МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
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

ВАРІАНТ №9

Виконав студент 5-го курсу
ІАТЕ групи ТР-31мп
Золотько В.В.
Перевірив: Демчишин А.А.

Київ – 2023

Завдання:

1. Нанесіть текстуру на поверхню з практичного завдання #2.
2. Реалізувати масштабування текстури (координати текстури) масштабування/обертання навколо вказаної користувачем точки - непарні варіанти **реалізують масштабування**, парні варіанти реалізують обертання
3. Повинна бути реалізована можливість переміщення точки вздовж простору поверхні (u,v) за допомогою клавіатури. Наприклад, клавіші A і D переміщують точку вздовж параметра u, а клавіші W і S переміщують точку вздовж параметра v.

Теорія:

WebGL (Web Graphics Library) - це технологія, що надає можливість вбудовувати тривимірну графіку у веб-браузери без необхідності встановлення додаткових плагінів чи розширень. Вона базується на мові програмування JavaScript та використовує специфікації OpenGL ES 2.0, які визначають API для роботи з тривимірною графікою.

Ця технологія дозволяє упроваджувати апаратно-прискорену 3D графіку у веб-сторінки без необхідності використовувати спеціальні плагіни веб-браузера на будь-якій платформі, що підтримує OpenGL або OpenGL ES. Технічно це буде прив'язкою скриптів JavaScript до функцій, визначених в бібліотеках OpenGL ES 2.0, реалізовану на рівні браузера.

Особливості WebGL:

- WebGL підтримує використання шейдерів - програм, що виконуються на графічних процесорах, для створення різноманітних ефектів та обробки графічної інформації.
- WebGL підтримує використання текстур для нанесення зображень на поверхні 3D-моделей, а також буферів для ефективного обміну даними між JavaScript і GPU.
- WebGL вбудований безпосередньо в сучасні веб-браузери, що дозволяє запускати 3D-графіку без необхідності встановлення додаткових плагінів чи розширень.

Шейдер (Shader) - це програма, яка використовується для розрахунків графічних ефектів в графічних програмах, зазвичай в області комп'ютерної графіки. Шейдери використовуються для контролю процесу відображення графіки на графічному процесорі (GPU).

Використання шейдерів дозволяє розробникам створювати складні та реалістичні графічні ефекти, забезпечуючи високий рівень контролю над візуалізацією графічних об'єктів. Шейдери широко використовуються в ігровій індустрії, віртуальній реальності, симуляціях та інших областях, де потрібна висока якість графіки.

Існує два типи шейдерів:

- Вершинний шейдер (Vertex Shader) - Вершинні шейдери застосовуються для кожної вершини, що запускається в програмованому конвеєрі. Основна мета — перетворити геометрію на координати екранного простору, щоб піксельний шейдер міг растеризувати зображення. Вершинні шейдери можуть змінювати ці позиційні координати для виконання деформації сітки. Вони також можуть отримувати додаткову інформацію від сітки, включаючи нормалі, дотичні та координати текстур. Потім вершинний

шейдер записує у вихідні регістри; записані значення потім інтерполюються по вершинах у піксельних шейдерах. Вершинні шейдери не можуть створювати вершини.

- Фрагментні (піксельні) шейдери (Pixel Shader/Fragment) - Піксельні шейдери застосовуються до кожного пікселя, відображеного на екрані. Піксельний шейдер очікує введення від інтерпольованих значень вершин, які потім використовує для растеризації зображення. Піксельні шейдери можуть створити величезний діапазон ефектів, пов’язаних із кольором окремих пікселів, таких як заломлення світла, попіксельне освітлення або відображення.

Текстура – це зображення, що використовується для накладання на геометричні об'єкти з метою створення враження реалістичності та деталізації.

Масштабування в WebGL відбувається через застосування матриці масштабування до координат вершин об'єкта. Основний принцип полягає в тому, що кожну вершину об'єкта множать на матрицю масштабування, яка визначає, наскільки розширити чи стиснути об'єкт по кожній з трьох осей (X, Y, Z).

