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 №18

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

Київ – 2023

Завдання

1. Використовуючи програмний код з другої лабораторної роботи, нанести текстуру на поверхню.
2. Реалізувати обертання текстури навколо точки.
3. Надати можливість користувачу змінювати точку обертання за допомогою клавіш: W та S зміщення за параметром u; A та D зміщення за параметром v.
4. Розроблений програмний код завантажити на віддалений GitHub репозиторій у гілку CGW.

Теорія

WebGL (Web Graphics Library) - це JavaScript API для рендерингу високопродуктивної інтерактивної 3D і 2D графіки в будь-якому сумісному веб-браузері без використання плагінів. WebGL робить це, представляючи API, який тісно пов'язаний з OpenGL ES 2.0, що може бути використаний в елементах HTML <canvas>. Ця відповідність дозволяє API використовувати переваги апаратного прискорення графіки, що надається пристроєм користувача.

Текстура - це об'єкт OpenGL, який містить одне або декілька зображень, що мають однаковий формат. Текстуру можна використовувати двома способами: вона може бути джерелом доступу до текстури з шейдеру або використовуватися як об'єкт рендерингу. Зображення - єдиний масив пікселів певної розмірності (1D, 2D або 3D), з певним розміром і певним форматом.

Текстура - це контейнер одного або декількох зображень. Але текстури не зберігають довільні зображення; текстура має певні обмеження на зображення, які вона може містити. Існує три визначальні характеристики текстури, кожна з яких визначає частину цих обмежень: тип текстури, розмір текстури та формат зображення, що використовується для зображень у текстурі. Тип текстури визначає розташування зображень всередині текстури. Розмір визначає розмір зображень у текстурі. А формат зображення визначає формат, у якому всі ці зображення мають спільний вигляд.

Розміри текстур мають обмеження на основі реалізації GL. Для одновимірних та двовимірних текстур максимальний розмір будь-якого виміру дорівнює GL_MAX_TEXTURE_SIZE. Для масивів текстур максимальна довжина масиву дорівнює GL_MAX_ARRAY_TEXTURE_ LAYERS. Для 3D-текстури жоден вимір не може бути більшим за GL_MAX_3D_TEXTURE_SIZE.

У цих межах розмір текстури може бути будь-яким. Однак, рекомендується дотримуватися степеня двійки для розмірів текстур, якщо тільки у вас немає нагальної потреби у використанні довільних розмірів.

UV-мапінг - це процес створення 2D-відображення 3D-об'єкта. UV-координати також відомі як координати текстури. U позначає горизонтальну вісь, а V - вертикальна вісь. Кожна UV-координата має відповідну точку у 3D-просторі, яка називається вершиною. Разом вершини утворюють ребра, ребра утворюють грані, грані утворюють багатокутники, а багатокутники утворюють поверхні.

Процедура завантаження текстури починається зі створення текстури об'єкта WebGL за допомогою виклику функції WebGL createTexture(). Потім можна за допомогою texImage2D() встановити суцільний колір текстури. Це робить текстуру одразу придатною для використання як суцільний колір, навіть якщо завантаження зображення може зайняти кілька хвилин.

Щоб завантажити текстуру з файлу зображення, створюється об'єкт Image і присвоюється значення src URL-адреси зображення, яке ми хочемо використати як текстуру. Далі необхідно визначити image.onload функцію, яка буде викликана після завершення завантаження зображення. У цей момент ми знову викликаємо texImage2D(), цього разу використовуючи зображення як джерело текстури. Після цього ми налаштовуємо фільтрацію та обгортання текстури на основі того, чи було завантажене зображення степенем 2 в обох вимірах, чи ні.

WebGL1 може використовувати не більше 2 текстур з фільтрацією, встановленою на NEAREST або LINEAR, і не може згенерувати міпмапу для них. Їхній режим обгортання також має бути встановлений на CLAMP_TO_EDGE. З іншого боку, якщо текстура має ступінь 2 в обох вимірах, WebGL може виконати якіснішу фільтрацію, використати mipmap і встановити режим обгортання REPEAT або MIRRORED_REPEAT.

