МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НТУУ «Київський політехнічний інститут ім. Ігоря Сікорського»

Інститут атомної та теплової енергетики
Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота

З дисципліни «Методи синтезу віртуальної реальності»

Варіант 13

Виконав:
Студент 5-го курсу, ІАТЕ
групи ТР-21мп
Нагорний Є.П.
Викладач:
доцент Демчишин А.А.

Київ - 2023

Тема: Реалізація руху звукового джерела за допомогою осяжного інтерфейсу з використанням візуалізації та фільтрації звуку WebAudio HTML5 API.

Завдання:

Використовуючи код лабораторної роботи з практичного завдання No2 реалізувати обертання джерела звуку навколо геометричного центру поверхні за допомогою матеріального інтерфейсу. Відтворювати пісню в форматі mp3, даючи змогу користувачу змінювати просторове розташування джерела звуку. Джерело звуку повинне бути візуалізовано у вигляді сфери. Додати звуковий фільтр низьких частот використавши інтерфейс BiquadFilterNode. В інтерфейс користувача додати елемент прапорця, що дає змогу вмикати та вимикати відповідний фільтр.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Web Audio API - це потужний інструмент, який дозволяє відтворювати, створювати та керувати звуком, додавати звукові ефекти, створювати візуалізацію аудіо та багато іншого за допомогою JavaScript у браузері.

Він надає набір інтерфейсів і об’єктів, які дозволяють створювати, модифікувати та маршрутизувати аудіосигнали в реальному часі. Одним із ключових аспектів API веб-аудіо є його здатність обробляти аудіо та керувати ним за допомогою модульного підходу, який дозволяє створювати складні конвеєри обробки аудіо.

Об'єкт AudioContext - це основа Web Audio, який має методи для створення інших об'єктів для обробки аудіо.

Основні об'єкти які використовуються для робити Web Audio:

- AudioContext - аудіо контекст для обробки аудіо.
- AnalyserNode - об'єкт який аналізує частоти звуку.
- AudioListener - об'єкт який вказує положення слухача звуку.
- BiquadFilterNode - фільтр звуку низького порядку
- ConvolverNode - об'єкт для створення ефекту реверберації звуку.
- GainNode - об'єкт для регулювання рівня гучності звукового сигналу.
- MediaElementAudioSourceNode - об'єкт для отримання звуку з медіа елемента.
- OscillatorNode - об'єкт для створення звукової хвилі.
- StereoPannerNode - стерео панорамування звуку.

Якщо детальніше розглянути головний об’єкт - AudioContext представляє граф обробки аудіо та діє як центральний центр для створення та підключення аудіо вузлів. Він служить точкою входу для доступу та керування аудіофункціями, які надає Web Audio API. Створюючи екземпляр AudioContext, розробники отримують доступ до широкого спектру методів і властивостей для керування відтворенням аудіо, маршрутизацією та ефектами.

Інтерфейс MediaElementAudioSourceNode є джерелом звуку, що міститься в елементах HTML5 <audio> або <video> . Його можна підключити до інших аудіо вузлів для подальшої обробки або маршрутизації. Використовуючи MediaElementSourceNode, розробники можуть включати існуючі медіа-елементи в екосистему Web Audio API і застосовувати різні звукові ефекти або маніпуляції.

StereoPannerNode відповідає за просторове позиціонування та панорамування звуку. Він імітує тривимірне аудіо, регулюючи положення, орієнтацію та швидкість аудіоджерела у віртуальному 3D-просторі. Цей об’єкт дозволяє розробникам створювати ефект занурення в аудіо, коли звук виходить із певних напрямків, створюючи відчуття глибини та руху.