Для здійснення потрібно:

- Матриця масштабування є 4x4 матрицею, де головною частиною є блок 3x3, який відповідає за масштабування по трьох осях (X, Y, Z)
- Матрицю масштабування передають у вершинний шейдер. У шейдері вона використовується для множення координат кожної вершини об'єкта, змінюючи їх розміри відповідно до масштабу.
- У вершинному шейдері кожну вершину об'єкта множать на матрицю масштабування. Результат цього множення буде новою позицією вершини після масштабування.

Реалізація

Функція, яка задає поверхню:

```
function processSurfaceEquations(u, v) {
    const A = 0.25;
    const B = 0.25;
    const C = 0.125;
       const x = A * deg2rad(u) * Math.sin(deg2rad(u)) *
Math.cos(deg2rad(v));
       const y = B * deg2rad(u) * Math.cos(deg2rad(u)) *
Math.cos(deg2rad(v));
    const z = -C * deg2rad(u) * Math.sin(deg2rad(v));
    return { x, y, z };

}
```

Для створення текстури використовується функція, що викликається, при завантажені елементу на сторінку відображення:

`let texture = gl.createTexture()` – відповідає за свторення об'єкту текстури;

Функції, що відповідають за налаштування параметрів текстури, фільтрація, тощо:

```
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);
```

Завантаження зображення в текстуру. В об'єкт image класу Image передається джерело (посилання на зображення) і після зв'язується з текстурою, після цього викликається функція draw, що відповідає за рендеринг сцени:

```
    const image = new Image();
    image.crossOrigin = 'anonymus';
    image.src = "https://i.ibb.co/TKM7Mt1/lol.jpg";
    image.onload = () => {
        gl.bindTexture(gl.TEXTURE_2D, texture);
         gl.texImage2D(gl.TEXTURE_2D,0,gl.RGBA,gl.RGBA,gl.UNSIG
NED_BYTE,image);
        console.log("imageLoaded")
        draw()
    }
```

Для здійснення масштабування і зсуву, потрібно було змінити вершинний шейдер. Додати матрицю масштабування і зсуву для трьох вимірних зображень:

```
arr[0][0] = S; arr[0][ 1] = 0.0; arr[0][ 2] = 0.0; arr[0][ 3] = 0.0;
arr[1][ 0] = 0.0; arr[1][ 1] = S; arr[1][ 2] = 0.0; arr[1][ 3] = 0.0;
arr[2][ 0] = 0.0; arr[2][ 1] = 0.0; arr[2][ 2] = S; arr[2][ 3] = 0.0;
arr[3][ 0] = 0.0; arr[3][ 1] = 0.0; arr[3][ 2] = 0.0; arr[3][ 3] = 1.0;

 arr[0][0] = 1.0; arr[0][ 1] = 0.0; arr[0][ 2] = 0.0; arr[0][ 3] = 0.0;
arr[1][ 0] = 0.0; arr[1][ 1] = 1.0; arr[1][ 2] = 0.0; arr[1][ 3] = 0.0;
arr[2][ 0] = 0.0; arr[2][ 1] = 0.0;  arr[2][ 2] = 1.0; arr[2][ 3] = 0.0;
arr[3][ 0] = t.x; arr[3][ 1] = t.y; arr[3][ 2] = 0.0; arr[3][ 3] = 1.0;
```

В Vertex Shader ці матриці, використовуються для зміни координат текстури перед використанням їх у подальших обчисленнях:

vec4 tr1 = translation(-transl) * vec4(texture, 0.0, 1.0);

vec4 tr2 = scaling(scale) * tr1;

vec4 tr3 = translation(transl) * tr2;

textureV = tr3.xy;

texture - це координати текстури перед масштабуванням, texture – це нові координати текстури після масштабування чи зсуву.

