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

МЕТОДИ СИНТЕЗУ ВІРТУАЛЬНОЇ РЕАЛЬНОСТІ

Розрахунково-графічна робота
Варіант №3

Виконав:
студент 1-го курсу НН ІАТЕ
групи ТР-32мп
Білик Максим Олександрович

Перевірив: Демчишин А. А.

Київ – 2024

ЗАВДАННЯ

1. Повторно використати код з практичного завдання №2;
2. Реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні по колу протягом певного часу (поверхня залишається нерухомою, а джерело звуку рухається). Відтворюйте улюблену пісню у форматі mp3/ogg, маючи просторове розташування джерела звуку, кероване користувачем;
3. Візуалізувати положення джерела звуку за допомогою сфери;
4. Додати звуковий фільтр (використати інтерфейс BiquadFilterNode): смуговий фільтр. Додати елемент з прапорцем, який вмикає або вимикає фільтр. Налаштувати параметри фільтра на свій розсуд.

ТЕОРЕТИЧНІ ОСНОВИ

WebGL та WebAudio - це два важливих API, які розширюють можливості HTML5, додаючи підтримку 3D-графіки та аудіо обробки відповідно. WebGL, або Web Graphics Library, це JavaScript API, який використовується для рендерингу 2D та 3D графіки безпосередньо в браузері без використання плагінів.

WebGL (Web Graphics Library) – це технологія, яка дозволяє створювати 3D-графіку в браузері без необхідності встановлення додаткових плагінів чи розширень. Вона базується на OpenGL ES (Embedded Systems), із рядом доповнень для взаємодії з елементами веб-сторінки. За допомогою WebGL розробники можуть створювати візуально багаті та інтерактивні веб-додатки, ігри та візуалізації, використовуючи високоякісну 3D-графіку.

WebAudio API - це високорівневий JavaScript API для обробки та синтезу аудіо в веб-додатках. З його допомогою можна створювати комплексні звукові ефекти, музичні треки та інтерактивні аудіо додатки прямо в браузері. WebAudio HTML5 API – це інтерфейс програмування додатків, який дозволяє розробникам створювати, обробляти і відтворювати аудіо у веб-браузерах. Розглянемо ключові аспекти WebAudio API.

AudioContext є основним об'єктом у Web Audio API, призначеним для роботи зі звуком у веб-додатках. Він дозволяє ініціалізувати аудіо середовище, створювати та зв'язувати аудіо вузли, а також керувати аудіо ланцюгами для формування складних звукових ефектів. Серед можливостей AudioContext — створення джерел звуку (осцилятори, аудіо буферні джерела), обробка звуку через різні ефекти (фільтри, компресори, реверберatори), контроль, аналіз звуку за допомогою аналізаторів, планування та синхронізація аудіо подій, завантаження та декодування аудіо файлів, а також динамічні зміни параметрів аудіо обробки в режимі реального часу. AudioContext забезпечує всі необхідні інструменти для створення інтерактивного і захоплюючого звукового досвіду на веб-сторінках.

Audio Sources у Web Audio API використовуються для представлення аудіо джерел. Вони є початковими елементами аудіо-графа і постачають аудіо-дані, які потім обробляються та відтворюються. Після того, як аудіо-сигнал пройшов крізь різні етапи обробки, він може бути підключений до аудіо-призначення (наприклад, AudioContext.destination), де відтворюється на динаміки або записується у файл.

WebAudio API дозволяє точно синхронізувати аудіо події, забезпечуючи плавне відтворення та обробку аудіо. Окрім цього, наявна підтримка тривимірного звуку, що дозволяє розташовувати джерела звуку в тривимірному просторі для створення більш реалістичних аудіо ефектів, для чого використовується Panner.

PannerNode в Web Audio API використовується для позиціонування звуку в тривимірному просторі, створюючи ефекти просторового звуку. Він змінює стереопозицію аудіо сигналу, створюючи ілюзію звуку, що виходить з певної точки.

PannerNode дозволяє налаштовувати положення та напрямок джерела звуку відносно слухача, використовуючи різні моделі позиціонування для зміни гучності залежно від відстані (лінійна, зворотна, експоненційна). Також можна налаштувати конус орієнтації, що визначає, як звук затухає залежно від напряму. Це дозволяє створювати реалістичні звукові сцени в веб-додатках.

