частина поверхні моделі. Це робиться шляхом проектування текстури на поверхню моделі, а потім обгортання текстури навколо поверхні.

Існує кілька способів нанесення текстури на 3D-модель:

1. **Planar mapping**: у цьому методі текстура проектується на модель так, ніби це плоска площина. Це простий метод, але він може спричинити спотворення, якщо поверхня моделі не рівна.
2. **Spherical mapping**: у цьому методі текстура проектується на модель так, ніби це сфера. Це корисно для відображення текстур на вигнутих поверхнях, таких як сфера чи циліндр.
3. **Cubic mapping**: у цьому методі текстура проектується на модель так, ніби це куб. Це корисно для нанесення текстур на складні поверхні неправильної форми.
4. **UV Mapping**: у цьому методі для поверхні моделі визначається двовимірна система координат текстури. Потім текстура наноситься на модель шляхом вирівнювання координат текстури з координатами текстури моделі. Це дуже гнучкий метод, який часто використовується в програмному забезпеченні для 3D-моделювання.

Текстурування є важливою технікою в 3D-графіці та використовується в багатьох різних типах програм, включаючи відеоігри, фільми, 3D-моделювання та віртуальну реальність.

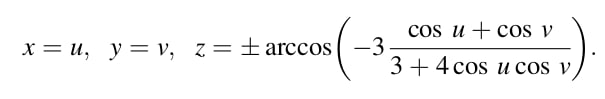
Затінення Гуро (Gouraud) — це метод інтерполяції нормалей до поверхні 3D-моделі та використання цих інтерпольованих нормалей для обчислення дифузного освітлення в кожному пікселі. Він може створювати плавне безперервне затінення по всій поверхні, але також може бути менш точним, ніж інші методи, такі як затінення Фонга (Phong), особливо на поверхневих деталях, таких як краї та кути.

# Деталі реалізації

У коді JavaScript поля *iVertexBuffer*, *iNormalBuffer* і *iTextureBuffer* об’єкта *Model* представляють об’єкти буфера WebGL для зберігання даних вершин, нормалей і текстур відповідно. Метод *BufferData* об’єкта *Model* використовується для надсилання цих даних у буфери WebGL.

Дані вершин, нормалей і текстури передаються в метод *BufferData* як об’єкт із трьома властивостями: *vertexList*, *normalList* і *textureList*. Ці властивості являють собою масиви значень, що представляють дані вершини, нормалі та текстури моделі відповідно.

Вершини вираховуються за допомогою функції:



Формула представляє собою параметричне рівняння поверхні Неовіуса (Neovious’ Surface).

Щоб розрахувати аналітичну нормаль, необхідно знайти часткові похідні функції *z* по *u* та *v*, сформувати тангент-вектори *U* та *V*, взяти cross product та нормалізувати їх.

Затінення Гуро реалізовано у вершинному шейдері (vertex shader). Зокрема, вершинний шейдер обчислює дифузне освітлення для кожної вершини за допомогою наступних кроків:

1. Розрахунок вектора нормалі *N* у вершині шляхом нормалізації нормалі і перетворенні його за допомогою *normalMatrix*
2. Розрахунок напряму джерела світла *L* шляхом нормалізації *lightDir*.
3. Розрахунок розсіяного освітлення у вершині, беручи скалярний добуток *N* і *L* і фіксуючи результат до мінімального значення 0. Цей скалярний добуток представляє кут між нормальним вектором і напрямком світла зі значенням 1, що вказує на те, що поверхня спрямована прямо до джерела світла, а значення 0 вказує на те, що поверхня перпендикулярна до джерела світла.
4. Обчислення кольору вершини, додаючи колір навколишнього середовища *ambientColor* до дифузного кольору *diffuseColor*, помноженого на розсіяне освітлення. Цей колір зберігається у змінній *vertColor* і використовуватиметься для обчислення кольору кожного пікселя у фрагментному шейдері.

В об’єкті *ShaderProgram* поля *iAttribVertex*, *iAttribNormal* і *iTexCoord* представляють розташування змінних атрибутів для даних вершини, нормалей та текстури в програмі шейдера відповідно. Потім шейдерна програма використовується для візуалізації моделі як *TRIANGLE\_STRIP* (набір трикутників), використовуючи дані вершин, нормалей і текстур із буферів WebGL.

Функція *getPointsLocation* була створена, аби знаходити положення точки на поверхні. В ній використовуються змінні *pX*, *pY* як положення у 2D синтетичний координатах. *scale* використовується як значення параметру масштабування.

Функція *createTexture* була створена, аби завантажити текстуру з інтернету.

Шейдер містить 3 функції: *scaling* (матриця масштабування), *trans* (матриця переміщення), *getTextCoordinates* (масштабовані координати).

# Інструкція використання

Програму можна завантажити, використовуючи *git clone <посилання на репозиторій>* та відкрити *index.html*, однак є можливість перейти на сайт, який вказаний у README.md.

