НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра цифрових технологій в енергетиці

**Розрахунково графічна робота**

З дисципліни: «Методи синтезу віртуальної реальності»

**Варіант 24**

**Виконав:**

Студент 1-го курсу

ІАТЕ

групи ТР-22мп

Садовничий Анатолій Анатолійович

**Перевірив:**

Демчишин А.А.

Київ – 2023

**Завдання (Варіант 24 - «Смуговий фільтр»)**

1. Повторно використовувати код із практичного завдання №2;
2. Реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні за допомогою матеріального інтерфейсу (цього разу 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. Додайте звуковий фільтр (використовуйте інтерфейс BiquadFilterNode) для кожного варіанту . Додайте елемент прапорця, який вмикає або вимикає фільтр. Встановіть параметри фільтра на свій смак.

**Мета роботи:** засвоїти навички роботи з просторовим аудіо та WebAudio HTML5 API.

**Основне завдання:** відтворити просторове аудіо за допомогою WebAudio HTML5 API.

**Теоретична частина**

WebAudio HTML5 API є потужним інструментом для роботи зі звуком у веб-браузері, який дозволяє вам здійснювати різноманітні маніпуляції з аудіо. Ця API надає широкі можливості для створення, обробки та синтезу звукових ефектів з високою гнучкістю та контролем.

За допомогою WebAudio HTML5 API можна створювати складні аудіо-графи, з'єднувати різні аудіо-вузли та створювати цікаві звукові ефекти. Це дозволяє обробляти звукові дані в реальному часі та створювати різноманітні звукові експерименти. Наприклад, є можливість застосовувати фільтри для зміни спектральних характеристик звуку або створювати ефекти просторової обробки, що додають відчуття тривимірності та глибини.

Крім того, WebAudio HTML5 API дозволяє працювати з різними джерелами звуку. Можна відтворювати звук з завантажених аудіофайлів, записувати аудіо з мікрофону користувача або навіть генерувати звук програмно. Це дозволяє повністю контролювати звуковий матеріал та створювати унікальні звукові елементи у вашому веб-додатку або веб-сайті.

Основні складові структури WebAudio HTML5 APIвключають:

1. Аудіо контекст (AudioContext): Це основний об'єкт, який представляє аудіо контекст, де відбувається весь звуковий процес. Цей об'єкт відповідає за створення, з'єднання та керування аудіо вузлами.
2. Аудіо вузли (Audio Nodes): WebAudio HTML5 API використовує аудіо вузли для створення і обробки звукових сигналів. Деякі з основних типів вузлів включають:

* Генератори (Generators): Наприклад, OscillatorNode для створення осциляційного сигналу.
* Звукові ефекти (Sound Effects): Наприклад, ConvolverNode для використання акустичних ефектів.
* Змішувачі (Mixers): Наприклад, GainNode для керування гучністю звуку.
* Аналізатори (Analyzers): Наприклад, AnalyserNode для аналізу звукових даних.

3. З'єднання аудіо вузлів (Audio Node Connections): Ви можете з'єднувати аудіо вузли, створюючи аудіо граф. Це дозволяє потоково обробляти звукові дані і створювати складні звукові ефекти.

4. Звукові джерела (Audio Sources): WebAudio HTML5 API підтримує різні типи звукових джерел, таких як аудіо файл, мікрофон або генерований звук. Є можливість використовувати AudioBufferSourceNode для відтворення аудіо з попередньо завантаженого буфера.

Ефекти звуку (Sound Filters): WebAudio HTML5 API надає можливість додавати звукові фільтри до аудіо вузлів. BiquadFilterNode є одним з типів фільтрів, які можна використовувати. Можна налаштовувати параметри, такі як тип фільтра (наприклад, нижнього або верхнього), частота, добуток та довільні параметри коефіцієнтів фільтра.

WebAudio HTML5 API дозволяє візуалізувати звукові ефекти та контролувати їх параметри через інтерактивний інтерфейс. За допомогою нього можна відтворювати звукові сигнали у просторовій області та відслідковувати їх рух з використанням різних візуальних елементів.

WebAudio HTML5 API є потужним інструментом для реалізації звукових можливостей у веб-додатках. Вона відкриває безліч можливостей для творчості та взаємодії з користувачем через звукові ефекти.