Смуговий фільтр - електронний фільтр, що пропускає сигнали в певному діапазоні (смузі) частот, і послаблює (вирізає) сигнали частот за межами цієї смуги. Наприклад, смуговий фільтр на 1800—1900 МГц пропускає тільки сигнали, частота яких лежить в інтервалі 1 800 ÷1 900 МГц. При цьому частота 1 800 МГц називається нижньою частотою зрізу, а 1 900 МГц — верхньою частотою зрізу.

ХІД РОБОТИ

Для роботи з аудіофайлами, в файл index.html додаємо наступні атрибути:

```
        <fieldset>
          <legend>Audio</legend>
          <label for="isFilterOn">Enable filter:</label>
          <input type="checkbox" id="isFilterOn" name="isFilterOn">
          <fieldset>
            <legend>Filter Q</legend>
            <input type="range" id="Q" min="-10" max="10" step="0.5" value="0"
onchange="getFilter()">
          </fieldset>
          <audio id="audio" controls loop crossorigin="anonymous">
            <source src="music.mp3" type="audio/mpeg" />
            Your browser does not support the audio element.
          </audio>
        </fieldset>
```

Чекбокс <input type="checkbox"> з id "isFilterOn" дозволяє увімкнути або вимкнути фільтр звуку bandpass, який заданий по варіанту. Поле <input type="range"> з id "Q" дозволяє користувачу вибирати ширину смуги. Ширина стає вужчою зі збільшенням значення Q. у діапазоні від -10 до 10. Аудіофайл вказаний за допомогою елементу <audio>. Цей елемент містить панель керування аудіо, що дозволяє користувачу керувати відтворенням (play/pause), звуком та прокруткою файлу.

Наступним кроком налаштуємо аудіо. Функція, яка відповідає за створення аудіо контексту та підключення аудіо файлу:

```
function createAudio() {
  audio = document.getElementById("audio");

  audio.addEventListener("pause", () => {
    audioContext.resume();
  });

  audio.addEventListener("play", () => {
    if (!audioContext) {
      audioContext = new (window.AudioContext || window.webkitAudioContext)();
```

```
        audioSource = audioContext.createMediaElementSource(audio);

        getPanner();
        getFilter();

        audioSource.connect(audioPanner);
        audioFilter.connect(audioContext.destination);

        audioContext.resume();
        audio.play();
      }
    });
}
```

В функції getPanner() створюємо та налаштовуємо PannerNode:

```
function getPanner() {
    audioPanner = audioContext.createPanner();
    audioPanner.panningModel = "HRTF";
    audioPanner.distanceModel = "linear";
}
```

За варіантом необхідно додати фільтр, тому в функції getFilter() створюємо фільтр bandpass і налаштовуємо його. Необхідно додати чекбокс для включення та виключення фільтру:

```
function getFilter() {
    if (!audioContext) {
        audioContext = new (window.AudioContext || window.webkitAudioContext)();
    }

    if (!audioFilter) {
        audioFilter = audioContext.createBiquadFilter();
        audioFilter.type = 'bandpass';
        audioFilter.frequency.value = 1000;
    }

    audioFilter.Q.value = parseFloat(document.getElementById("Q").value);

    var isFilterOn = document.getElementById("isFilterOn");

    isFilterOn.addEventListener("change", function () {
        if (isFilterOn.checked) {
```

```
            audioSource.connect(audioFilter);
            audioFilter.connect(audioContext.destination);
        } else {
            audioSource.disconnect(audioFilter);
            audioSource.connect(audioContext.destination);
        }
    });
}
```

Для візуалізації джерела звуку створюємо сферу:

```
sphereRotationAngle += 0.0001;
let spherePos = getSpherePosition(sphereRotationAngle, 7);
let sphereModelView = m4.translate(matAccumLeft, spherePos.x, spherePos.y, spherePos.z);
gl.uniform1i(shProgram.isSphere, true);
gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, sphereModelView);
sphere.DrawSphere();
gl.uniform1i(shProgram.isSphere, false);
gl.clear(gl.DEPTH_BUFFER_BIT);
gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccumRight);
gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projectionRight);
gl.colorMask(false, true, true, false);
surface.Draw();
sphereModelView = m4.translate(matAccumRight, spherePos.x, spherePos.y, spherePos.z);


function getSpherePosition(angle, radius) {
    return {
        x: radius * Math.cos(angle),
        y: 0, // Adjust Y if rotation should occur in a different plane
        z: radius * Math.sin(angle)
    };
}
```