Хід роботи

Відповідно до завдання, розрахунково графічна робота повинна бути виконана на основі другого практичного завдання в якому потрібно було реалізувати обертання поверхні « Continuous Topographic Surface of Cassini» (рис. 1) на смартфоні.

```
211
212  function funct(u, z, a) {
213      return Math.sqrt((z * 3) ** 2 * Math.cos(2 * u) + Math.sqrt(a ** 4 - (z * 3) ** 4 * Math.sin(2 * u) ** 2));
214  }
215
216  function calculateStartShape(u, v) {
217      let _c = 8;
218      let u_scaled = u * Math.PI * 2;
219      let v_scaled = v * 8/3*2 - 8/3;
220      let scale = 0.125;
221
222      const x = scale * funct(u, v, _c) * Math.cos(u);
223      const y = scale * funct(u, v, _c) * Math.sin(u);
224      const z = scale * 8 * v;
225
226      return { x: x, y: y, z: z }
227  }
```

Рисунок 1. – рівняння для побудови поверхні Virich Cyclic Surface.

В цей код було інтегровано об’єкт аудіоконтексту, що дозволяє отримати доступ до Web Audio API. Далі на html інтерфейс було додано елемент «audio», що дозволяє програвати обрану музику та елемент «checkbox» для керування фільтром (рис. 2).

Рисунок 2 – аудіопрогравач та керування фільтром.

Далі в обєкт «MediaElementSource» було передано елемент «audio» відповідно створений раніше. Наступним кроком було створення «panner» з Web Audio API, він дозволяє проводити просторову роботу зі звуком.

Відповідно до контексту даної роботи він використовується для зміни позиції джерела звуку в результаті маніпулювання з матеріальним інтерфейсом (обертання смартфону). Використовуючи інтерфейс BiquadFilterNode було створено об’єкт фільтру і налаштовано (рис. 3)

```
filter.type = 'peaking';
filter.Q.value = 0.75;
filter.frequency.value = 1000;
filter.gain.value = 16;
ctx.resume();
```

Рисунок 3 – фільтр низьких частот та налаштування.

Відповідно до завдання джерело звуку повинно бути реалізовано в вигляді сфери, для цього було створено рівняння для координат сфери і побудовано відповідну поверхню (рис. 4)

```
586  function createSphereSurface(r) {
587      let vertexList = [];
588      let lon = -Math.PI;
589      let lat = -Math.PI * 0.5;
590      const STEP = 0.1;
591      while (lon < Math.PI) {
592          while (lat < Math.PI * 0.5) {
593              let v1 = sphere(r, lon, lat);
594              let v2 = sphere(r, lon + STEP, lat);
595              let v3 = sphere(r, lon, lat + STEP);
596              let v4 = sphere(r, lon + STEP, lat + STEP);
597              vertexList.push(v1.x, v1.y, v1.z);
598              vertexList.push(v2.x, v2.y, v2.z);
599              vertexList.push(v3.x, v3.y, v3.z);
600              vertexList.push(v3.x, v3.y, v3.z);
601              vertexList.push(v4.x, v4.y, v4.z);
602              vertexList.push(v2.x, v2.y, v2.z);
603              lat += STEP;
604          }
605          lat = -Math.PI * 0.5
606          lon += STEP;
607      }
608      return vertexList;
609  }
```

Рисунок 4 – створення графічного джерела звуку.

Також для того, щоб мати можливість зупиняти програвання музики використано «addEventListener» та прописано увімкнення та вимкнення (рис. 5) треку з використання фільтру чи без.

```
566     audio.addEventListener('pause', () => {
567         console.log('pause');
568         ctx.resume();
569     })
```

Рисунок 5 – реалізація паузи.

ІНСТРУКЦІЯ КОРИСТУВАЧА

Після запуску програми користувач бачить перед собою інтерфейс, що зображено на рисунку 6.