Відкривши сайт, маємо опис основних елементів управління (рис. 1):

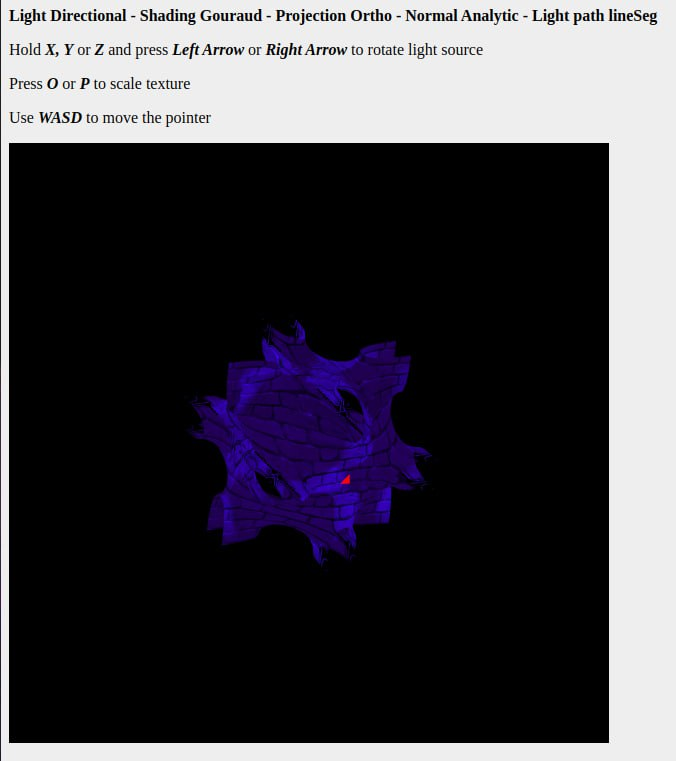


Рис. 1. Головний екран програми

Користувач може затиснути ліву кнопку комп’ютерної миші та відвести її, щоб перемістити фігуру навколо центру її координат (рис. 2):

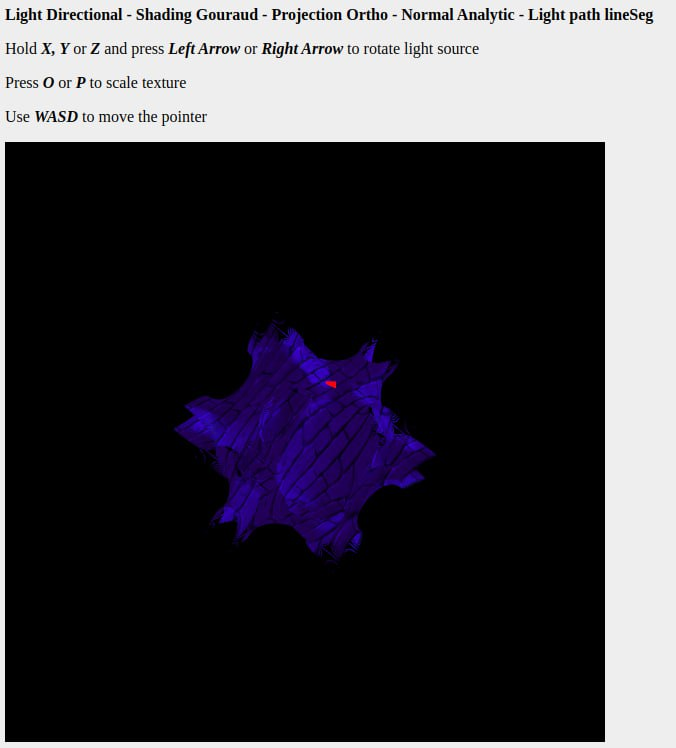


Рис. 2. Переміщення навколо центру координат

Налаштування світла здійснюється завдяки затисканню однієї з кнопок X, Y, Z та одночасним натисканням стрілки вліво чи вправо (рис. 3):

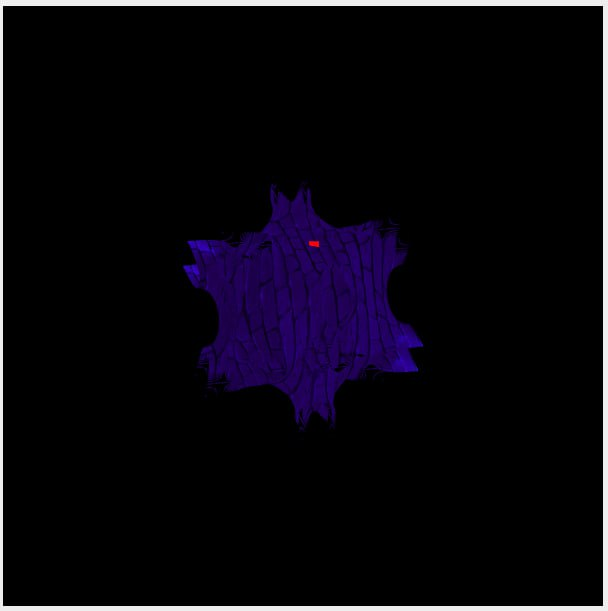


Рис. 3. Зміна координат освітлення

Переміщення точки відбувається завдяки натисканню кнопок W, A, S, D (рис. 4):