Джерело звуку переміщуємо за допомогою Panner:

```
if (audioPanner) {
        audioPanner.setPosition(
            spherePos.x,
            spherePos.y,
            spherePos.z
        );
    }
```

ІНСТРУКЦІЯ КОРИСТУВАЧА

Після запуску програми бачимо головний інтерфейс (рис. 1). На ньому представлене відео з веб-камери, зображена поверхня та зелена сфера, що візуалізує джерело звуку аудіо файлу. Сфера рухається навколо фігури. З правого боку розташована панель, де є можливість в реальному часу змінювати параметри.

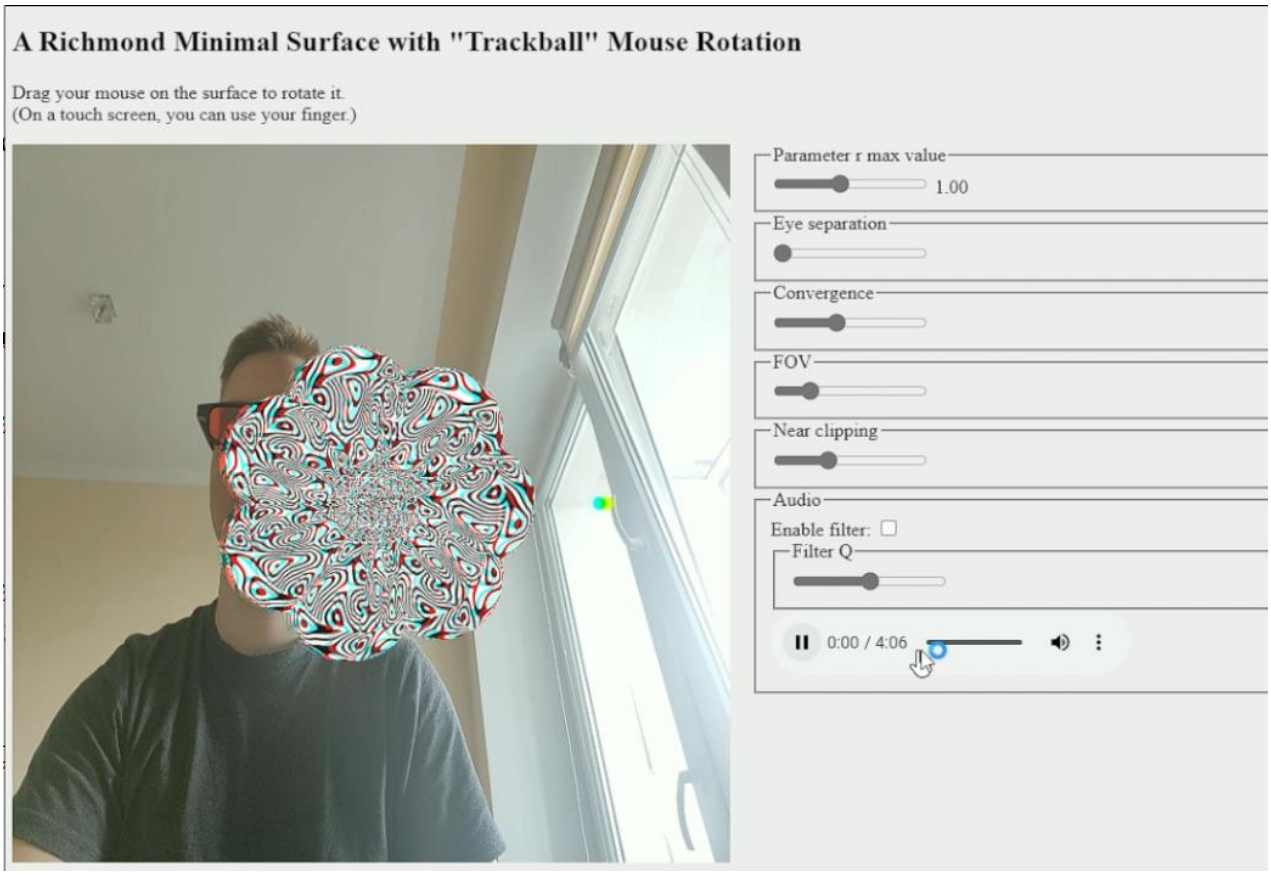

Рисунок 1. – Інтерфейс програми

В блоці керування аудіо є можливість вмикати та вимикати аудіо файл, перемотувати його, змінювати гучність, а також вмикати та вимикати фільтр та змінювати його значення. Використовуючи чекбокс можемо вимкнути фільтр:

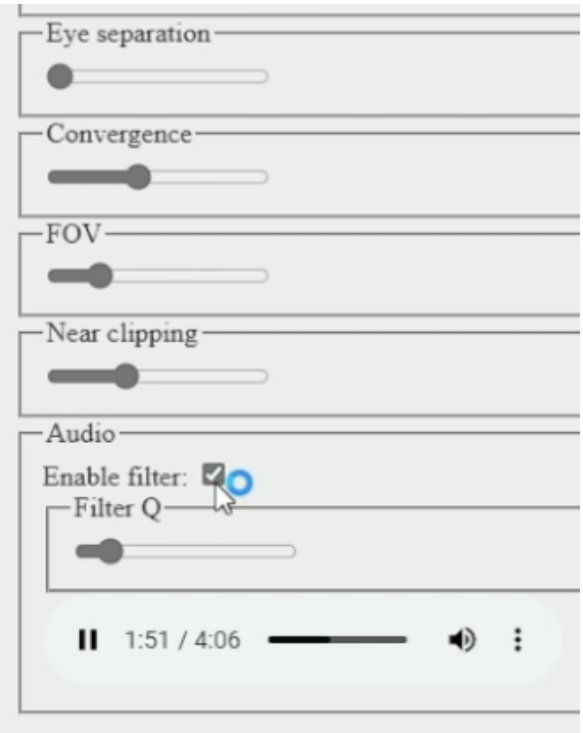

Рисунок 2. – Панель керування

Сфера рухається навколо геометричного центру фігури:

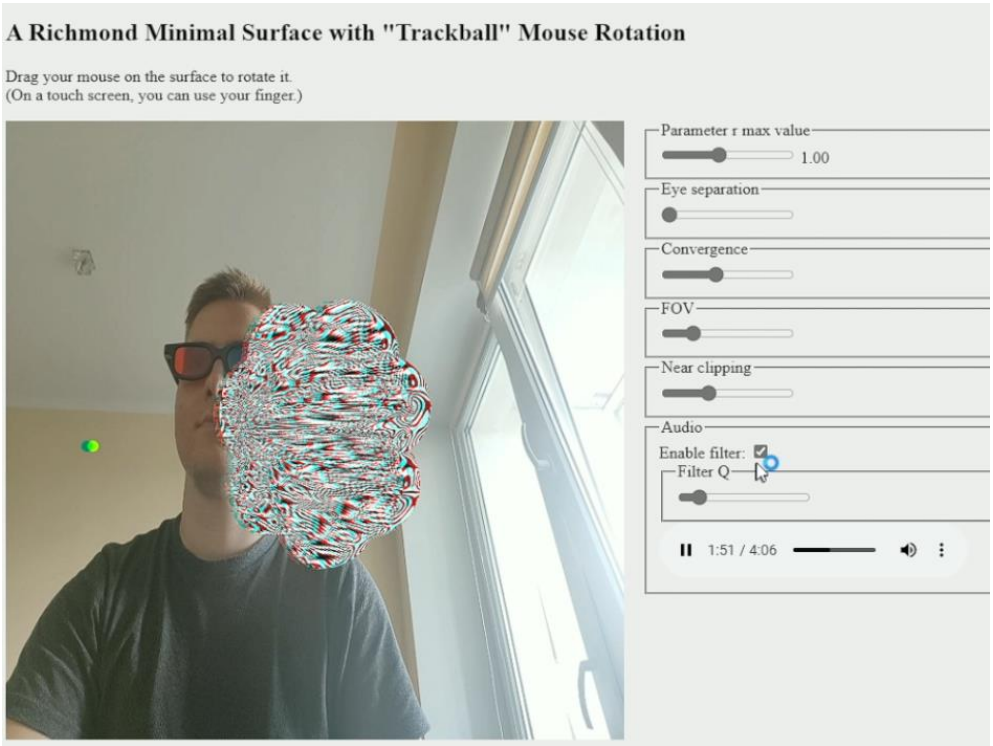

Рисунок 3. – Інтерфейс програми

Окрім цього надає можливість змінювати масштабування, а також керувати параметрами для анагліфічного стереозображення.