Для зсуву на клавіатурі, використовуються клавіші на комп’ютері, числа означають ASCII код клавіш W, A, S, D:

```
window.onkeydown = (e) => {
    if (e.keyCode == 87) {
        trU = Math.min(trU + 0.02, 1);
    }
    else if (e.keyCode == 65) {
        trV = Math.max(trV - 0.02, 0);
    }
    else if (e.keyCode == 83) {
        trU = Math.max(trU - 0.02, 0);
    }
    else if (e.keyCode == 68) {
        trV = Math.min(trV + 0.02, 1);
    }
}
```

Інструкція

Запуск програми, з поверхньою, текстурою та з налаштуваннями справа:

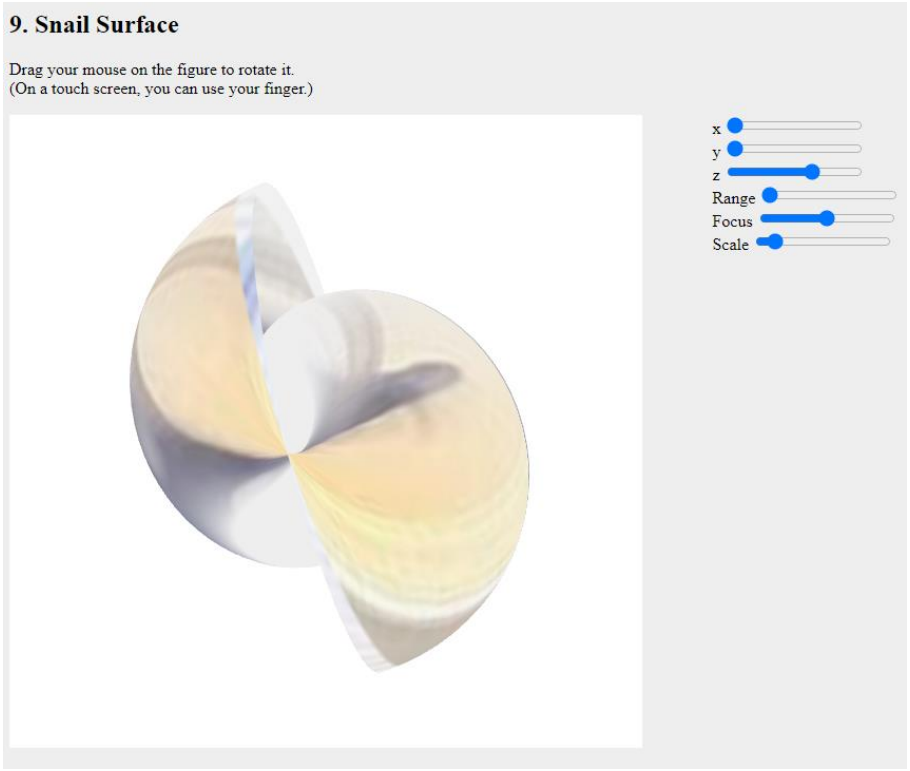

Рисунок 1 – Поверхня за замовчуванням

Для зміни масштабу, потрібно у слайдері Scale змінити параметри, утримуючи ліву кнопку миші і проводячи її вліво, чи вправо