Імплементація програмного коду

Для початку створимо буфер для зберігання координат текстури та напишемо функцію для переведення синтетичних параметрів поверхні в UV-координати.

Прив’язка буферу та його заповнення координатами текстури:

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

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

Для знаходження синтетичних координат було написано редаговано функцію для обчислення координат поверхні з другої лабораторної роботи. Її оновлений вигляд з обчисленням координат текстури:

```
function CreateSurfaceData() {
	let vertexList = [];
	let texturePoints = [];

	let angleStep = Math.PI / parameters.angleStep;

	for (let z = 0; z <= (parameters.a - parameters.zStep).toFixed(2); z = +(parameters.zStep + z).toFixed(2)) {
	for (let angle = 0; angle <= maxAngle - angleStep; angle += angleStep) {
		let u1 = z;
		let v1 = angle;
		let u2 = z;
		let v2 = angle + angleStep;
		let u3 = +(parameters.zStep + z).toFixed(2);
		let v3 = angle;
		let u4 = +(parameters.zStep + z).toFixed(2);
		let v4 = angle + angleStep;
		let p1 = calcVertPoint(u1, v1);
		let p2 = calcVertPoint(u2, v2);
		let p3 = calcVertPoint(u3, v3);
		let p4 = calcVertPoint(u4, v4);
		let uv1 = calcUVPoint(u1, parameters.a, v1, maxAngle);
		let uv2 = calcUVPoint(u2, parameters.a, v2, maxAngle);
		let uv3 = calcUVPoint(u3, parameters.a, v3, maxAngle);
		let uv4 = calcUVPoint(u4, parameters.a, v4, maxAngle);
		vertexList.push(...p1.transformVector(), ...p2.transformVector(),
		...p3.transformVector(), ...p4.transformVector());
		texturePoints.push(...uv1.transformVector(), ...uv2.transformVector(),
		...uv3.transformVector(), ...uv4.transformVector());
	}
```

```
        }
        return new SurfaceData(vertexList, texturePoints);
    }

    function calcVertPoint(z, angle) {
        let rZ = RZ(z);
        let x = X(rZ, angle);
        let y = Y(rZ, angle);
        return new Point(x, y, z);
    }

    function calcUVPoint(u, uMAx, v, vMax) {
        return new UVPoint(map(u, uMAx), map(v, vMax));
    }

    function map(val, max) {
        return val / max;
    }
```

Наступним кроком було написано функцію для завантаження текстури з віддаленого репозиторію GitHub:

```
    function LoadTexture() {
        var 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);

        var image = new Image();
        image.crossOrigin = 'anonymous';
        image.src                                                                                    =
"https://raw.githubusercontent.com/twistedmisted/surf-rev-pear/CGW/texture/water.png";
        image.onload = () => {
        gl.bindTexture(gl.TEXTURE_2D, texture);
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);

        draw();
        }
    }
```

Обертання текстури було реалізовано на GPU у вершинному шейдері:

```
    const vertexShaderSource = `
    attribute vec2 texCoord;
    uniform vec3 pTranslate;
    uniform vec2 pTexture;
    uniform float angleRad;
    varying vec2 texInterp;
```

```
mat4 translate(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 rotate(float angleRad) {
  float c = cos(angleRad);
  float s = sin(angleRad);

  return mat4(
    vec4(c, s, 0.0, 0.0),
    vec4(-s, c, 0.0, 0.0),
    vec4(0.0, 0.0, 1.0, 0.0),
    vec4(0.0, 0.0, 0.0, 1.0)
  );
}

void main() {
    ...
    vec4 translatedToZero =  matTranslateToZero * vec4(texCoord, 0.0, 0.0);
    vec4 rotatedByAngleRad = translatedToZero * matRotate;
    vec4 translatedBackToPoint = rotatedByAngleRad * matTranslateBackToPoint;
    texInterp = vec2(translatedBackToPoint.x, translatedBackToPoint.y);
    ...
}`;
```

Приклади роботи з програмою

Приклад фігури та нанесення текстури на неї з початковими параметрами зображено на рисунку 1.