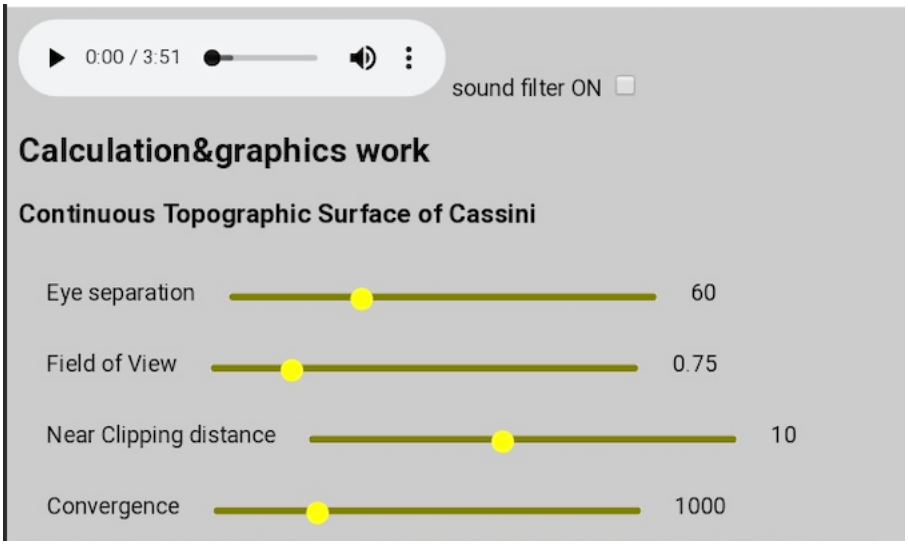

Рисунок 6 – інтерфейс.

Для управління аудіотреком можна використовувати програвач, а також за допомогою прапорця можна увімкнути чи вимкнути звуковий фільтр (рис.7)

Рисунок 7 – частина інтерфейсу для аудіотреку.

За допомогою матеріального інтерфейсу (смартфон) користувач може переміщати графічне відображення джерела звуку (по завданню представлене у вигляді сфери навколо графічного поверхні. Такі маніпуляції дозволяють створювати звуковий ефект переміщення звуку. Наприклад, якщо користувач

слухає музику в навушника то під час запуску сфера знаходиться ліворуч від поверхні, отже музика буде грати лише в лівому навушнику, відповідно обертаючи сферу звучання аудіодоріжки буде переміщатися віртуально переміщатися і в навушниках користувача. Результати маніпуляцій з сферою за допомогою матеріального інтерфейсу продемонстровано на рисунках 8-11.