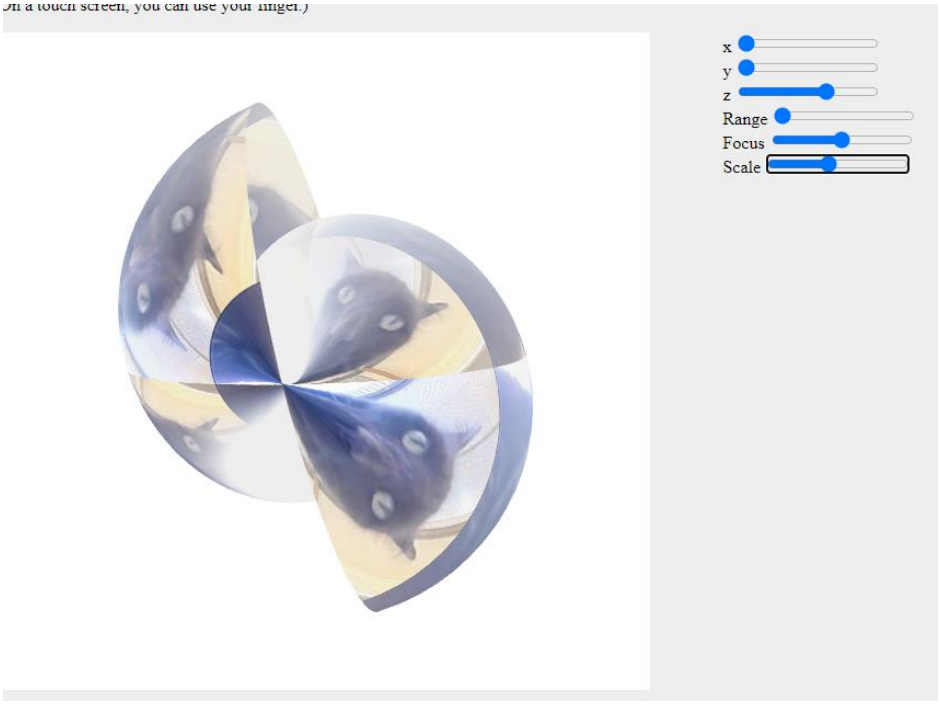

Рисунок 2 – зміна параметру Scale

Якщо користувач хоче здійснити зсув, наприклад в центр поверхні, йому потрібно затиснути клавішу D:

