МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

КАФЕДРА ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Розрахункова-графічна робота

з дисципліни «Візуалізація графічної та геометричної інформації»

Варіант №8

**Виконав:**

Студент 1-го курсу магістратури

ІАТЕ

групи ТР-22мп

Євтушенко Дмитро Миколайович

**Перевірив:**

Демчишин А.А.

Київ-2022

**Завдання**

1. Нанести текстуру на поверхню з лабораторної роботи №2.
2. Розробити програму для обертання текстури навколо точки що задає користувач.
3. Розробити можливість змінювати положення точки користувача вздовж u, v за допомогою клавіатури: клавіші A і D переміщують точку вздовж параметра u, а клавіші W і S переміщують точку вздовж параметра v.

4. Створити гілку CGW в репозиторії та завантажити розроблену програму.

5. Створити звіт до розрахунково-графічної роботи та завантажити в гілку.

**Теорія**

WebGL (Web Graphics Library) - програмна бібліотека для мови JavaScript, призначена для візуалізації інтерактивної тривимірної графіки та двовимірної графіки в межах сумісності веб-браузера без використання плагінів. WebGL приносить у веб тривимірну графіку, вводячи API, який побудований на основі OpenGL ES 2.0, що дозволяє його використовувати в елементах canvas HTML5.

Використання цього потужного API вимагає значного розуміння 3D-програмування та 3D-математики. Веб-розробники можуть виявити проблеми через його низький рівень. Один із способів спростити використання веб-розробниками WebGL полягає в наданні середнього рівня API.

Тож, WebGL повністю інтегрований з іншими Web-API, що дозволяє за допомогою графічного процесора використовувати фізику, обробку зображень та ефекти як частину веб-сторінки.

WebGL можна з'єднувати з HTML елементами та поєднувати з іншими частинами веб-сторінки або фону сторінки.

Створення технології WebGL стало кроком уперед, що дозволяє відображати та маніпулювати тривимірною графікою на web-сторінках за допомогою JavaScript. Завдяки WebGL розробники можуть створювати сучасні розширені інтерфейси, призначені для користувача, тривимірні ігри та використовувати тривимірну графіку для візуалізації різної інформації з Інтернету. Незважаючи на значні можливості, WebGL відрізняється від інших технологій доступністю та простотою у використанні, що сприяє її швидкому поширенню.

Щоб завантажити текстуру з файлу зображення, функція створює об'єкт Image і надає атрибуту src адресу, з якої ми хочемо завантажити текстуру. Функція, яку ми призначили на подію image.onload, буде викликана після завершення завантаження зображення. У цей момент ми викликаємо texImage2D() (en-US), використовуючи завантажене зображення як вихідний текстури. Потім ми встановлюємо фільтрацію та натяг, виходячи з того, чи є розмір зображення ступенем 2 чи ні.

У WebGL1 зображення розміру, що не є ступенем 2, можуть використовувати тільки NEAREST або LINEAR фільтрацію, і не можна створити mipmap. Також для таких зображень необхідно встановити натяг CLAMP\_TO\_EDGE. З іншого боку, якщо зображення має розмір ступеня 2 по обидва осі, WebGL може проводити більш якісну фільтрацію, використовувати mipmap і режими натягу REPEAT або MIRRORED\_REPEAT.

Прикладом текстури, що повторюється, є зображення кількох цегли, яке розмножується для покриття поверхні і створення зображення цегляної стіни.

Міпмапінг та UV-повторення можуть бути відключені за допомогою texParameteri() (en-US). Так ви зможете використовувати текстури з розміром, що не є ступенем 2 (NPOT - non-power-of-two), ціною відключення міпмапінгу, UV-натягу, UV-повторення, і вам самому доведеться контролювати, як саме пристрій оброблятиме текстуру.

**Реалізація продукту**

Створюємо функцію у файлі main.js LoadTexture . Ця функція буде створювати WebGl об'єкт текстури та об'єкт картинки Image, після чого картинка буде завантажуватися з Інтернету за допомогою функції onLoad та привязуватися до об'єкту текстури. Представлена функція:

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/dmtiriyy/WEBGLLABWORKS/CGW/image\_texture.jpg";

    image.onload = () => {

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

        gl.texImage2D(

            gl.TEXTURE\_2D,

            0,

            gl.RGBA,

            gl.RGBA,

            gl.UNSIGNED\_BYTE,

            image

        );

        console.log("imageLoaded")

        draw()