**Реалізація**

Для створення ефекту просторового звуку у веб-додатку було використано JavaScript як мову програмування, а також бібліотеку WebGL для візуалізації джерела звуку. Однак, головною складовою для реалізації звукової частини було використано WebAudio HTML5 API. У наступних розділах будуть описані ключові деталі їх інтеграції:

1. По-перше, була створена нова гілка з назвою CGW у репозиторії GitHub. Після виконання завдання, програмний код, відео-демонстрацію та звіт були завантажені до цієї гілки..
2. Було створене візуальне представлення джерела звуку шляхом відображення сфери. Для цього ми використовуємо функцію, яка надає текстурні координати та вершини для подальшого відображення цієї сфери

const CreateSphereData = (radius) => {

    const vertexList = [];

    const textureList = [];

    const splines = 20;

    const maxU = Math.PI;

    const maxV = 2 \* Math.PI;

    const stepU = maxU / splines;

    const stepV = maxV / splines;

    const getU = (u) => u / maxU;

    const getV = (v) => v / maxV;

    for (let u = 0; u <= maxU; u += stepU) {

        for (let v = 0; v <= maxV; v += stepV) {

            const x = radius \* Math.sin(u) \* Math.cos(v);

            const y = radius \* Math.sin(u) \* Math.sin(v);

            const z = radius \* Math.cos(u);

            vertexList.push(x, y, z);

            textureList.push(getU(u), getV(v));

            const xNext = radius \* Math.sin(u + stepU) \* Math.cos(v + stepV);

            const yNext = radius \* Math.sin(u + stepU) \* Math.sin(v + stepV);

            const zNext = radius \* Math.cos(u + stepU);

            vertexList.push(xNext, yNext, zNext);

            textureList.push(getU(u + stepU), getV(v + stepV));

        }

    }

    return {

        verticesSphere: vertexList,

        texturesSphere: textureList,

    };

};

1. Перед рендерінгом сфери, до неї застосовується текстура яка завантажується за допомогою функції loadSphereTexture..

const loadSphereTexture = () => {

    const image = new Image();

    image.crossOrigin = "anonymous";

    image.src = "https://www.manytextures.com/download/18/texture/jpg/256/stone-wall-256x256.jpg";

    image.addEventListener("load", () => {

        sphereTex = gl.createTexture();

        gl.bindTexture(gl.TEXTURE\_2D, sphereTex);

        gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, image);

        gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.LINEAR);

        gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MAG\_FILTER, gl.LINEAR);

        console.log("Sphere texture is loaded!")

    });

};

1. Після завершення процесу рендерінгу сфери в функції draw, наступним кроком є додавання звуку та прив'язка його до координат фігури. Перед налаштуванням та роботою зі звуком, спочатку потрібно додати його до HTML-файлу (Рисунок 4) за допомогою тегу audio. В цьому тегу вказується посилання на аудіофайл або його розташування за допомогою елементу source. Також були вказані параметри для відображення кнопок управління звуком та для зациклення відтворення. У цій роботі у якості аудіофайлу використовується композиція Бориса Сичевського - Як би мені сивий кінь.

<audio id="audio" style="margin-top: 15px; margin-bottom: 15px" controls loop>

      <source src="./audio.mp3" type="audio/mpeg" />

      Audio is not supported by browser.

</audio>

1. Наступним кроком є прив’язка показників отриманих з датчика, до фактичних координат джерела звуку, також підв'язка зміни координат до перемикачів в html файлі.

 xPos = parseFloat(document.getElementById("xPos").value);

    yPos = parseFloat(document.getElementById("yPos").value);

    zPos = parseFloat(document.getElementById("zPos").value);

    if (

        orientationEvent.alpha &&

        orientationEvent.beta &&

        orientationEvent.gamma

    ) {

        let rotationMatrix = getRotationMatrix(

            orientationEvent.alpha,

            orientationEvent.beta,

            orientationEvent.gamma

        );

        let translationMatrix = m4.translation(0, 0, -1);

        xPos = orientationEvent.gamma;

        yPos = orientationEvent.beta;

        modelView = m4.multiply(rotationMatrix, translationMatrix);

    }

    if (audioPanner) {

        audioPanner.setPosition(

            xPos,

            yPos,

            zPos

        );

