Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

**Візуалізація графічної та геометричної інформації**

**Розрахунково-графічна робота**

Лень В.В.

ТР-42мп

Каленюк О.С.

Демчишин А.А

1. **Завдання**

Реалізація візуалізації поверхні з використанням текстур та інтерактивного керування.

1. **Візуалізація поверхні з використанням текстур**  
   Реалізувати відображення поверхні, яка використовує три типи текстур:
   * Дифузна текстура: модулює компонент дифузного освітлення.
   * Дзеркальна текстура: модулює компонент дзеркального освітлення.
   * Текстура нормалі: містить дані нормалей для піксельного освітлення.
2. **Ортогоналізація Грама-Шмідта**  
   При виконанні ортогоналізації забезпечити пріоритет нормалі (головна вісь нормалі має залишатися стабільною, коригуючи інші напрями).
3. **Реалізація масштабування текстури**  
   Додати можливість масштабування текстурних координат з урахуванням визначеної користувачем точки.
4. **Інтерактивне керування точкою масштабування**  
   Реалізувати можливість переміщення точки масштабування вздовж простору поверхні (u, v) за допомогою клавіатури:
   * Клавіші A та D переміщують точку вздовж параметра u (горизонталь).
   * Клавіші W та S переміщують точку вздовж параметра v (вертикаль).
5. **Теорія**

**Візуалізація поверхні з використанням текстур**

Текстурування — це процес нанесення двовимірного зображення (текстури) на поверхню тривимірного об’єкта для підвищення реалістичності та деталізації. У сучасній комп’ютерній графіці використовується кілька типів текстур для створення складних візуальних ефектів:

1. **Дифузна текстура (Diffuse texture):**  
   Відповідає за визначення основного кольору поверхні. Вона використовується в моделі освітлення для розрахунку дифузного компоненту світла, який залежить від кута падіння променя світла на поверхню. Найпоширеніша модель — освітлення Ламберта, де яскравість поверхні пропорційна косинусу кута між нормаллю поверхні та напрямком до джерела світла.
2. **Дзеркальна текстура (Specular texture):**  
   Визначає, наскільки інтенсивно поверхня відбиває світло в напрямку спостерігача. Використовується для симуляції блиску, глянцевих матеріалів і реалістичного відображення. Модель Фонга є найпоширенішою для розрахунку дзеркального компоненту, використовуючи нормаль, напрямок до світла та позицію спостерігача.
3. **Звичайна текстура (Normal texture):**  
   Це вид текстури, що використовується для моделювання складного рельєфу поверхні шляхом зміни нормалі для кожного пікселя. Звичайна текстура зберігає нормалі у вигляді кольорових значень (RGB), які інтерпретуються як вектори у тривимірному просторі. Це дозволяє досягти більш детального освітлення без збільшення геометричної складності.

**Ортогоналізація Грама-Шмідта**

Процес ортогоналізації дозволяє отримати базис із взаємно перпендикулярних векторів. У комп’ютерній графіці це необхідно для коректного визначення тангентно-нормального простору, що складається з трьох основних векторів:

* Нормаль (Normal) — перпендикуляр до поверхні.
* Тангенс (Tangent) — напрямок уздовж текстурної координати u.
* Бітангенс (Bitangent) — напрямок уздовж текстурної координати v.

Алгоритм Грама-Шмідта забезпечує стабільність розрахунків навіть при неточних початкових даних, пріоритет нормалі дозволяє уникнути артефактів освітлення, таких як спотворення відблисків або некоректне текстурування.

**Інтерактивне керування точкою масштабування**

Для динамічного налаштування координат текстур користувач може переміщувати точку масштабування за допомогою клавіш:

* **Клавіші A/D** змінюють координату u, рухаючи точку вліво/вправо.
* **Клавіші W/S** змінюють координату v, рухаючи точку вгору/вниз.

Це забезпечується обробкою подій клавіатури, які в режимі реального часу змінюють значення параметрів. Для плавності руху можна використовувати інтерполяцію значень.

**Оптимізація для реального часу**

Щоб забезпечити плавну візуалізацію, необхідно оптимізувати обчислення. Використання графічних шейдерів (наприклад, у OpenGL або HLSL) дозволяє перенести основну обробку текстур та освітлення на GPU. Також слід враховувати:

* Буферизацію текстурних координат для уникнення повторних розрахунків.
* Мультитредингову обробку векторів та текстур.