ВИХІДНИЙ КОД

```
'use strict';

let gl;                         // The webgl context.
let surface;                    // A surface model
let shProgram;                  // A shader program
let spaceball;                  // A SimpleRotator object that lets the user rotate the view by mouse.
let maxR = 1;
let stereoCamera;

let texture;

let webCameraTexture;
let webCameraVideo;
let webCamera;

let audio = null;
let audioContext;
let audioSource;
let audioPanner;
let audioFilter;

let isFilterOn = false;

let sphere;
let sphereCenter = { x: 0, y: 0, z: 0 };

let sensor;
let reading = { x: 0, y: 0, z: 0 };

function deg2rad(angle) {
    return angle * Math.PI / 180;
}

let previousTimestamp = 0;
let sphereRotationAngle = 0;



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

    this.BufferData = function (vertices, textCoords) {

        gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
```

```
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STREAM_DRAW);

        gl.bindBuffer(gl.ARRAY_BUFFER, this.iTextCoordBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textCoords), gl.STREAM_DRAW);

        this.count = vertices.length / 3;
    }

    this.BufferDataSphere = 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.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.iTextCoordBuffer);
        gl.vertexAttribPointer(shProgram.iAttribTextCoord, 2, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(shProgram.iAttribTextCoord);

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

    this.DrawSphere = 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.TRIANGLE_STRIP, 0, this.count);
    }
}

// Constructor
function ShaderProgram(name, program) {

    this.name = name;
    this.prog = program;

    // Location of the attribute variable in the shader program.
    this.iAttribVertex = -1;

    this.iAttribTextCoord = -1;

    // Location of the uniform matrix representing the combined transformation.
    this.iModelViewProjectionMatrix = -1;
```

```
    this.iModelViewMatrix = -1;

    this.iProjectionMatrix = -1;

    this.iTMU = -1;

    this.Use = function () {
        gl.useProgram(this.prog);
    }
}


/* Draws a colored cube, along with a set of coordinate axes.
 * (Note that the use of the above drawPrimitive function is not an efficient
 * way to draw with WebGL.  Here, the geometry is so simple that it doesn't matter.)
 */
function draw(timestamp) {
    gl.clearColor(0, 0, 0, 1);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    let projection = m4.orthographic(0, 1, 0, 1, -1, 1);
    let modelView = spaceball.getViewMatrix();
    let rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0);
    let matAccum = m4.multiply(rotateToPointZero, modelView);
    let noRot = m4.multiply(rotateToPointZero, [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]);

    const stereoCamera = {
        eyeSeparation: parseFloat(document.getElementById("eyeSeparation").value),
        convergence: parseFloat(document.getElementById("convergence").value),
        aspectRatio: gl.canvas.width / gl.canvas.height,
        fov: parseFloat(document.getElementById("fov").value),
        near: parseFloat(document.getElementById("near").value),
        far: 50.0,
    };

    if (audioPanner) {
        audioPanner.setPosition(
            spherePos.x,
            spherePos.y,
            spherePos.z
        );
    }

    gl.uniform1i(shProgram.iTMU, 0);

    let projectionLeft = applyLeftFrustum(stereoCamera);
    let projectionRight = applyRightFrustum(stereoCamera);

    let translateToLeft = m4.translation(-0.03, 0, -20);
    let translateToRight = m4.translation(0.03, 0, -20);
```

```
    gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, noRot);
    gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projection);

    gl.bindTexture(gl.TEXTURE_2D, webCameraTexture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, webCameraVideo);
    webCamera.Draw();

    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.clear(gl.DEPTH_BUFFER_BIT);

    let matAccumLeft = m4.multiply(translateToLeft, matAccum);
    let matAccumRight = m4.multiply(translateToRight, matAccum);

    gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccumLeft);
    gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projectionLeft);
    gl.colorMask(true, false, false, false);
    surface.Draw();


    sphereRotationAngle += 0.0001;
    let spherePos = getSpherePosition(sphereRotationAngle, 7);

    let sphereModelView = m4.translate(matAccumLeft, spherePos.x, spherePos.y, spherePos.z);

    gl.uniform1i(shProgram.isSphere, true);
    gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, sphereModelView);
    sphere.DrawSphere();
    gl.uniform1i(shProgram.isSphere, false);

    gl.clear(gl.DEPTH_BUFFER_BIT);

    gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccumRight);
    gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projectionRight);
    gl.colorMask(false, true, true, false);
    surface.Draw();

    sphereModelView = m4.translate(matAccumRight, spherePos.x, spherePos.y, spherePos.z);

    gl.uniform1i(shProgram.isSphere, true);
    gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, sphereModelView);
    sphere.DrawSphere();
    gl.uniform1i(shProgram.isSphere, false);

    gl.colorMask(true, true, true, true);

    window.requestAnimationFrame(draw);


}


function getSpherePosition(angle, radius) {
```

```
    return {
        x: radius * Math.cos(angle),
        y: 0, // Adjust Y if rotation should occur in a different plane
        z: radius * Math.sin(angle)
    };
}



function CreateSurfaceData() {
    let vertexList = [];
    let normalList = [];
                let textCoordList = [];
    let step = 0.03;
    let delta = 0.001;

    for (let u = -3.5 * Math.PI; u <= 3.5 * Math.PI; u += step) {
        for (let v = 0.005 * Math.PI; v < Math.PI / 2; v += step) {

            let v1 = equations(u, v);
            let v2 = equations(u, v + step);
            let v3 = equations(u + step, v);
            let v4 = equations(u + step, v + 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(v2.x, v2.y, v2.z);
            vertexList.push(v4.x, v4.y, v4.z);
            vertexList.push(v3.x, v3.y, v3.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.x, n1.y, n1.z);
            normalList.push(n2.x, n2.y, n2.z);
            normalList.push(n3.x, n3.y, n3.z);

            normalList.push(n2.x, n2.y, n2.z);
            normalList.push(n4.x, n4.y, n4.z);
            normalList.push(n3.x, n3.y, n3.z);

            let t1 = CalculateTextCoord(u, v);
            let t2 = CalculateTextCoord(u, v + step);
            let t3 = CalculateTextCoord(u + step, v);
            let t4 = CalculateTextCoord(u + step, v + step);

            textCoordList.push(t1.u, t1.v);
            textCoordList.push(t2.u, t2.v);
```

```
            textCoordList.push(t3.u, t3.v);

            textCoordList.push(t2.u, t2.v);
            textCoordList.push(t4.u, t4.v);
            textCoordList.push(t3.u, t3.v);
        }
    }

    return { vertices: vertexList, normal: normalList, textCoords: textCoordList };
}

function CalculateNormal(u, v, delta) {
    let currentPoint = equations(u, v);
    let pointR = equations(u + delta, v);
    let pointTheta = equations(u, v + delta);

    let dg_du = {
        x: (pointR.x - currentPoint.x) / delta,
        y: (pointR.y - currentPoint.y) / delta,
        z: (pointR.z - currentPoint.z) / delta
    };

    let dg_dv = {
        x: (pointTheta.x - currentPoint.x) / delta,
        y: (pointTheta.y - currentPoint.y) / delta,
        z: (pointTheta.z - currentPoint.z) / delta
    };

    let normal = cross(dg_du, dg_dv);

    normalize(normal);

    return normal;
}

function cross(a, b) {
    let x = a.y * b.z - b.y * a.z;
    let y = a.z * b.x - b.z * a.x;
    let z = a.x * b.y - b.x * a.y;
    return { x: x, y: y, z: z }
}

function normalize(a) {
    var b = Math.sqrt(a.x * a.x + a.y * a.y + a.z * a.z);
    a.x /= b;
    a.y /= b;
    a.z /= b;
}


function CalculateTextCoord(u, v) {
```

```
    u = (u - 0.25)/(maxR - 0.25);
    v = v / 2*Math.PI;

    return {u, v};
}

function equations(u, v) {
    let C = 2;
    let fiU = -u / (Math.sqrt(C + 1)) + Math.atan(Math.sqrt(C + 1) * Math.tan(u));
    let aUV = 2 / (C + 1 - C * Math.pow(Math.sin(v), 2) * Math.pow(Math.cos(u), 2));