    }

    const translationMatrix = m4.translation(xPos, yPos, zPos);

const xPosInput = document.getElementById("xPos");

    const yPosInput = document.getElementById("yPos");

    const zPosInput = document.getElementById("zPos");

    xPosInput.addEventListener("input", draw);

    yPosInput.addEventListener("input", draw);

    zPosInput.addEventListener("input", draw);

1. Останнім кроком є підключення створення event для відтворення аудіо, та задання параметрів для звукового фільру. Початково необхідно створити аудіо-контекст за допомогою **window.AudioContext** або **window.webkitAudioContext** для старіших версій браузерів. Також потрібно створити початковий елемент **AudioSource** та вузли для обробки звуку, такі як **PannerNode** та **BiquadFilterNode**. Після створення об'єктів потрібно налаштувати їх. Для об'єкту audioPanner встановлюється модель панування (**panningModel)** зі значенням **"HRTF" (Head-Related Transfer Function)**, а також модель відстані (**distanceModel**) зі значенням "**linear**" для лінійного зменшення гучності пропорційно до відстані. Для об'єкту audioFilter вказується тип фільтру відповідно до варіанту ("**bandpass**" - смуговий фільтр), частота зрізу (визначена за формулою *geometricMean = sqrt(from \* to))*, а також фактор якості (Q/Quality factor), який обчислюється за формулою *geometricMean / (to - from****).***

audio = document.getElementById("audio");

    audio.addEventListener("pause", () => {

        audioContext.resume();

    });

    audio.addEventListener("play", () => {

        if (!audioContext) {

        audioContext = new (window.AudioContext || window.webkitAudioContext)();

        audioSource = audioContext.createMediaElementSource(audio);

        audioPanner = audioContext.createPanner();

        audioFilter = audioContext.createBiquadFilter();

        var from = 300;

        var to = 30000;

        var geometricMean = Math.sqrt(from \* to);

        audioPanner.panningModel = "HRTF";

        audioPanner.distanceModel = "linear";

        audioFilter.type = "bandpass";

        audioFilter.frequency.value = geometricMean;

        audioFilter.Q.value = geometricMean / (to - from);

        audioSource.connect(audioPanner);

        audioPanner.connect(audioFilter);

        audioFilter.connect(audioContext.destination);

        audioContext.resume();

        }

    });

    let filterOn = document.getElementById("filterCheckbox");

    filterOn.addEventListener("change", function () {

        if (filterOn.checked) {

        audioPanner.disconnect();

        audioPanner.connect(audioFilter);

        audioFilter.connect(audioContext.destination);

        } else {

        audioPanner.disconnect();

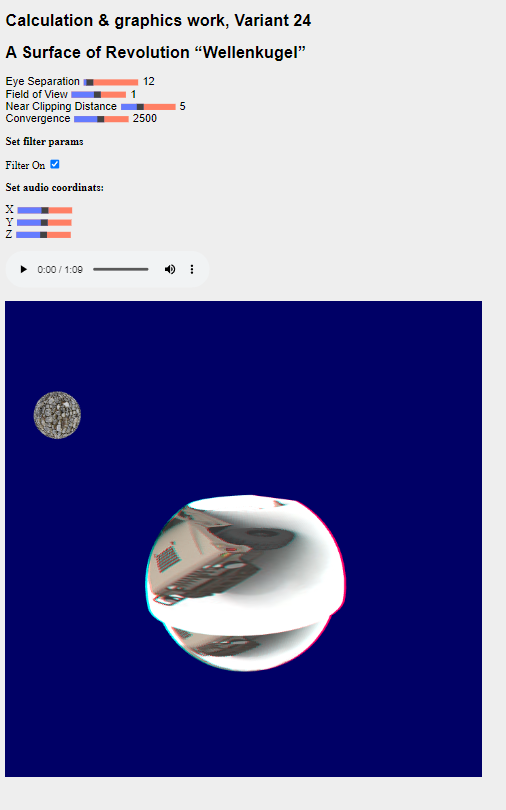
        audioPanner.connect(audioContext.destination);

        }

    });

**Інструкція користувача**

Візуальна частина озробленого веб додатка виглядає наступним чином.