**Практичне застосування**

Подібні підходи використовуються у розробці ігор, симуляцій та візуалізації складних матеріалів, таких як метали, тканини або органічні поверхні. Інтерактивне управління параметрами дозволяє створювати динамічні ефекти для візуального підсилення сцени.

1. **Реалізація**

Реалізована система дозволяє інтерактивно масштабувати текстуру на 3D поверхні відносно заданої точки у текстурному просторі (u,v). Користувач може:

* Переміщувати точку масштабування за допомогою клавіш WASD
* Змінювати масштаб текстури відносно цієї точки за допомогою клавіш +/-
* Обертати об'єкт для перегляду результату з різних ракурсів

Ключові компоненти системи

1. **Управління станом текстури**

const textureState = {

scalePoint: { u: 0.5, v: 0.5 }, *// Точка масштабування в UV-просторі*

scaleFactor: 1.0, *// Коефіцієнт масштабування*

moveSpeed: 0.1, *// Швидкість руху точки*

scaleSpeed: 0.2 *// Швидкість зміни масштабу*

};

1. **Обробка користувацького вводу**

Система відслідковує натискання клавіш та оновлює параметри текстурування:

function handleKeyPress(event) {

switch(event.key.toLowerCase()) {

case 'w': *// Рух вгору по V координаті*

textureState.scalePoint.v = Math.max(0,

textureState.scalePoint.v - textureState.moveSpeed);

break;

case 's': *// Рух вниз по V координаті*

textureState.scalePoint.v = Math.min(1,

textureState.scalePoint.v + textureState.moveSpeed);

break;

*// ... аналогічно для A/D та +/-*

}

*// ...*

}

1. **Генерація текстурних координат**

Ключова частина системи - це алгоритм трансформації текстурних координат у функції CreateSurfaceData:

*// Базові текстурні координати*

const u = j / segments;

const v = i / segments;

*// Обчислення зміщення від точки масштабування*

const du = u - scalePoint.u;

const dv = v - scalePoint.v;

*// Масштабування відносно точки*

const scaledU = scalePoint.u + du \* scaleFactor;

const scaledV = scalePoint.v + dv \* scaleFactor;

*// Забезпечення повторення текстури*

const wrappedU = ((scaledU % 1) + 1) % 1;

const wrappedV = ((scaledV % 1) + 1) % 1;

texcoords.push(wrappedU, wrappedV);

1. **Масштабування відносно точки**

Для масштабування текстурних координат відносно заданої точки використовується наступний алгоритм:

1. Обчислення вектора зміщення від точки масштабування до поточної точки
2. Множення цього вектора на коефіцієнт масштабування
3. Додавання результату до точки масштабування
4. **Оптимізація продуктивності**
5. Текстури завантажуються один раз і кешуються:

async function loadTextures() {

textures = {

diffuse: LoadTexture(gl, "textures/diffuse.png"),

specular: LoadTexture(gl, "textures/specular.png"),

normal: LoadTexture(gl, "textures/normal.png")

};

}

1. Оновлення геометрії відбувається тільки при необхідності:

if (needsUpdate) {

initSurface();

draw();

}

1. **Інструкції користувача**

**Загальний опис**

Ця програма дозволяє вам взаємодіяти з 3D об'єктом, на який накладена текстура. Ви можете маніпулювати текстурою, змінюючи її масштаб відносно обраної точки та переміщуючи цю точку по поверхні об'єкта.