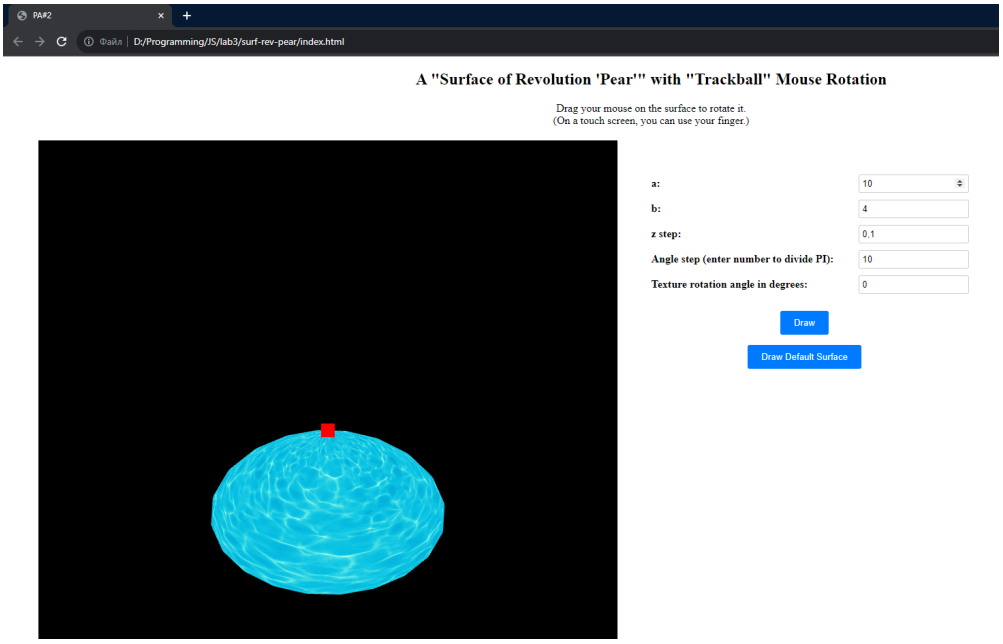

Рис. 1. – Фігура та текстура зображені з початковими параметрами

На рисунку 2 зображено зміну кута обертання (градусів на скільки виконувати обертання навколо точки) – 90 градусів.