Користувач має змогу налаштовувати параметри анагліфного ефекту, змінюючи значення eye separation, field of view, clipping distance, convergence. Також присутній чекбокс для вмикання/вимикання звукового фільтру та регулятори координат центру сфери (що представляє джерело звуку. Нижче присутній плеєр для вмикання, вимикання, перемотки та регулювання звуку музики.

**Вихідний код**

**index.html**

<!DOCTYPE html>

<html>

<head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

    <title>CGW</title>

    <style>

        p, h2, span, input {

            font-family: Arial, Helvetica, sans-serif;

        }

        body {

            background-color: #EEEEEE;

        }

        @media screen and (-webkit-min-device-pixel-ratio:0) {

            input[type='range'] {

              overflow: hidden;

              width: 80px;

              -webkit-appearance: none;

              background-color: #FF7F64;

            }

            input[type='range']::-webkit-slider-runnable-track {

              height: 10px;

              -webkit-appearance: none;

              color: #647BFF;

              margin-top: -1px;

            }

            input[type='range']::-webkit-slider-thumb {

              width: 10px;

              -webkit-appearance: none;

              height: 10px;

              cursor: ew-resize;

              background: #434343;

              box-shadow: -80px 0 0 80px #647BFF;

            }

        }

    </style>

    <script src="./Utils/trackball-rotator.js"></script>

    <script src="./Utils/m4.js"></script>

    <script src="./shader.gpu"></script>

    <script src="./main.js"></script>

</head>

<body onload="init()">

    <h2>Calculation & graphics work, Variant 24</h2>

    <h2>A Surface of Revolution “Wellenkugel”</h2>

    <span>Eye Separation</span>

    <input type="range" value="70" min="2" max="200" step="2" id="id1" />

    <span class="s">0</span><br>

    <span>Field of View</span>

    <input type="range" value="0.5" min="0.1" max="2.0" step="0.1" id="id2" />

    <span class="s">0</span><br>

    <span>Near Clipping Distance</span>

    <input type="range" value="5" min="0" max="15" step="0.5" id="id3" />

    <span class="s">0</span><br>

    <span>Convergence</span>

    <input type="range" value="2500" min="100" max="5000" step="100" id="id4" />

    <span class="s">0</span><br>

    <div style="font-weight: bold; margin-top: 15px; margin-bottom: 15px">

      Set filter params

    </div>

    <div>

      <label for="filterCheckbox">Filter On</label>

      <input type="checkbox" id="filterCheckbox" name="filterCheckbox" checked/>

    </div>

    <div style="font-weight: bold; margin-top: 15px; margin-bottom: 15px">

      Set audio coordinats:

    </div>

    <div>

      <label for="xPos">X</label>

      <input type="range" id="xPos" name="xPosition" min="-35" max="35" step="0.01" value="0"/>

    </div>

    <div>

      <label for="yPos">Y</label>

      <input type="range" id="yPos" name="yPosition" min="-35" max="35" step="0.01" value="0"/>

    </div>

    <div>

      <label for="zPos">Z</label>

      <input type="range" id="zPos" name="zPosition" min="-1" max="1" step="0.01" value="0"/>

    </div>

    <audio id="audio" style="margin-top: 15px; margin-bottom: 15px"  controls loop>

      <source src="./audio.mp3" type="audio/mpeg" />

      Audio is not supported by browser.

    </audio>

    <div id="canvas-holder">

        <canvas width="700" height="700" id="webglcanvas" style="background-color:black"></canvas>

    </div>

</body>

</html>

**main.js**

"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 inputValue = 0.0;

let scaleU = 0.0;

let scaleV = 0.0;

let scaleValue = 1;

const r = parseFloat(1.0);

const a = parseFloat(0.5);

const n = parseInt(300);

const uDel = 0.001;

const vDel = 0.001;

let orientationEvent = { alpha: 0, beta: 0, gamma: 0 };

let camera,

    texture,

    webCamText,

    video,

    stream,

    track,

    surface2;

let sphere, sphereTex;

let audio = null;

let audioContext;

let audioSource;

let audioPanner;

let audioFilter;

let xPos = 0;

let yPos = 0;

let zPos = 0;

function deg2rad(angle) {

    return (angle \* Math.PI) / 180;

}

function getWebcam() {

    navigator.getUserMedia({ video: true, audio: false }, function (stream) {

        video.srcObject = stream;

        track = stream.getTracks()[0];

    }, function (e) {

        console.error('Rejected!', e);