**Елементи керування** зазначені під полем з 3D об’єктом.

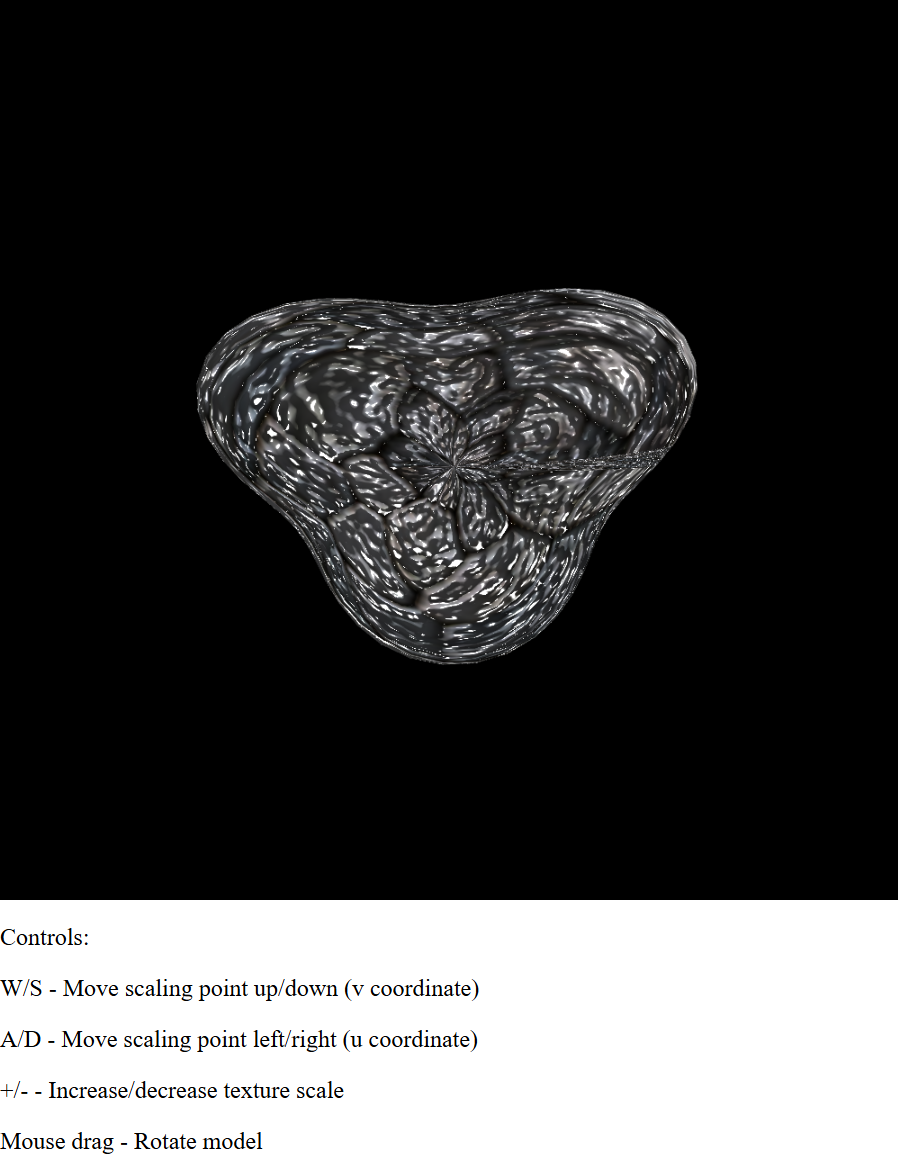


Рисунок 1. Приклад елементів керування об’єктом

**Керування масштабом текстури**

* + або = - збільшити масштаб текстури
  + - - - зменшити масштаб текстури



Рисунок 2. Приклад збільшення та зменшення масштабу

**Переміщення точки масштабування**

* W - рух точки масштабування вгору
* S - рух точки масштабування вниз
* A - рух точки масштабування вліво
* D - рух точки масштабування вправо



Рисунок 3. Приклад переміщення точки масштабування

**Управління виглядом об'єкта**

* Ліва кнопка миші - обертання об'єкта
  + Утримуйте ліву кнопку миші та переміщуйте курсор для обертання об'єкта
  + Це дозволяє розглянути об'єкт з різних ракурсів



Рисунок 4. Приклад обертання об’єкта

**Поради з використання**

1. Почніть з невеликих змін масштабу, щоб звикнути до поведінки текстури
2. Використовуйте клавіші WASD для точного позиціонування точки масштабування
3. Комбінуйте обертання об'єкта з маніпуляціями текстурою для кращого розуміння результату
4. **Приклад коду**