    }

}

Створюємо буфер, який потрібний для координат текстури і його прив’язка до атрибуту

  this.iVertexBuffer = gl.createBuffer();

    this.iTextureBuffer = gl.createBuffer();

    this.count = 0;

    this.countText = 0;

    this.BufferData = function (vertices) {

        gl.bindBuffer(gl.ARRAY\_BUFFER, this.iVertexBuffer);

        gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(vertices), gl.STREAM\_DRAW);

        this.count = vertices.length / 3;

    }

    this.TextureBufferData = function (normals) {

        gl.bindBuffer(gl.ARRAY\_BUFFER, this.iTextureBuffer);

        gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(normals), gl.STREAM\_DRAW);

        this.countText = normals.length / 2;

    }

Для того щоб перетворити передану точку нам знадобляться функції translate() та rotate(). Оскільки їх немає в стандартній бібліотеці, необхідно створити їх самим.

    shProgram.iTranslatePoint = gl.getUniformLocation(prog, 'translatePoint');

    shProgram.iTexturePoint = gl.getUniformLocation(prog, 'texturePoint');

    shProgram.iRotateValue = gl.getUniformLocation(prog, 'rotateValue');

// функція для обертання текстури

onmousemove = (e) => {

    rotateValue = map(e.clientX, 0, window.outerWidth, 0, Math.PI)

    draw()

};

Після чого створюємо функцію для обертання текстури навколо точки, що задає користувач

function mat4Invert(m, inverse) {

    var inv = new Float32Array(16);

    inv[0] = m[5] \* m[10] \* m[15] - m[5] \* m[11] \* m[14] - m[9] \* m[6] \* m[15] +

        m[9] \* m[7] \* m[14] + m[13] \* m[6] \* m[11] - m[13] \* m[7] \* m[10];

    inv[4] = -m[4] \* m[10] \* m[15] + m[4] \* m[11] \* m[14] + m[8] \* m[6] \* m[15] -

        m[8] \* m[7] \* m[14] - m[12] \* m[6] \* m[11] + m[12] \* m[7] \* m[10];

    inv[8] = m[4] \* m[9] \* m[15] - m[4] \* m[11] \* m[13] - m[8] \* m[5] \* m[15] +

        m[8] \* m[7] \* m[13] + m[12] \* m[5] \* m[11] - m[12] \* m[7] \* m[9];

    inv[12] = -m[4] \* m[9] \* m[14] + m[4] \* m[10] \* m[13] + m[8] \* m[5] \* m[14] -

        m[8] \* m[6] \* m[13] - m[12] \* m[5] \* m[10] + m[12] \* m[6] \* m[9];

    inv[1] = -m[1] \* m[10] \* m[15] + m[1] \* m[11] \* m[14] + m[9] \* m[2] \* m[15] -

        m[9] \* m[3] \* m[14] - m[13] \* m[2] \* m[11] + m[13] \* m[3] \* m[10];

    inv[5] = m[0] \* m[10] \* m[15] - m[0] \* m[11] \* m[14] - m[8] \* m[2] \* m[15] +

        m[8] \* m[3] \* m[14] + m[12] \* m[2] \* m[11] - m[12] \* m[3] \* m[10];

    inv[9] = -m[0] \* m[9] \* m[15] + m[0] \* m[11] \* m[13] + m[8] \* m[1] \* m[15] -

        m[8] \* m[3] \* m[13] - m[12] \* m[1] \* m[11] + m[12] \* m[3] \* m[9];

    inv[13] = m[0] \* m[9] \* m[14] - m[0] \* m[10] \* m[13] - m[8] \* m[1] \* m[14] +

        m[8] \* m[2] \* m[13] + m[12] \* m[1] \* m[10] - m[12] \* m[2] \* m[9];

    inv[2] = m[1] \* m[6] \* m[15] - m[1] \* m[7] \* m[14] - m[5] \* m[2] \* m[15] +

        m[5] \* m[3] \* m[14] + m[13] \* m[2] \* m[7] - m[13] \* m[3] \* m[6];

    inv[6] = -m[0] \* m[6] \* m[15] + m[0] \* m[7] \* m[14] + m[4] \* m[2] \* m[15] -

        m[4] \* m[3] \* m[14] - m[12] \* m[2] \* m[7] + m[12] \* m[3] \* m[6];

    inv[10] = m[0] \* m[5] \* m[15] - m[0] \* m[7] \* m[13] - m[4] \* m[1] \* m[15] +

        m[4] \* m[3] \* m[13] + m[12] \* m[1] \* m[7] - m[12] \* m[3] \* m[5];