    });

}

function CreateWebCamTexture() {

    let textureID = gl.createTexture();

    gl.bindTexture(gl.TEXTURE\_2D, textureID);

    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);

    return textureID;

}

//Constructor

function StereoCamera(

    Convergence,

    EyeSeparation,

    AspectRatio,

    FOV,

    NearClippingDistance,

    FarClippingDistance

) {

    this.mConvergence = Convergence;

    this.mEyeSeparation = EyeSeparation;

    this.mAspectRatio = AspectRatio;

    this.mFOV = FOV;

    this.mNearClippingDistance = NearClippingDistance;

    this.mFarClippingDistance = FarClippingDistance;

    this.mProjectionMatrix = null;

    this.mModelViewMatrix = null;

    this.ApplyLeftFrustum = function () {

        let top, bottom, left, right;

        top = this.mNearClippingDistance \* Math.tan(this.mFOV / 2);

        bottom = -top;

        let a = this.mAspectRatio \* Math.tan(this.mFOV / 2) \* this.mConvergence;

        let b = a - this.mEyeSeparation / 2;

        let c = a + this.mEyeSeparation / 2;

        left = (-b \* this.mNearClippingDistance) / this.mConvergence;

        right = (c \* this.mNearClippingDistance) / this.mConvergence;

        // Set the Projection Matrix

        this.mProjectionMatrix = m4.frustum(

            left,

            right,

            bottom,

            top,

            this.mNearClippingDistance,

            this.mFarClippingDistance

        );

        // Displace the world to right

        this.mModelViewMatrix = m4.translation(

            this.mEyeSeparation / 2,

            0.0,

            0.0

        );

    };

    this.ApplyRightFrustum = function () {

        let top, bottom, left, right;

        top = this.mNearClippingDistance \* Math.tan(this.mFOV / 2);

        bottom = -top;

        let a = this.mAspectRatio \* Math.tan(this.mFOV / 2) \* this.mConvergence;

        let b = a - this.mEyeSeparation / 2;

        let c = a + this.mEyeSeparation / 2;

        left = (-c \* this.mNearClippingDistance) / this.mConvergence;

        right = (b \* this.mNearClippingDistance) / this.mConvergence;

        // Set the Projection Matrix

        this.mProjectionMatrix = m4.frustum(

            left,

            right,

            bottom,

            top,

            this.mNearClippingDistance,

            this.mFarClippingDistance

        );

        // Displace the world to left

        this.mModelViewMatrix = m4.translation(

            -this.mEyeSeparation / 2,

            0.0,

            0.0

        );

    };

    this.updateValues = function () {

        let values = document.getElementsByClassName("s");

        let eyeSepar = 70.0;

        eyeSepar = document.getElementById("id1").value;

        values[0].innerHTML = eyeSepar;

        this.mEyeSeparation = eyeSepar;

        let fov = 0.5;

        fov = document.getElementById("id2").value;

        values[1].innerHTML = fov;

        this.mFOV = fov;

        let nearClipDist = 5.0;

        nearClipDist = document.getElementById("id3").value - 0.0;

        values[2].innerHTML = nearClipDist;

        this.mNearClippingDistance = nearClipDist

        let convergence = 2500.0;

        convergence = document.getElementById("id4").value;

        values[3].innerHTML = convergence;

        this.mConvergence = convergence

    }

}

// Constructor

function Model(name) {

    this.name = name;

    this.iVertexBuffer = gl.createBuffer();

    this.iTextureBuffer = gl.createBuffer();

    this.count = 0;

    this.BufferData = function (vertices, texCoords) {

        gl.bindBuffer(gl.ARRAY\_BUFFER, this.iVertexBuffer);

        gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(vertices), gl.STREAM\_DRAW);

        gl.bindBuffer(gl.ARRAY\_BUFFER, this.iTextureBuffer)

        gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(texCoords), 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.iTextureBuffer);

        gl.vertexAttribPointer(shProgram.iTextureCoords, 2, gl.FLOAT, false, 0, 0);

        gl.enableVertexAttribArray(shProgram.iTextureCoords);

        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.iTextureCoords = -1;

    // Location of the uniform matrix representing the combined transformation.

    this.iModelViewProjectionMatrix = -1;

    this.iTexture = -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() {

    camera.updateValues();

    gl.clearColor(0, 0, 0.4, 0.4);

    gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

    gl.enable(gl.CULL\_FACE);