**geometry.js**

export function CreateSurfaceData(data, scalePoint = { u: 0.5, v: 0.5 }, scaleFactor = 1.0) {

    const segments = 50;

    const vertices = [];

    const texcoords = [];

    const normals = [];

    const indices = [];

    const a = 1.0;

    const b = 0.8;

    const c = 0.6;

    for (let i = 0; i <= segments; i++) {

        const phi = (i \* Math.PI) / segments;

        const sinPhi = Math.sin(phi);

        const cosPhi = Math.cos(phi);

        for (let j = 0; j <= segments; j++) {

            const theta = (j \* 2 \* Math.PI) / segments;

            const sinTheta = Math.sin(theta);

            const cosTheta = Math.cos(theta);

            const x = a \* cosTheta \* sinPhi \* (1 + 0.2 \* Math.sin(3 \* theta));

            const y = b \* sinTheta \* sinPhi \* (1 + 0.2 \* Math.sin(3 \* theta));

            const z = c \* cosPhi;

            vertices.push(x, y, z);

            const u = j / segments;

            const v = i / segments;

            const du = u - scalePoint.u;

            const dv = v - scalePoint.v;

            const scaledU = scalePoint.u + du \* scaleFactor;

            const scaledV = scalePoint.v + dv \* scaleFactor;

            const wrappedU = ((scaledU % 1) + 1) % 1;

            const wrappedV = ((scaledV % 1) + 1) % 1;

            texcoords.push(wrappedU, wrappedV);

            const nx = x / a;

            const ny = y / b;

            const nz = z / c;

            const length = Math.sqrt(nx \* nx + ny \* ny + nz \* nz);

            normals.push(nx / length, ny / length, nz / length);

        }

    }

    for (let i = 0; i < segments; i++) {

        for (let j = 0; j < segments; j++) {

            const first = (i \* (segments + 1)) + j;

            const second = first + segments + 1;

            indices.push(first, second, first + 1);

            indices.push(second, second + 1, first + 1);

        }

    }

    calculateTangentsAndBitangents(vertices, texcoords, normals, indices, data);

    data.verticesF32 = new Float32Array(vertices);

    data.normalsF32 = new Float32Array(normals);

    data.texcoordsF32 = new Float32Array(texcoords);

    data.indicesU16 = new Uint16Array(indices);

}

**main.js**

import { createShaderProgram } from './shaders.js';

import { Model } from './model.js';

import { LoadTexture } from './textureLoader.js';

import { CreateSurfaceData } from './geometry.js';

let gl;

let surface;

let shProgram;

let spaceball;

let textures = null;

const textureState = {

    scalePoint: { u: 0.5, v: 0.5 },

    scaleFactor: 1.0,

    moveSpeed: 0.2,

    scaleSpeed: 0.2

};

function loadTextures() {

    textures = {

        diffuse: LoadTexture(gl, "textures/diffuse.png"),

        specular: LoadTexture(gl, "textures/specular.png"),

        normal: LoadTexture(gl, "textures/normal.png")

    };

    initSurface();

}

function initSurface() {

    let data = {};

    CreateSurfaceData(data, textureState.scalePoint, textureState.scaleFactor);

    surface = new Model('Surface', gl, shProgram);

    surface.BufferData(data.verticesF32, data.indicesU16, data.texcoordsF32,

                      data.normalsF32, data.tangentsF32, data.bitangentsF32);

    if (textures) {

        surface.diffuseMap = textures.diffuse;

        surface.specularMap = textures.specular;

        surface.normalMap = textures.normal;

    }

    draw();

}

function draw() {

    if (!surface || !surface.diffuseMap) return;

    gl.clearColor(0,0,0,1);

    gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

    let projection = m4.perspective(Math.PI/8, 1, 8, 12);

    let modelView = spaceball.getViewMatrix();

    let rotateToPointZero = m4.axisRotation([0.707,0.707,0], 0.7);

    let translateToPointZero = m4.translation(0,0,-10);

    let matAccum0 = m4.multiply(rotateToPointZero, modelView);

    let matAccum1 = m4.multiply(translateToPointZero, matAccum0);

    let modelViewProjection = m4.multiply(projection, matAccum1);

    let normalMatrix = m4.transpose(m4.inverse(matAccum1));

    surface.draw(modelView, modelViewProjection, normalMatrix);

}