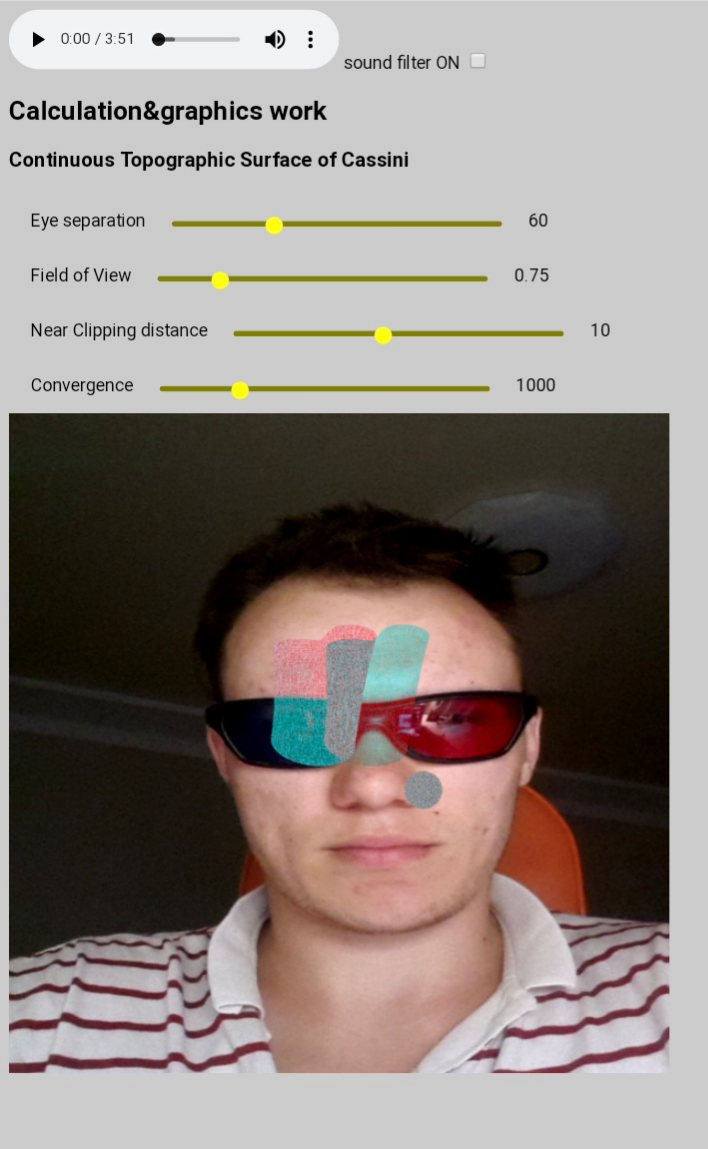

Рисунок 8 – обертання сфери.

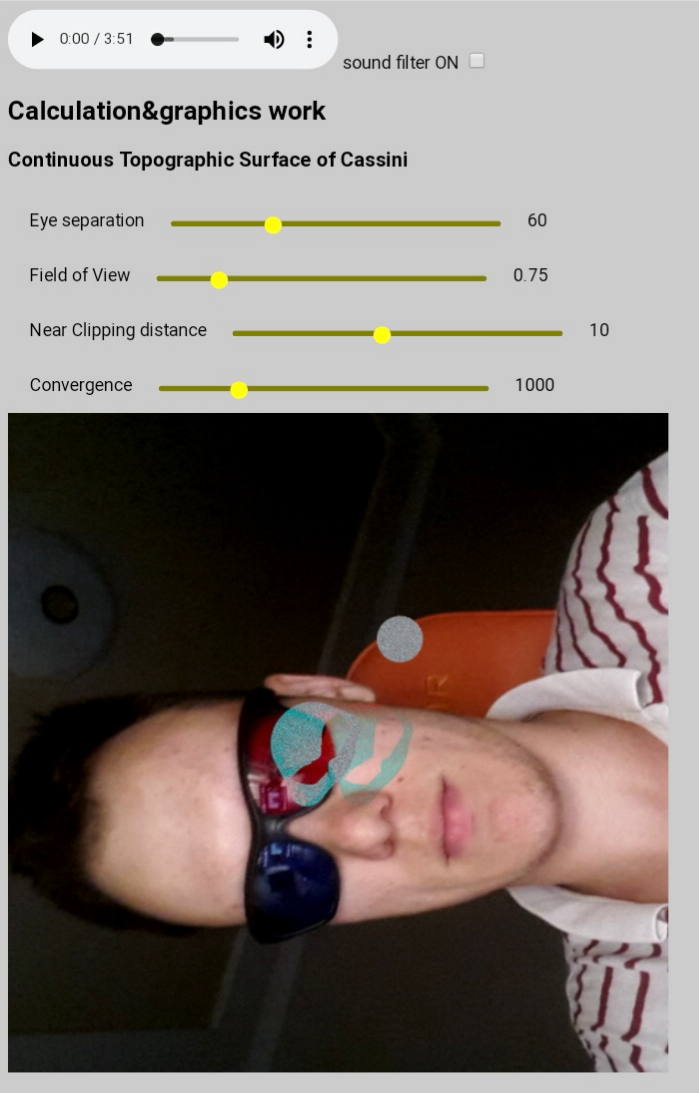

Рисунок 9 – обертання сфери.