    inv[14] = -m[0] \* m[5] \* m[14] + m[0] \* m[6] \* m[13] + m[4] \* m[1] \* m[14] -

        m[4] \* m[2] \* m[13] - m[12] \* m[1] \* m[6] + m[12] \* m[2] \* m[5];

    inv[3] = -m[1] \* m[6] \* m[11] + m[1] \* m[7] \* m[10] + m[5] \* m[2] \* m[11] -

        m[5] \* m[3] \* m[10] - m[9] \* m[2] \* m[7] + m[9] \* m[3] \* m[6];

    inv[7] = m[0] \* m[6] \* m[11] - m[0] \* m[7] \* m[10] - m[4] \* m[2] \* m[11] +

        m[4] \* m[3] \* m[10] + m[8] \* m[2] \* m[7] - m[8] \* m[3] \* m[6];

    inv[11] = -m[0] \* m[5] \* m[11] + m[0] \* m[7] \* m[9] + m[4] \* m[1] \* m[11] -

        m[4] \* m[3] \* m[9] - m[8] \* m[1] \* m[7] + m[8] \* m[3] \* m[5];

    inv[15] = m[0] \* m[5] \* m[10] - m[0] \* m[6] \* m[9] - m[4] \* m[1] \* m[10] +

        m[4] \* m[2] \* m[9] + m[8] \* m[1] \* m[6] - m[8] \* m[2] \* m[5];

    var det = m[0] \* inv[0] + m[1] \* inv[4] + m[2] \* inv[8] + m[3] \* inv[12];

    if (det == 0) return false;

    det = 1.0 / det;

    for (var i = 0; i < 16; i++) inverse[i] = inv[i] \* det;

    return true;

}

**Результати виконання**

На рисунку 1 відображені результати роботи з нанесенням текстури:

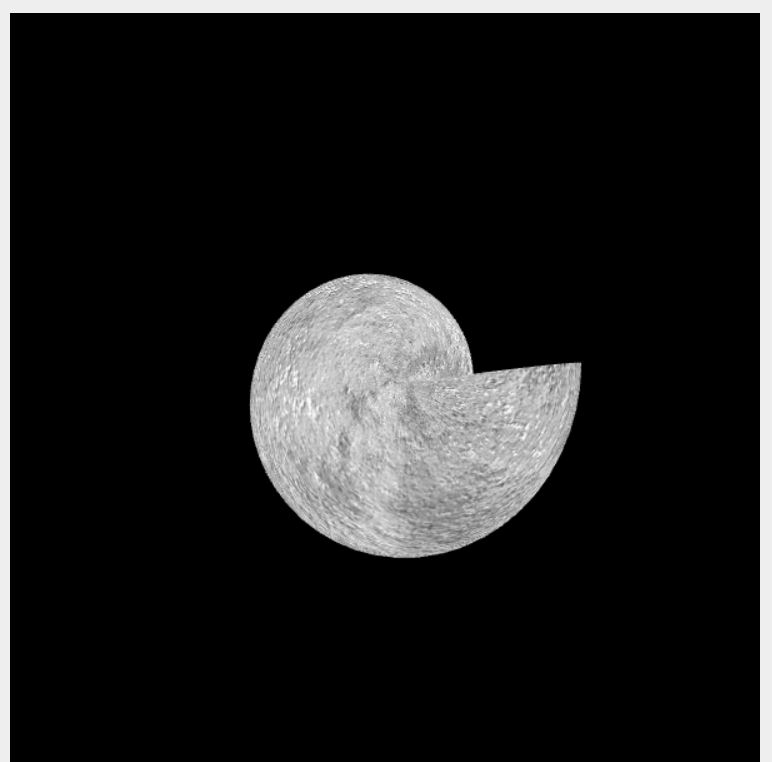


Рисунок 1- поверхня з нанесеною текстурою

На рисунку 2 було змінено кут обертання фігури:

