Міністерство освіти і науки України
Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Кафедра цифрових технологій в енергетиці

Розрахункова графічна робота
з дисципліни «Візуалізація графічної та геометричної інформації»
ВАРІАНТ №23

Виконав:
студент 5-го курсу
групи ТР-31мп
Позняк В.О.

Перевірив:
Демчишин А.А.

Київ – 2023

Завдання

1. Нанести текстуру на поверхню з практичного завдання №2.
2. Реалізувати масштабування текстури навколо визначеної користувачем точки
3. Надати можливість переміщати точку вздовж поверхні (u,v) за допомогою клавіатури: клавіші A і D переміщують точку вздовж параметра u, а клавіші W і S переміщують точку вздовж параметра v.

Теорія

WebGL - це відкритий стандарт, який дозволяє розробникам веб-додатків створювати та відтворювати 3D-графіку в веб-браузері без використання плагінів. WebGL базується на OpenGL ES, стандарті для мобільної та вбудованої графіки.

WebGL працює за допомогою графічного процесора (GPU) комп'ютера. GPU - це спеціалізований мікропроцесор, який призначений для обробки графічних даних. WebGL дозволяє розробникам передавати дані GPU за допомогою JavaScript.

Процес рендерингу 3D-графіки в WebGL можна розділити на кілька етапів:

1. Введення геометрії. На цьому етапі розробник передає GPU дані про 3D-об'єкти, які потрібно відобразити. Ці дані можуть включати координати вершин, нормалі, текстурні координати та інші параметри.
2. Рендеринг вершин. На цьому етапі GPU перетворює дані про вершини в 2D-координати пікселів. Цей процес називається вершинним шейдингом.
3. Рендеринг пікселів. На цьому етапі GPU визначає, як забарвити кожен піксель. Цей процес називається піксельним шейдингом.

Основні компоненти WebGL

WebGL включає в себе кілька основних компонентів:

- Контекст WebGL - це об'єкт, який надає доступ до API WebGL. Для створення контексту WebGL потрібно викликати метод getContext() елемента <canvas>.

- Буфери даних - це об'єкти, які зберігають дані, які потрібно передати GPU. WebGL підтримує кілька типів буферів даних, в тому числі буфери вершин, буфери індексів та буфери текстур.
- Шейдери - це фрагменти коду, які виконуються GPU для рендерингу 3D-графіки.

Широке застосування WebGL

WebGL використовується в широкому спектрі веб-додатків, включаючи:

- 3D-ігри
- 3D-презентації
- 3D-моделювання
- 3D-графіка в веб-браузері

WebGL дозволяє розробникам створювати захоплюючі та реалістичні 3D-інтерфейси, які можна використовувати в будь-якому веб-додатку.

WebGL відкриває широкі можливості для створення вражаючих тривимірних веб-додатків, від ігор і віртуальної реальності до візуалізації даних та інтерактивного мультимедіа. Ця технологія забезпечує високий рівень контролю над графікою та взаємодією користувача, що робить її потужним інструментом для розробки веб-застосунків.

Деталі виконання

Реалізовуємо функцію для завантаження фото текстури з мережі інтернет за посиланням та налаштовуємо параметри текстури:

```
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);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);

    let image = new Image();
    image.crossOrigin = 'anonymous';
    image.src = "https://images.pexels.com/photos/168442/pexels-photo-
168442.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1";
    image.onload = () => {
        gl.bindTexture(gl.TEXTURE_2D, texture);
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
image);

        draw();
    }
}
```

Обчислюємо точки поверхні та точки текстури в циклі:

```
for (let z = 0; z < b; z += 0.5) {
    for (let beta = 0; beta < 2 * Math.PI; beta += 0.2) {
        const p1 = createPoint(beta, z);
        const p2 = createPoint(beta, z + 0.5);
        const p3 = createPoint(beta + 0.2, z);
        const p4 = createPoint(beta + 0.2, z + 0.5);

        vertexList.push(p1.x, p1.y, p1.z);

        vertexList.push(p2.x, p2.y, p2.z);

        vertexList.push(p3.x, p3.y, p3.z);

        vertexList.push(p4.x, p4.y, p4.z);

        const texturePoint1 = [z / b, beta / (2 * Math.PI)]
        const texturePoint2 = [(z + 0.5) / b, beta / (2 * Math.PI)]
        const texturePoint3 = [z / b, (beta + 0.2) / (2 * Math.PI)]
        const texturePoint4 = [(z + 0.5) / b, (beta + 0.2) / (2 * Math.PI)]

        textureList.push(...texturePoint1, ...texturePoint2,
...texturePoint3, ...texturePoint4);
    }
}
```