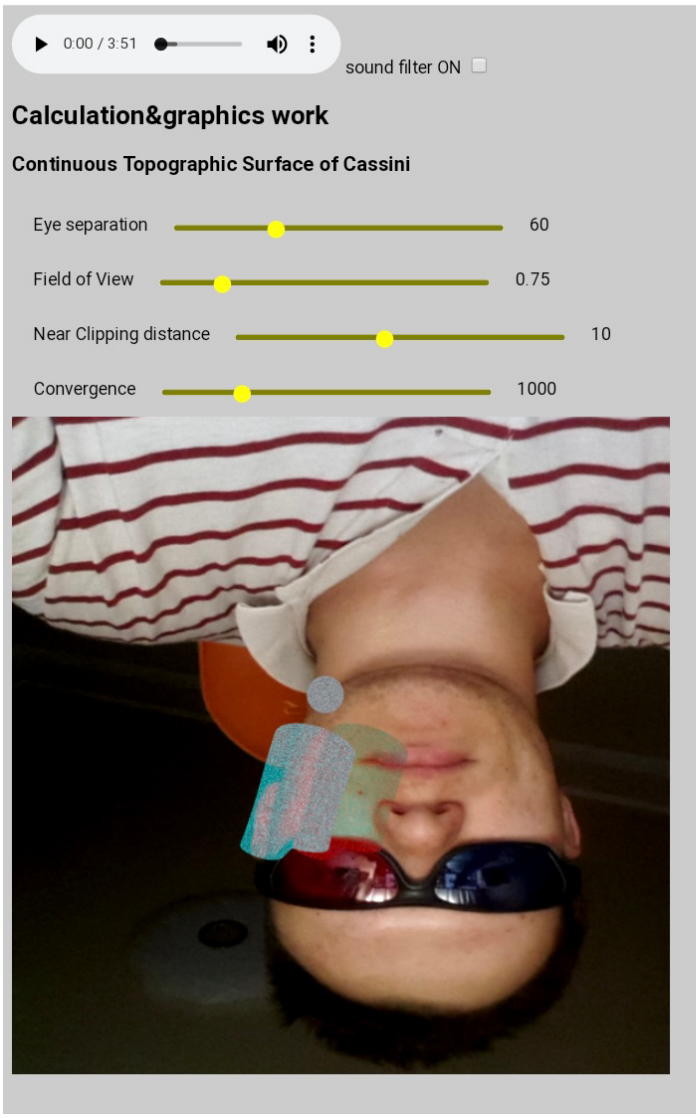

Рисунок 10 – обертання сфери.