    let rUV = (aUV / Math.sqrt(C)) * Math.sqrt((C + 1) * (1 + C * Math.pow(Math.sin(u), 2))) * Math.sin(v);

    let x = rUV * Math.cos(fiU);
    let y = rUV * Math.sin(fiU);
    let z = (Math.log(Math.tan(v / 2)) + aUV * (C + 1) * Math.cos(v)) / Math.sqrt(C);

    return { x: x, y: y, z: z };
}

function applyLeftFrustum(stereoCamera) {
    let { eyeSeparation, convergence, aspectRatio, fov, near, far } = stereoCamera;
    let top = near * Math.tan(fov / 2);
    let bottom = -top;

    let a = aspectRatio * Math.tan(fov / 2) * convergence;
    let b = a - eyeSeparation / 2;
    let c = a + eyeSeparation / 2;

    let left = (-b * near) / convergence;
    let right = (c * near) / convergence;

    return m4.orthographic(left, right, bottom, top, near, far);
}

function applyRightFrustum(stereoCamera) {
    let { eyeSeparation, convergence, aspectRatio, fov, near, far } = stereoCamera;
    let top = near * Math.tan(fov / 2);
    let bottom = -top;

    let a = aspectRatio * Math.tan(fov / 2) * convergence;
    let b = a - eyeSeparation / 2;
    let c = a + eyeSeparation / 2;

    let left = (-c * near) / convergence;
    let right = (b * near) / convergence;

    return m4.orthographic(left, right, bottom, top, near, far);
}


// Function to update the surface with the new max value of parameter r
function updateSurface() {
```

```
    maxR = parseFloat(document.getElementById("paramR").value);

    let data = CreateSurfaceData(maxR);
    surface.BufferData(data.vertices, data.textCoords);

    document.getElementById("currentMaxR").textContent = maxR.toFixed(2);

    draw();
}

function updateWebCamera() {
    draw();
    window.requestAnimationFrame(updateWebCamera);
}

function LoadTexture() {

    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);

    var image = new Image();
    image.crossOrigin = "anonymous";
    image.src = "https://i.ibb.co/DpJ0WzF/texture6.png";
    image.addEventListener('load', () => {
        gl.bindTexture(gl.TEXTURE_2D, texture);
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);

        draw();
    }
    );
}

function LoadWebCameraTexture() {

    webCameraTexture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, webCameraTexture);
    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);
}

function getWebCamera() {
    return new Promise((resolve) =>
        navigator.mediaDevices
            .getUserMedia({ video: true, audio: false })
            .then((s) => resolve(s))
    );
};
```

```
function createSphereData() {
  let radius = 0.1;
  let vertexList = [];
  let step = 10;
  let max = 360;
  for (let u = 0; u <= max; u += step) {
    for (let v = 0; v <= max; v += step) {
      let x1 = sphereCenter.x + (radius * Math.cos(deg2rad(u)) * Math.sin(deg2rad(v)));
      let y1 = sphereCenter.y + (radius * Math.sin(deg2rad(u)) * Math.sin(deg2rad(v)));
      let z1 = sphereCenter.z + (radius * Math.cos(deg2rad(v)));

      let x2 = sphereCenter.x + (radius * Math.cos(deg2rad(u + step)) * Math.sin(deg2rad(v + step)));
      let y2 = sphereCenter.y + (radius * Math.sin(deg2rad(u + step)) * Math.sin(deg2rad(v + step)));
      let z2 = sphereCenter.z + (radius * Math.cos(deg2rad(v + step)))

      vertexList.push(x1, y1, z1);
      vertexList.push(x2, y2, z2);
    }
  }
  return vertexList;
}


function createAudio() {
  audio = document.getElementById("audio");

  audio.addEventListener("pause", () => {
    audioContext.resume();
  });

  audio.addEventListener("play", () => {
    if (!audioContext) {
      audioContext = new (window.AudioContext || window.webkitAudioContext)();
      audioSource = audioContext.createMediaElementSource(audio);

      getPanner();
      getFilter();

      audioSource.connect(audioPanner);
      audioFilter.connect(audioContext.destination);

      audioContext.resume();
      audio.play();
    }
  });
}

function getPanner() {
  audioPanner = audioContext.createPanner();
  audioPanner.panningModel = "HRTF";
  audioPanner.distanceModel = "linear";
}
```

```
function getFilter() {
    if (!audioContext) {
        audioContext = new (window.AudioContext || window.webkitAudioContext)();
    }

    if (!audioFilter) {
        audioFilter = audioContext.createBiquadFilter();