    // Enable the depth buffer

    gl.enable(gl.DEPTH\_TEST);

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

    /\* Set the values of the projection transformation \*/

    let projection = m4.perspective(Math.PI / 4, 1, 4, 12);

    let projectionS = m4.orthographic(0, 1, 0, 1, -1, 1);

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

    let modelView = spaceball.getViewMatrix();

    let modelView2 = m4.identity();

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

    let translateToPointZero2 = m4.translation(-5, -5, -10);

    let translateToPointZeroS = m4.translation(0.0, 0, 0);

    let modelView1 = null;

    gl.bindTexture(gl.TEXTURE\_2D, sphereTex);

    xPos = parseFloat(document.getElementById("xPos").value);

    yPos = parseFloat(document.getElementById("yPos").value);

    zPos = parseFloat(document.getElementById("zPos").value);

    if (

        orientationEvent.alpha &&

        orientationEvent.beta &&

        orientationEvent.gamma

    ) {

        let rotationMatrix = getRotationMatrix(

            orientationEvent.alpha,

            orientationEvent.beta,

            orientationEvent.gamma

        );

        let translationMatrix = m4.translation(0, 0, -1);

        xPos = orientationEvent.gamma;

        yPos = orientationEvent.beta;

        modelView = m4.multiply(rotationMatrix, translationMatrix);

    }

    if (audioPanner) {

        audioPanner.setPosition(

            xPos,

            yPos,

            zPos

        );

    }

    const translationMatrix = m4.translation(xPos, yPos, zPos);

    const scaleMatrix = m4.scaling(0.01, 0.01, 0.01);

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

    let matAccumTranslationS = m4.multiply(translationMatrix, matAccumS);

    let matAccumZeroS = m4.multiply(translateToPointZeroS, matAccumTranslationS);

    matAccumZeroS = m4.multiply(scaleMatrix, matAccumZeroS);

    gl.clear(gl.DEPTH\_BUFFER\_BIT);

    gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projectionS);

    gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccumZeroS);

    sphere.Draw();

    gl.clear(gl.DEPTH\_BUFFER\_BIT);

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

    let matAccum1 = m4.multiply(translateToPointZero, matAccum);

    let matAccum12 = m4.multiply(translateToPointZero2, matAccum);

    gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projection);

    gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, m4.multiply(matAccum12, m4.scaling(10, 10, 1)));

    gl.uniform1i(shProgram.iTexture, 0);

    gl.bindTexture(gl.TEXTURE\_2D, webCamText);

    gl.texImage2D(

        gl.TEXTURE\_2D,

        0,

        gl.RGBA,

        gl.RGBA,

        gl.UNSIGNED\_BYTE,

        video

    );

    surface2.Draw();

    gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccum1);

    camera.ApplyLeftFrustum()

    gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, camera.mProjectionMatrix);

    gl.bindTexture(gl.TEXTURE\_2D, texture);

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

    surface.Draw();

    gl.clear(gl.DEPTH\_BUFFER\_BIT);

    camera.ApplyRightFrustum()

    gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, camera.mProjectionMatrix);

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

    surface.Draw();

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

}

function draw2() {

    draw();

    window.requestAnimationFrame(draw2);

}

function getRotationMatrix(alpha, beta, gamma) {

    var \_x = beta ? deg2rad(beta) : 0;

    var \_y = gamma ? deg2rad(gamma) : 0;

    var \_z = alpha ? deg2rad(alpha) : 0;

    var cX = Math.cos(\_x);

    var cY = Math.cos(\_y);

    var cZ = Math.cos(\_z);

    var sX = Math.sin(\_x);

    var sY = Math.sin(\_y);

    var sZ = Math.sin(\_z);

    var m11 = cZ \* cY - sZ \* sX \* sY;

    var m12 = -cX \* sZ;

    var m13 = cY \* sZ \* sX + cZ \* sY;

    var m21 = cY \* sZ + cZ \* sX \* sY;

    var m22 = cZ \* cX;

    var m23 = sZ \* sY - cZ \* cY \* sX;

    var m31 = -cX \* sY;

    var m32 = sX;

    var m33 = cX \* cY;

    return [m11, m12, m13, 0, m21, m22, m23, 0, m31, m32, m33, 0, 0, 0, 0, 1];