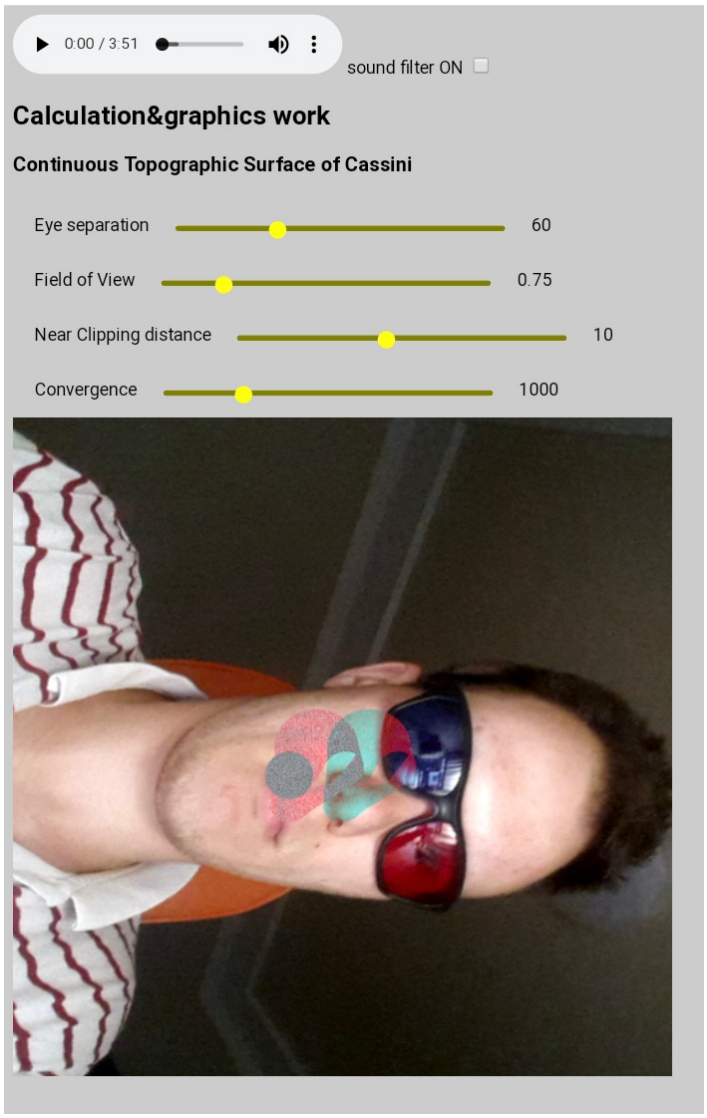

Рисунок 11 – обертання сфери.