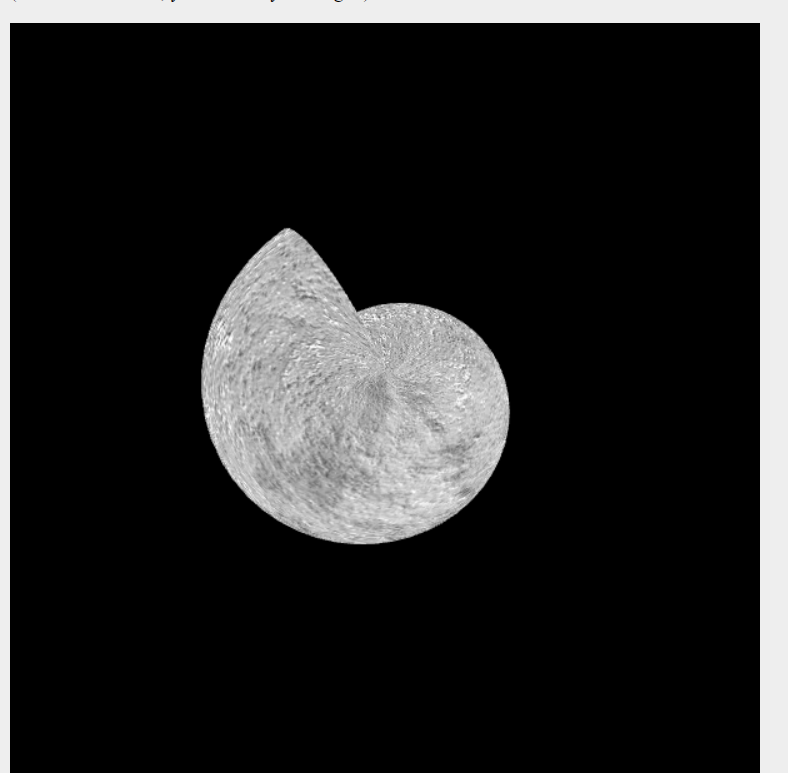


Рисунок 2- поверхня з зміненими параметрами

**Код виконання роботи**

function CreateSurfaceData() {

    let vertexList = [];

    let u = 0;

    let v = 0;

    let uMax = Math.PI \* 2

    let vMax = Math.PI \* 2

    let uStep = uMax / 50;

    let vStep = vMax / 50;

    for (let u = 0; u <= uMax; u += uStep) {

        for (let v = 0; v <= vMax; v += vStep) {

            let vert = Cornucopia(u, v)

            let avert = Cornucopia(u + uStep, v)

            let bvert = Cornucopia(u, v + vStep)

            let cvert = Cornucopia(u + uStep, v + vStep)

            vertexList.push(vert.x, vert.y, vert.z)

            vertexList.push(avert.x, avert.y, avert.z)

            vertexList.push(bvert.x, bvert.y, bvert.z)

            vertexList.push(avert.x, avert.y, avert.z)

            vertexList.push(cvert.x, cvert.y, cvert.z)

            vertexList.push(bvert.x, bvert.y, bvert.z)

        }

    }

    return vertexList;

}

function CreateTexture() {

    let texture = [];

    let u = 0;

    let v = 0;

    let uMax = Math.PI \* 2

    let vMax = Math.PI \* 2

    let uStep = uMax / 50;

    let vStep = vMax / 50;

    for (let u = 0; u <= uMax; u += uStep) {

        for (let v = 0; v <= vMax; v += vStep) {

            let u1 = map(u, 0, uMax, 0, 1)

            let v1 = map(v, 0, vMax, 0, 1)

            texture.push(u1, v1)

            u1 = map(u + uStep, 0, uMax, 0, 1)

            texture.push(u1, v1)

            u1 = map(u, 0, uMax, 0, 1)

            v1 = map(v + vStep, 0, vMax, 0, 1)

            texture.push(u1, v1)

            u1 = map(u + uStep, 0, uMax, 0, 1)

            v1 = map(v, 0, vMax, 0, 1)

            texture.push(u1, v1)

            v1 = map(v + vStep, 0, vMax, 0, 1)

            texture.push(u1, v1)

            u1 = map(u, 0, uMax, 0, 1)

            v1 = map(v + vStep, 0, vMax, 0, 1)

            texture.push(u1, v1)

        }

    }

    return texture;

}

function map(val, f1, t1, f2, t2) {

    let m;

    m = (val - f1) \* (t2 - f2) / (t1 - f1) + f2

    return Math.min(Math.max(m, f2), t2);

}

function Cornucopia(u, v) {

    const k = 0.25

    const p = 0.15

    const m = 0.1

    let x = (Math.E \*\* (m \* u) + Math.E \*\* (p \* u) \* Math.cos(v)) \* Math.cos(u)

    let y = (Math.E \*\* (m \* u) + Math.E \*\* (p \* u) \* Math.cos(v)) \* Math.sin(u)

    let z = Math.E \*\* (p \* u) \* Math.sin(v)

    return { x: x \* k, y: y \* k, z: z \* k }

}