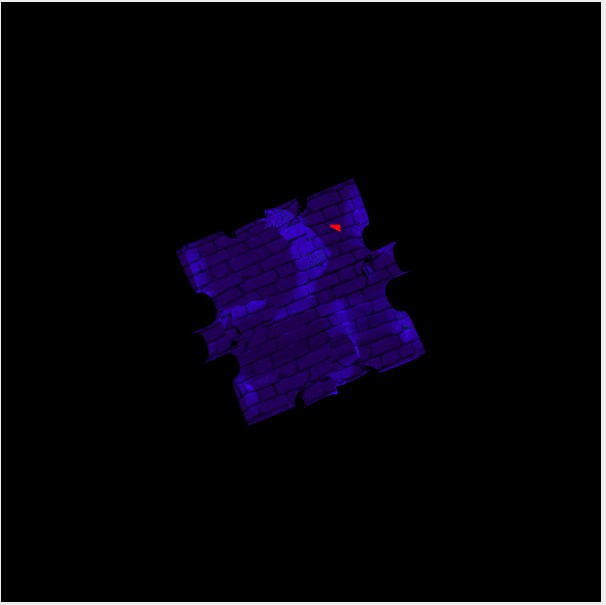


Рис. 4. Переміщення точки

Зміна масштабу відбувається на кнопки O та P (рис. 5):

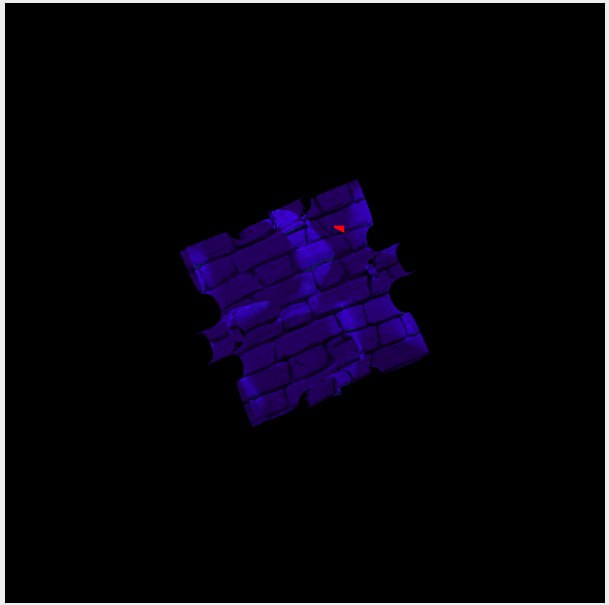


Рис. 5. Масштабування

# Код програми

Обчислення вершин, нормалей і текстури:

function CreateSurfaceData() {

    const step = .5;

    const min = -180;

    const max = 180;

    let vertexList = [];

    let normalList = [];

    let textureList = [];

    for (let u = min; u < max; u += step) {

        const uRad = deg2rad(u);

        for (let v = min; v <= max; v += step) {

            const vRad = deg2rad(v);

            const h = 0.0001;

            const df\_du = (f(uRad + h, vRad) - f(uRad, vRad)) / deg2rad(h);

            const df\_dv = (f(uRad, vRad + h) - f(uRad, vRad)) / deg2rad(h);

            const plusTangentU = m4.normalize([1, vRad, df\_du]);

            const plusTangentV = m4.normalize([uRad, 1, df\_dv]);

            const minusTangentU = m4.normalize([1, vRad, -df\_du]);

            const minusTangentV = m4.normalize([uRad, 1, -df\_dv]);

            const plusNormal = m4.normalize(m4.cross(plusTangentU, plusTangentV));

            const minusNormal = m4.normalize(m4.cross(minusTangentU, minusTangentV));

            vertexList.push(uRad, vRad, f(uRad, vRad));

            vertexList.push(uRad, vRad, -f(uRad, vRad));

            normalList.push(plusNormal[0], plusNormal[1], plusNormal[2]);

            normalList.push(minusNormal[0], minusNormal[1], minusNormal[2]);

            textureList.push((uRad - min) / (max - min), (vRad - min) / (max - min));

            textureList.push((uRad - min) / (max - min), (vRad - min) / (max - min));

    }

}

  return { vertexList, normalList, textureList };

}

Знаходження точки на поверхні:

function getPointLocation(){

    let pointList = [];

    let x,y,z;

    x = deg2rad(pX);

    y = deg2rad(pY);

    z = f(x, y);

    pointList.push(x,y,z);

    return pointList;

}

Створення текстури:

function createTexture() {

    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);

    gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED\_BYTE,

        new Uint8Array([255, 255, 255, 255]));

    let img = new Image();

    img.crossOrigin = "anonymous";

    img.src = 'https://www.manytextures.com/download/36/texture/jpg/512/red-brick-wall-512x512.jpg';

    img.addEventListener('load', function() {

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

        gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, img);

        draw();

    });

}