Лістинг програми

```
function draw() {


    gl.clearColor(0, 0, 0, 1);

    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);


    calcCamParameters();


    applyLeftFrustrum();


    let projectionLeft = m4.frustum(left, right, bottom, top1, near, far);


    applyRightFrustrum();


    let projectionRight = m4.frustum(left, right, bottom, top1, near, far);


    /* Get the view matrix from the SimpleRotator object.*/

    let modelView = spaceball.getViewMatrix();


    let rotateToPointZero = m4.multiply(m4.axisRotation([0.707, 0.707, 0], 0.0),
getRotationMatrix());

    let translateToPointZero = m4.translation(0, 0, -10);

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

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


    let matAccum0 = m4.multiply(rotateToPointZero, modelView);

    let matAccumLeft = m4.multiply(translateToLeft, matAccum0);

    let matAccumRight = m4.multiply(translateToRight, matAccum0);


    gl.uniform1i(shProgram.iTMU, 0);

    gl.enable(gl.TEXTURE_2D);

    gl.uniform2fv(shProgram.iTexturePoint, [texturePoint.x, texturePoint.y]);
```

```
    gl.uniform1f(shProgram.iscale, -1000.0);

    let projectionNoRotation = m4.perspective(Math.PI / 32, 1, 8, 22);
    let translatetoCenter = m4.translation(-0.5, -0.5, 0);
    let matrixWebCam = m4.multiply(projectionNoRotation, translatetoCenter);
    gl.bindTexture(gl.TEXTURE_2D, webCamTexture);
    gl.texImage2D(
        gl.TEXTURE_2D,
        0,
        gl.RGBA,
        gl.RGBA,
        gl.UNSIGNED_BYTE,
        video
    );
    gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, translateToPointZero);
    gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, matrixWebCam);
    webCamSurface.Draw();
    gl.clear(gl.DEPTH_BUFFER_BIT);

    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.uniform1f(shProgram.iscale, scale);
    gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccumLeft);
    gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projectionLeft);
    gl.colorMask(false, true, true, false);
    surface.Draw();

    gl.clear(gl.DEPTH_BUFFER_BIT);

    gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccumRight);
    gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projectionRight);
```

```
    gl.colorMask(true, false, false, false);
    surface.Draw();

    gl.colorMask(true, true, true, true);
    let t = getPos(beta,gamma,alpha)
    gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false,
m4.multiply(m4.translation(t[0]*1.5,t[1]*1.5,t[2]*1.5),matAccumRight));
    sphereSurface.Draw();
  if(panner){
    panner.setPosition(t[0]*1.5,t[1]*1.5,t[2]*1.5)
  }
}

function setAudioParams() {
    box = document.getElementById('boxFilter');
    audio = document.getElementById('audio');

    audio.addEventListener('play', () => {
        if (!ctx) {
            ctx = new AudioContext();
            source = ctx.createMediaElementSource(audio);
            panner = ctx.createPanner();
            filter = ctx.createBiquadFilter();

            source.connect(panner);
            panner.connect(filter);
            filter.connect(ctx.destination);

            filter.type = 'peaking';
            filter.Q.value = 0.75;
            filter.frequency.value = 1000;
            filter.gain.value = 16;
```

```
            ctx.resume();
        }
    })


    audio.addEventListener('pause', () => {
        console.log('pause');
        ctx.resume();
    })
}

function initAudioContext() {
    box.addEventListener('change', function () {
        if (box.checked) {
            panner.disconnect();
            panner.connect(filter);
            filter.connect(ctx.destination);
        } else {
            panner.disconnect();
            panner.connect(ctx.destination);
        }
    });
    audio.play();
}

function createSphereSurface(r) {
    let vertexList = [];
    let lon = -Math.PI;
    let lat = -Math.PI * 0.5;
    const STEP = 0.1;
    while (lon < Math.PI) {
```

```
        while (lat < Math.PI * 0.5) {
            let v1 = sphere(r, lon, lat);
            let v2 = sphere(r, lon + STEP, lat);
            let v3 = sphere(r, lon, lat + STEP);
            let v4 = sphere(r, lon + STEP, lat + 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(v4.x, v4.y, v4.z);
            vertexList.push(v2.x, v2.y, v2.z);
            lat += STEP;
        }
        lat = -Math.PI * 0.5
        lon += STEP;
    }
    return vertexList;
}

function sphere(r, u, v) {
    let x = r * Math.sin(u) * Math.cos(v);
    let y = r * Math.sin(u) * Math.sin(v);
    let z = r * Math.cos(u);
    return { x: x, y: y, z: z };
}
function getPos(alpha, beta, gamma) {
    const alphaRad = alpha;
    const betaRad = beta;
    const gammaRad = gamma;

    // Define the initial vector along the x-axis
```

```
    let vector = [0, 2, 0];

    // Rotation around the z-axis (gamma)
    const rotZ = [
        [Math.cos(gammaRad), -Math.sin(gammaRad), 0],
        [Math.sin(gammaRad), Math.cos(gammaRad), 0],
        [0, 0, 1]
    ];
    vector = multiplyMatrixVector(rotZ, vector);

    // Rotation around the y-axis (beta)
    const rotY = [
        [Math.cos(betaRad), 0, Math.sin(betaRad)],
        [0, 1, 0],
        [-Math.sin(betaRad), 0, Math.cos(betaRad)]
    ];
    vector = multiplyMatrixVector(rotY, vector);

    // Rotation around the x-axis (alpha)
    const rotX = [
        [1, 0, 0],
        [0, Math.cos(alphaRad), -Math.sin(alphaRad)],
        [0, Math.sin(alphaRad), Math.cos(alphaRad)]
    ];
    vector = multiplyMatrixVector(rotX, vector);

    return vector;
}

function multiplyMatrixVector(matrix, vector) {
    const result = [];
```

```
    for (let i = 0; i < matrix.length; i++) {
        let sum = 0;
        for (let j = 0; j < vector.length; j++) {
            sum += matrix[i][j] * vector[j];
        }
        result.push(sum);
    }
    return result;
}

function init() {
    texturePoint = { x: 0.9, y: 0.5 }
    scale = 0.0;
    let canvas;
    try {
        canvas = document.getElementById("webglcanvas");
        gl = canvas.getContext("webgl");
        video = document.createElement('video');
        video.setAttribute('autoplay', true);
        window.vid = video;
        getWebcam();
        CreateWebCamTexture();
        window.addEventListener('deviceorientation', e => {
            alpha = e.alpha / 180 * Math.PI;
            beta = e.beta / 180 * Math.PI;
            gamma = e.gamma / 180 * Math.PI;
        }, true);
        setAudioParams()
        initAudioContext();
        if (!gl) {
            throw "Browser does not support WebGL";
```

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

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

    playVideoFix()
}
```