Далі створюємо буфер для зберігання координат текстури :

```
gl.bindBuffer(gl.ARRAY_BUFFER, this.iTextureBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(texture), gl.STREAM_DRAW);

gl.bindBuffer(gl.ARRAY_BUFFER, this.iTextureBuffer);
gl.vertexAttribPointer(shProgram.iAttribTexture, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(shProgram.iAttribTexture);
```

Після цього у вершинному шейдері реалізовуємо масштабування текстури:

```
const vertexShaderSource = `
attribute vec3 vertex;
attribute vec2 textureCoord;
uniform vec2 pTexture;
uniform vec3 pTranslate;
uniform float fScale;
varying vec2 texInterp;
uniform mat4 ModelViewProjectionMatrix, ModelViewInverseTranspose;
mat4 translatePoint(vec3 point) {
  return mat4(
    vec4(1.0, 0.0, 0.0, point.x),
    vec4(0.0, 1.0, 0.0, point.y),
    vec4(0.0, 0.0, 1.0, point.z),
    vec4(0.0, 0.0, 0.0, 1.0)
  );
}

mat4 scale(float fScale) {
    return mat4(
        vec4(fScale, 0.0, 0.0, 0.0),
        vec4(0.0, fScale, 0.0, 0.0),
        vec4(0.0, 0.0, 1.0, 0.0),
        vec4(0.0, 0.0, 0.0, 1.0)
    );
}
void main() {
    if(fScale < 0.0){
        vec4 point = vec4(vertex, 1.0) * translatePoint(pTranslate);
        gl_Position = ModelViewProjectionMatrix * point;
    } else {
        vec4 vertPos4 = ModelViewProjectionMatrix * vec4(vertex, 1.0);
        gl_Position = vertPos4;
    }
    mat4 matTranslateToZero = translatePoint(-vec3(pTexture, 0.0));
    mat4 matScale = scale(fScale);
    mat4 matTranslateBackToPoint = translatePoint(vec3(pTexture, 0.0));
    vec4 translatedToZero =  matTranslateToZero * vec4(texCoord, 0.0, 0.0);
    vec4 scaled = translatedToZero * matScale;
    vec4 translatedBackToPoint = scaled * matTranslateBackToPoint;
    texInterp = vec2(translatedBackToPoint.x, translatedBackToPoint.y);
    gl_PointSize = 20.0;
}`;
```