        audioFilter.type = 'bandpass';
        audioFilter.frequency.value = 1000;
    }

    audioFilter.Q.value = parseFloat(document.getElementById("Q").value);

    var isFilterOn = document.getElementById("isFilterOn");

    isFilterOn.addEventListener("change", function () {
        if (isFilterOn.checked) {
            audioSource.connect(audioFilter);
            audioFilter.connect(audioContext.destination);
        } else {
            audioSource.disconnect(audioFilter);
            audioSource.connect(audioContext.destination);
        }
    });
}



/* Initialize the WebGL context. Called from init() */
function initGL() {
    let prog = createProgram(gl, vertexShaderSource, fragmentShaderSource);

    shProgram = new ShaderProgram('Basic', prog);
    shProgram.Use();

    shProgram.iAttribVertex = gl.getAttribLocation(prog, "vertex");
    shProgram.iModelViewProjectionMatrix = gl.getUniformLocation(prog, "ModelViewProjectionMatrix");
    shProgram.iModelViewMatrix = gl.getUniformLocation(prog, "ModelViewMatrix");
    shProgram.iProjectionMatrix = gl.getUniformLocation(prog, "ProjectionMatrix");
    shProgram.iAttribTextCoord = gl.getAttribLocation(prog, "textCoord");
    shProgram.iTMU = gl.getUniformLocation(prog, "tmu");
    shProgram.isSphere = gl.getUniformLocation(prog, "isSphere");

    surface = new Model('Surface');
    let data = CreateSurfaceData();
    surface.BufferData(data.vertices, data.textCoords);

    sphere = new Model('Sphere');
    sphere.BufferDataSphere(createSphereData());

    webCamera = new Model('WebCamera');
```

```
    webCamera.BufferData(
        [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,],
        [1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1]
    );

    LoadTexture();
    LoadWebCameraTexture();

    gl.enable(gl.DEPTH_TEST);
}


/* Creates a program for use in the WebGL context gl, and returns the
 * identifier for that program.  If an error occurs while compiling or
 * linking the program, an exception of type Error is thrown.  The error
 * string contains the compilation or linking error.  If no error occurs,
 * the program identifier is the return value of the function.
 * The second and third parameters are strings that contain the
 * source code for the vertex shader and for the fragment shader.
 */
function createProgram(gl, vShader, fShader) {
    let vsh = gl.createShader(gl.VERTEX_SHADER);
    gl.shaderSource(vsh, vShader);
    gl.compileShader(vsh);
    if (!gl.getShaderParameter(vsh, gl.COMPILE_STATUS)) {
        throw new Error("Error in vertex shader:  " + gl.getShaderInfoLog(vsh));
    }
    let fsh = gl.createShader(gl.FRAGMENT_SHADER);
    gl.shaderSource(fsh, fShader);
    gl.compileShader(fsh);
    if (!gl.getShaderParameter(fsh, gl.COMPILE_STATUS)) {
        throw new Error("Error in fragment shader:  " + gl.getShaderInfoLog(fsh));
    }
    let prog = gl.createProgram();
    gl.attachShader(prog, vsh);
    gl.attachShader(prog, fsh);
    gl.linkProgram(prog);
    if (!gl.getProgramParameter(prog, gl.LINK_STATUS)) {
        throw new Error("Link error in program:  " + gl.getProgramInfoLog(prog));
    }
    return prog;
}


/**
 * initialization function that will be called when the page has loaded
 */
function init() {
    let canvas;
    try {
        canvas = document.getElementById("webglcanvas");
        gl = canvas.getContext("webgl");
```

```
    if (!gl) {
      throw "Browser does not support WebGL";
    }

    webCameraVideo = document.createElement("video");
    webCameraVideo.setAttribute("autoplay", true);
    window.vid = webCameraVideo;

    getWebCamera().then((stream) => (webCameraVideo.srcObject = stream));

  }
  catch (e) {
    document.getElementById("canvas-holder").innerHTML =
      "<p>Sorry, could not get a WebGL graphics context.</p>" + e;
    return;
  }
  try {
    initGL();  // initialize the WebGL graphics context
  }
  catch (e) {
    document.getElementById("canvas-holder").innerHTML =
      "<p>Sorry, could not initialize the WebGL graphics context: " + e + "</p>";
    return;
  }

  spaceball = new TrackballRotator(canvas, draw, 0);

  updateSurface();
  updateWebCamera();

  createAudio();
}
```