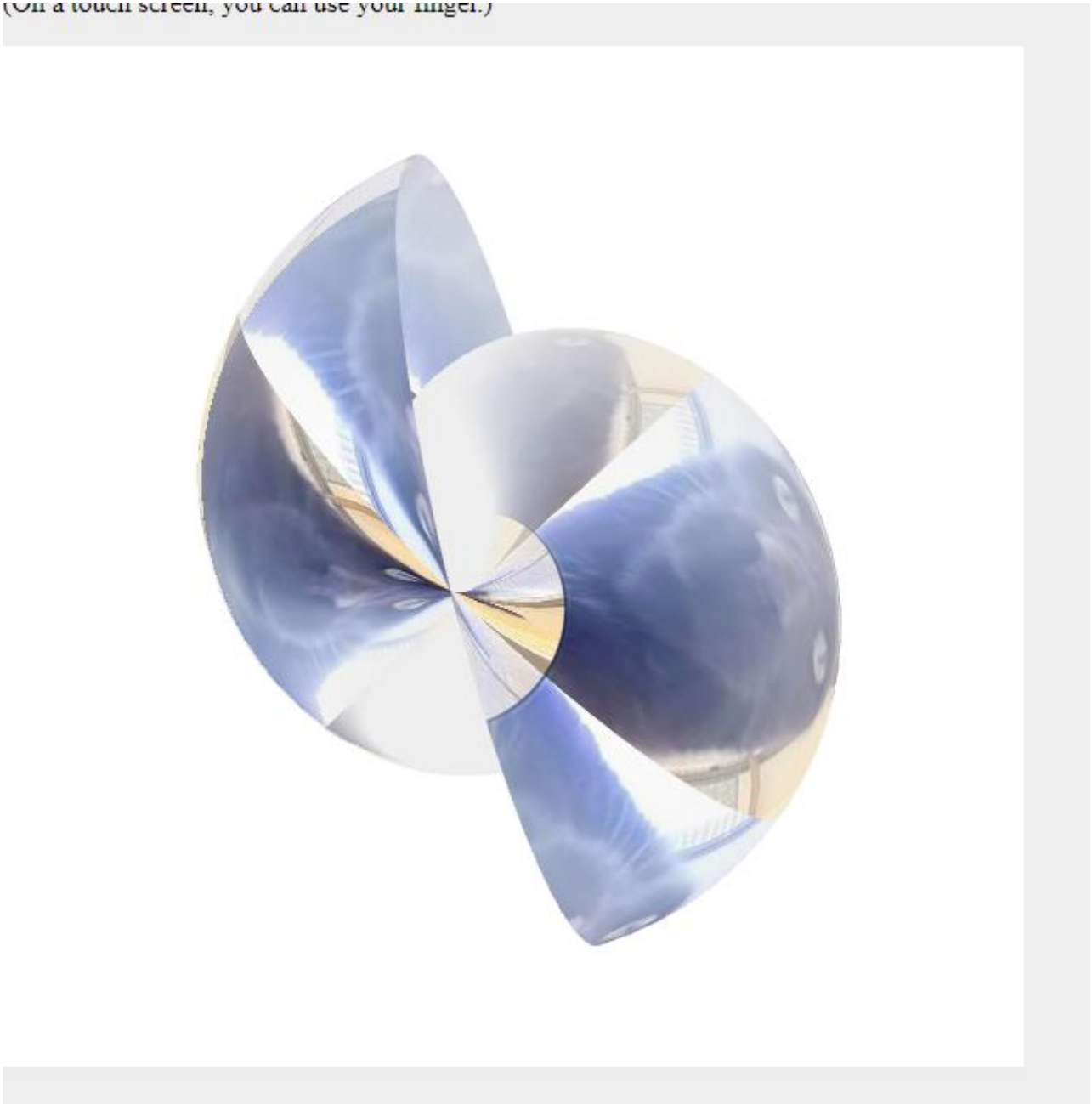

Рисунок 3 – Зсув відносно центру

Вихідний код

```
function draw() {
    gl.clearColor(1,1,1,1);
    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 inversion = m4.inverse(modelViewProjection);
    let transposedModel = m4.transpose(inversion);

    gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false,
modelViewProjection);
    gl.uniformMatrix4fv(shProgram.iModelMatrixNormal, false, transposedModel );

    gl.uniform4fv(shProgram.iColor, [1, 1, 0, 1]);
    let z = document.getElementById('z').value
    let x = document.getElementById('x').value
    let y = document.getElementById('y').value
    gl.uniform3fv(shProgram.iLightPosition, [x, y, z]);

    gl.uniform3fv(shProgram.iLightDirection, [1, -1, -1]);
    let f = document.getElementById('f').value
    let r = document.getElementById('r').value
    let s = document.getElementById('s').value
    gl.uniform1f(shProgram.iRange, r);
    gl.uniform1f(shProgram.iFocus, f);
    gl.uniform1f(shProgram.iScale, s);
    gl.uniform2fv(shProgram.iTranslateTo, [trU, trV])

    surface.Draw();
}
function processSurfaceEquations(u, v) {
    const A = 0.25;
    const B = 0.25;
    const C = 0.125;
    const x = A * deg2rad(u) * Math.sin(deg2rad(u)) * Math.cos(deg2rad(v));
    const y = B * deg2rad(u) * Math.cos(deg2rad(u)) * Math.cos(deg2rad(v));
    const z = -C * deg2rad(u) * Math.sin(deg2rad(v));
    return { x, y, z };

}
```

```
function CreateSurfaceData()
{
    let vertexList = [];
    let normalList = [];
    let textureList = [];
    let step = 5;
    let delta = 0.001;

    for (let u = -180; u <= 180; u += step) {
        for (let v = 0; v <= 360; v += step) {

            let v1 = processSurfaceEquations(v, u);
            let v2 = processSurfaceEquations(v + step, u);
            let v3 = processSurfaceEquations(v, u + step);
            let v4 = processSurfaceEquations(v + step, u + step);
            vertexList.push(v1.x, v1.y, v1.z);
            vertexList.push(v2.x, v2.y, v2.z);
            vertexList.push(v3.x, v3.y, v3.z);

            vertexList.push(v3.x, v3.y, v3.z);
            vertexList.push(v2.x, v2.y, v2.z);
            vertexList.push(v4.x, v4.y, v4.z);


            let n1 = CalculateNormal(u, v, delta);
            let n2 = CalculateNormal(u, v + step, delta);
            let n3 = CalculateNormal(u + step, v, delta);
            let n4 = CalculateNormal(u + step, v + step, delta)

            normalList.push(...n1, ...n2, ...n3, ...n3, ...n2, ...n4);

            textureList.push(v / 360, u / 360 + 0.5);
            textureList.push((v + step) / 360, u / 360 + 0.5);
            textureList.push(v / 360, (u + step) / 360 + 0.5);
            textureList.push(v / 360, (u + step) / 360 + 0.5);
            textureList.push((v + step) / 360, u / 360 + 0.5);
            textureList.push((v + step) / 360, (u + step) / 360 + 0.5);
        }
    }


    return { vertices: vertexList, normal: normalList, textures: textureList };
}
```