Використання програми

На рисунку 1 зображений початковий екран програми

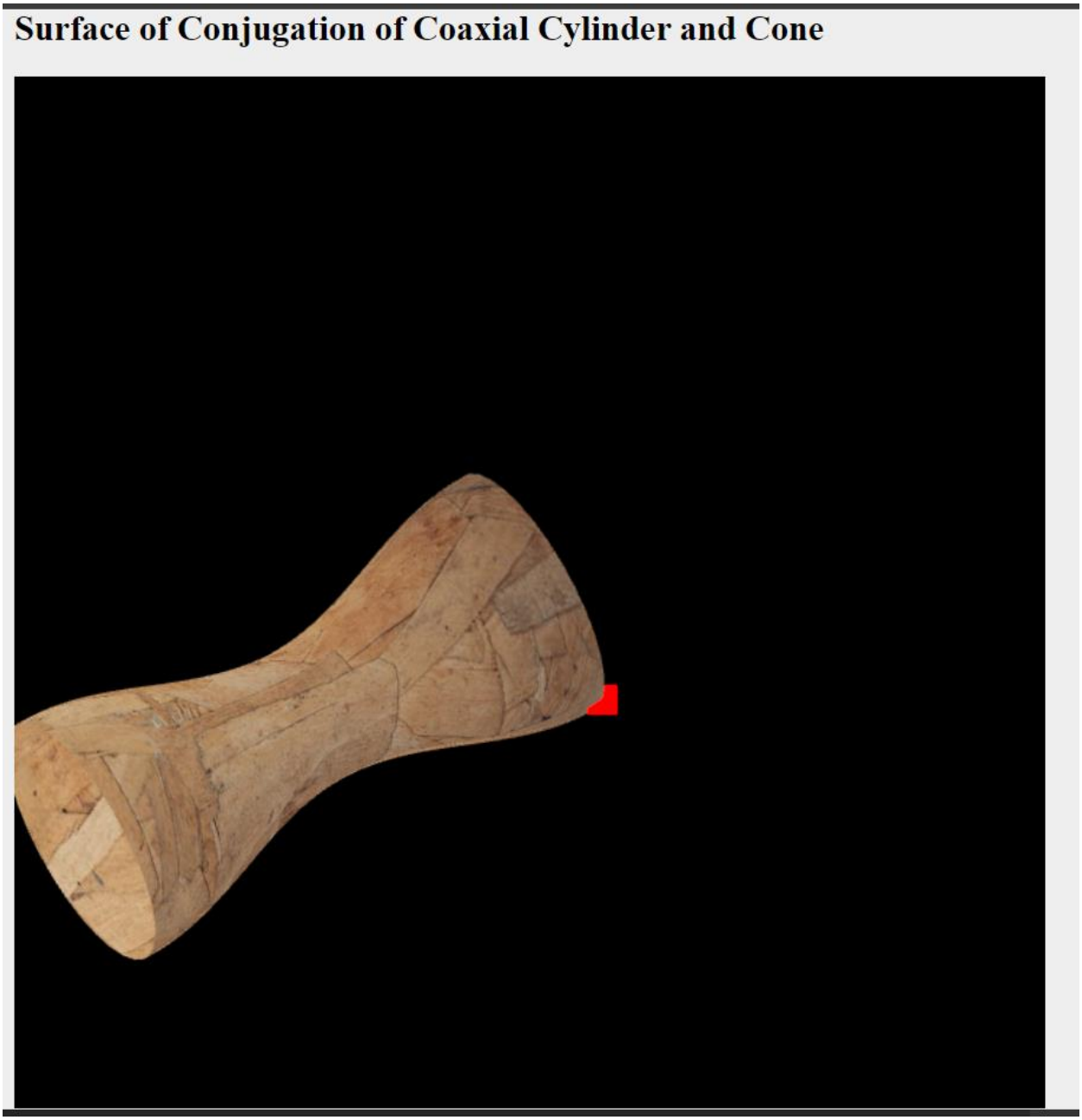

Рисунок 1 Початковий екран програми

За допомогою клавіші A рухаємо точку по параметру u, результат переміщення точки зображений на рисунку 2.

Рисунок 2 Рух по параметру u

За допомогою клавіші W рухаємо точку по параметру v, результат переміщення точки зображений на рисунку 3.