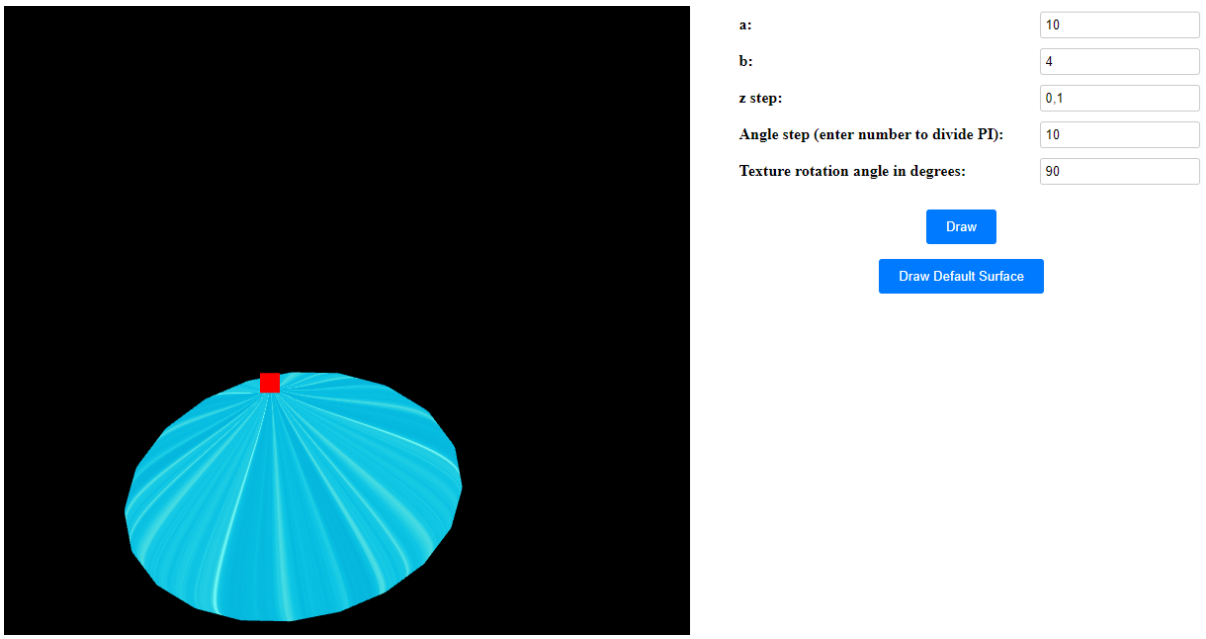

Рис. 2. – Зміна кута обертання рівному 90 градусів

На рисунку 3 зображено зміну точки обертання за допомогою клавіш WASD та кута обертання – 45 градусів.

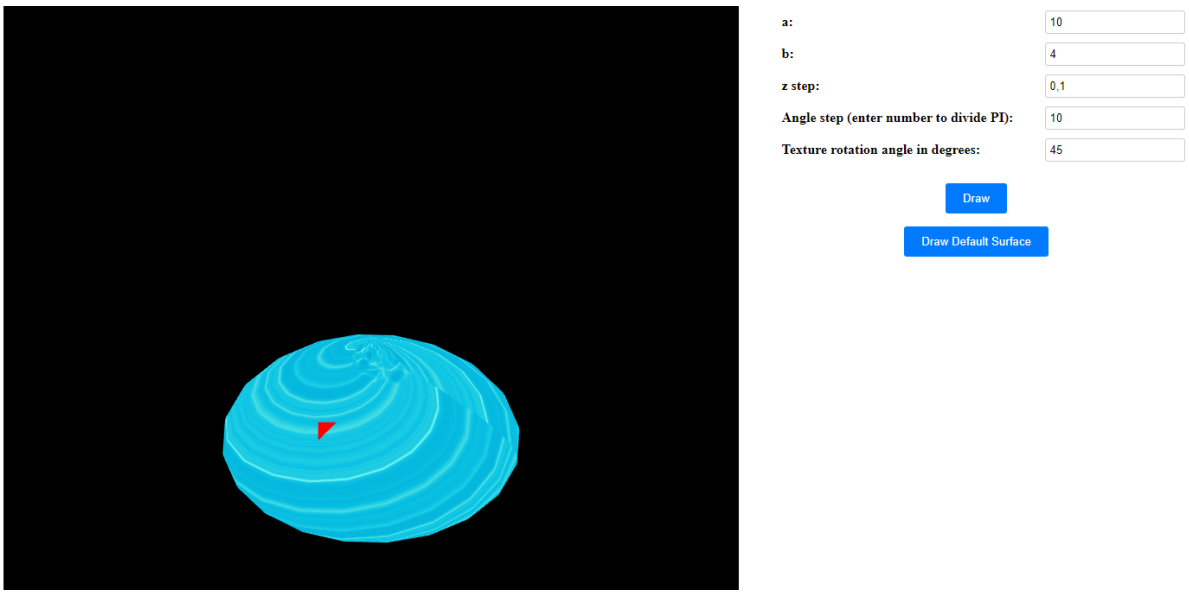

Рис. 3. – Обертання навколо зміщенної точки обертання та кутом обертання рівним 45 градусів