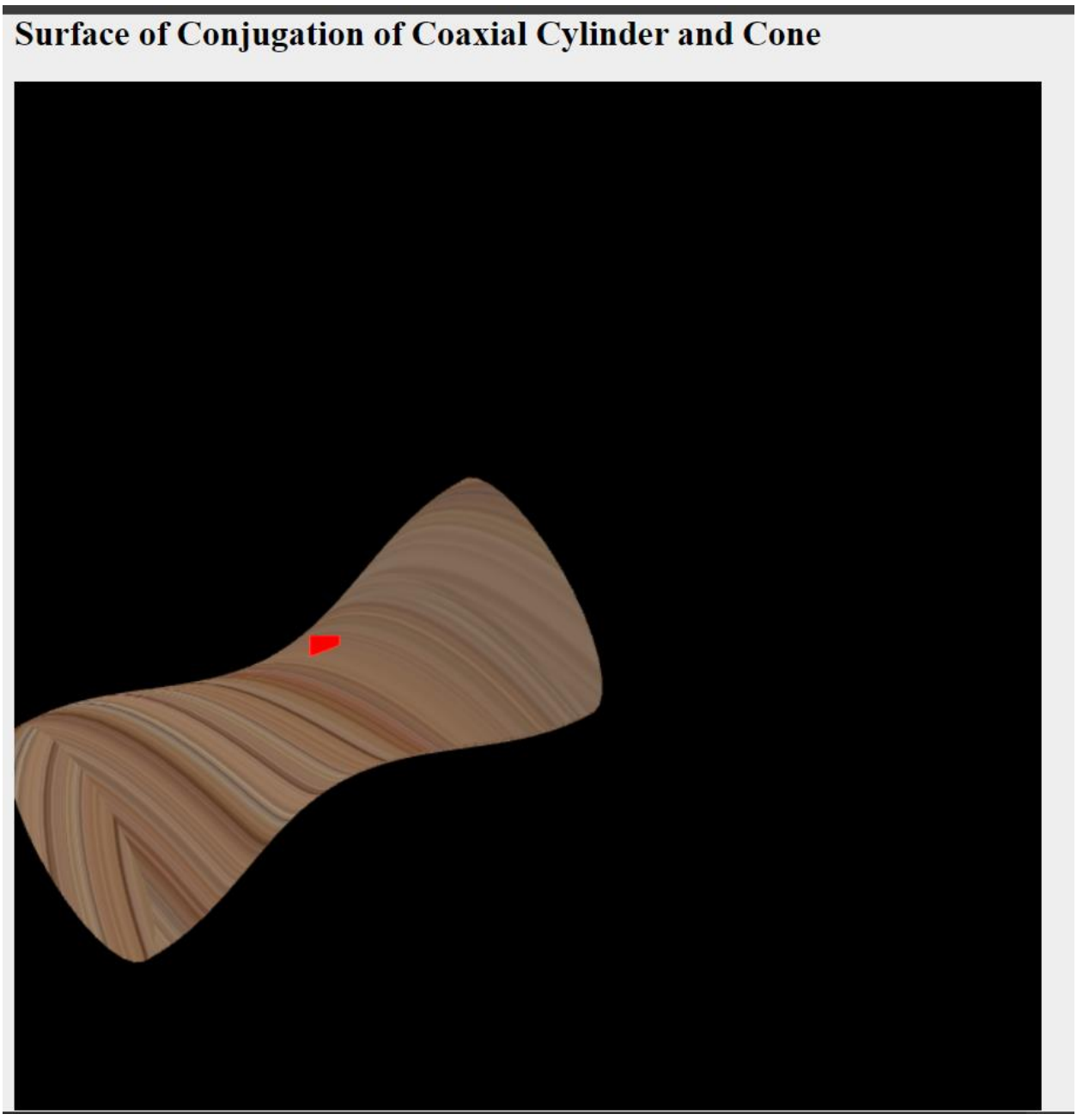

Рисунок 3 Рух по параметру v

Код програми

```
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);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);

    let image = new Image();
    image.crossOrigin = 'anonymous';
    image.src = "https://images.pexels.com/photos/168442/pexels-photo-
168442.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1";
    image.onload = () => {
        gl.bindTexture(gl.TEXTURE_2D, texture);
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
image);

        draw();
    }
}

// Constructor
function Model(name) {
    this.name = name;
    this.iVertexBuffer = gl.createBuffer();
    this.iTextureBuffer = gl.createBuffer();
    this.count = 0;

    this.BufferData = function(vertices, texture) {
        gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STREAM_DRAW);

        this.count = vertices.length / 3;

        gl.bindBuffer(gl.ARRAY_BUFFER, this.iTextureBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(texture),
gl.STREAM_DRAW);
    }

    this.Draw = function() {

        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.iTextureBuffer);
        gl.vertexAttribPointer(shProgram.iAttribTexture, 2, gl.FLOAT, false,
0, 0);
        gl.enableVertexAttribArray(shProgram.iAttribTexture);

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

```
    this.PointBuffer = function(point) {
        gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(point),
gl.DYNAMIC_DRAW);
    }

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

        gl.drawArrays(gl.POINTS, 0, 1);
    }
}
function CreateSurfaceData() {
    let vertexList = [];
    let textureList = [];
    let scale = 1;
    let R2 = 4;
    let R1 = 1.3 * R2;
    const a = R2 - R1;
    const c = 4 * R1;
    const b = c;

    //  Surface of Conjugation of Coaxial Cylinder and Cone

    for (let z = 0; z < b; z += 0.5) {
        for (let beta = 0; beta < 2 * Math.PI; beta += 0.2) {
            const p1 = createPoint(beta, z);
            const p2 = createPoint(beta, z + 0.5);
            const p3 = createPoint(beta + 0.2, z);
            const p4 = createPoint(beta + 0.2, z + 0.5);

            vertexList.push(p1.x, p1.y, p1.z);

            vertexList.push(p2.x, p2.y, p2.z);

            vertexList.push(p3.x, p3.y, p3.z);

            vertexList.push(p4.x, p4.y, p4.z);

            const texturePoint1 = [z / b, beta / (2 * Math.PI)]
            const texturePoint2 = [(z + 0.5) / b, beta / (2 * Math.PI)]
            const texturePoint3 = [z / b, (beta + 0.2) / (2 * Math.PI)]
            const texturePoint4 = [(z + 0.5) / b, (beta + 0.2) / (2 *
Math.PI)]

            textureList.push(...texturePoint1, ...texturePoint2,
...texturePoint3, ...texturePoint4);
        }
    }

    return { vertices: vertexList,  texture: textureList };
}
```