Приклад вихідного коду

```
class Point {
    constructor(x, y, z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    transformVector() {
        return [this.x, this.y, this.z];
    }
}

class UVPoint {
    constructor(u, v) {
        this.u = u;
        this.v = v;
    }

    transformVector() {
        return [this.u, this.v];
    }
}

class SurfaceData {
    constructor(vertexList, texturePoints) {
        this.vertexList = vertexList;
        this.texturePoints = texturePoints;
    }
}
function Model(name) {
    this.name = name;
    this.iVertexBuffer = gl.createBuffer();
    this.iTextureBuffer = gl.createBuffer();
    this.verticesLength = 0;
    this.textureLength = 0;

    this.BufferData = function(surfData) {
        gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(surfData.vertexList), gl.STREAM_DRAW);
        this.verticesLength = surfData.vertexList.length / 3;
        gl.bindBuffer(gl.ARRAY_BUFFER, this.iTextureBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(surfData.texturePoints), gl.STREAM_DRAW);
        this.textureLength = surfData.texturePoints.length / 2;
    }

    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.verticesLength);
        }

        this.PointBuffer = function(pointData) {
            gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
            gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(pointData), 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 texturePoints = [];
        let angleStep = Math.PI / parameters.angleStep;
         for (let z = 0; z <= (parameters.a - parameters.zStep).toFixed(2); z = +(parameters.zStep +
z).toFixed(2)) {
            for (let angle = 0; angle <= maxAngle - angleStep; angle += angleStep) {
                let u1 = z;
                let v1 = angle;
                let u2 = z;
                let v2 = angle + angleStep;
                let u3 = +(parameters.zStep + z).toFixed(2);
                let v3 = angle;
                let u4 = +(parameters.zStep + z).toFixed(2);
                let v4 = angle + angleStep;
                let p1 = calcVertPoint(u1, v1);
                let p2 = calcVertPoint(u2, v2);
                let p3 = calcVertPoint(u3, v3);
                let p4 = calcVertPoint(u4, v4);
                vertexList.push(...p1.transformVector(), ...p2.transformVector(),
                ...p3.transformVector(), ...p4.transformVector());
                let uv1 = calcUVPoint(u1, parameters.a, v1, maxAngle);
                let uv2 = calcUVPoint(u2, parameters.a, v2, maxAngle);
                let uv3 = calcUVPoint(u3, parameters.a, v3, maxAngle);
                let uv4 = calcUVPoint(u4, parameters.a, v4, maxAngle);
                texturePoints.push(...uv1.transformVector(), ...uv2.transformVector(),
                ...uv3.transformVector(), ...uv4.transformVector());
            }
        }
        return new SurfaceData(vertexList, texturePoints);
    }

    function initGL() {
        let prog = createProgram( gl, vertexShaderSource, fragmentShaderSource );
        shProgram = new ShaderProgram('Basic', prog);
        shProgram.Use();
        shProgram.iAttribVertex              = gl.getAttribLocation(prog, "vVertex");
        shProgram.iModelViewProjectionMatrix = gl.getUniformLocation(prog,
    "ModelViewProjectionMatrix");
```

```
    shProgram.iAttribTexture             = gl.getAttribLocation(prog, "texCoord");
    shProgram.iTMU                       = gl.getUniformLocation(prog, "tmu");
    shProgram.iTranslatePoint            = gl.getUniformLocation(prog, 'pTranslate');
    shProgram.iTexturePoint              = gl.getUniformLocation(prog, 'pTexture');
    shProgram.iAngleRad                  = gl.getUniformLocation(prog, 'angleRad');
    surface = new Model('Surface of Revolution "Pear"');
    initParameters();
    LoadTexture();
    setBufferData(surface);
    rotationPointModel = new Model('Rotation Point');
    rotationPointModel.PointBuffer([0.0, 0.0, 0.0]);
    gl.enable(gl.DEPTH_TEST);
}
```

Фрагментний шейдер:

```
// Fragment shader
const fragmentShaderSource = `
#ifdef GL_FRAGMENT_PRECISION_HIGH
   precision highp float;
#else
   precision mediump float;
#endif
varying vec2 texInterp;
uniform sampler2D tmu;
uniform float angleRad;
void main() {
    // Check if this is user point
    if(angleRad == -1.0){
      gl_FragColor = vec4(0.0, 0.0 , 0.0 , 0.0);
    } else {
      gl_FragColor = texture2D(tmu, texInterp);
    }
}`;
```