}

//Creating data as vertices for surface

function derUFunc(u, v, uDelta) {

    let x = calculateXCoordinate(u, v);

    let y = calculateYCoordinate(u, v);

    let z = calculateZCoordinate(u, v);

    let Dx = calculateXCoordinate(u + uDelta, v);

    let Dy = calculateYCoordinate(u + uDelta, v);

    let Dz = calculateZCoordinate(u + uDelta, v);

    let Dxdu = (Dx - x) / deg2rad(uDelta);

    let Dydu = (Dy - y) / deg2rad(uDelta);

    let Dzdu = (Dz - z) / deg2rad(uDelta);

    return [Dxdu, Dydu, Dzdu];

}

function derVFunc(u, v, vDelta) {

    let x = calculateXCoordinate(u, v);

    let y = calculateYCoordinate(u, v);

    let z = calculateZCoordinate(u, v);

    let Dx = calculateXCoordinate(u, v + vDelta);

    let Dy = calculateYCoordinate(u, v + vDelta);

    let Dz = calculateZCoordinate(u, v + vDelta);

    let Dxdv = (Dx - x) / deg2rad(vDelta);

    let Dydv = (Dy - y) / deg2rad(vDelta);

    let Dzdv = (Dz - z) / deg2rad(vDelta);

    return [Dxdv, Dydv, Dzdv];

}

function calculateULimit(u) {

    return u \* Math.PI \* 14.5;

}

function calculateVLimit(v) {

    return v \* Math.PI \* 1.5;

}

function calculateXCoordinate(u, v) {

    return u \* Math.cos(Math.cos(u)) \* Math.cos(v);

}

function calculateYCoordinate(u, v) {

    return u \* Math.cos(Math.cos(u)) \* Math.sin(v);

}

function calculateZCoordinate(u, v) {

    return u \* Math.sin(Math.cos(u))

}

function CreateSurfaceData() {

    let vertices = [];

    let normals = [];

    let textCoords = [];

    const n = 25;

    let step = 0.1;

    let uend = Math.PI \* 14.5 + step;

    let vend = Math.PI \* 1.5 + step;

    let DeltaU = 0.0001;

    let DeltaV = 0.0001;

    //Proportionally changes the size of the figure along three axes

    const sizeIndex = 0.05;

    for (let u = 0; u < uend; u += step) {

        //let u1 = i / n;

        let unext = u + step;

        for (let v = 0; v < vend; v += step) {

            let x = calculateXCoordinate(u, v) \* sizeIndex;

            let y = calculateYCoordinate(u, v) \* sizeIndex;

            let z = calculateZCoordinate(u, v) \* sizeIndex;

            vertices.push(x, y, z);

            x = calculateXCoordinate(unext, v) \* sizeIndex;

            y = calculateYCoordinate(unext, v) \* sizeIndex;

            z = calculateZCoordinate(unext, v) \* sizeIndex;

            vertices.push(x, y, z);

            let derU = derUFunc(u, v, DeltaU);

            let derV = derVFunc(u, v, DeltaV);

            let result = m4.cross(derV, derU);

            normals.push(result[0]);

            normals.push(result[1]);

            normals.push(result[2]);

            derU = derUFunc(unext, v, uDel);

            derV = derVFunc(unext, v, vDel);

            result = m4.cross(derV, derU);

            normals.push(result[0]);

            normals.push(result[1]);

            normals.push(result[2]);

            textCoords.push(u / uend \* n, v / vend);

            textCoords.push(unext / uend \* n, v / vend);

        }

    }

    return [vertices, normals, textCoords];

}

const CreateSphereData = (radius) => {

    const vertexList = [];

    const textureList = [];

    const splines = 20;

    const maxU = Math.PI;

    const maxV = 2 \* Math.PI;

    const stepU = maxU / splines;

    const stepV = maxV / splines;

    const getU = (u) => u / maxU;

    const getV = (v) => v / maxV;

    for (let u = 0; u <= maxU; u += stepU) {

        for (let v = 0; v <= maxV; v += stepV) {

            const x = radius \* Math.sin(u) \* Math.cos(v);

            const y = radius \* Math.sin(u) \* Math.sin(v);

            const z = radius \* Math.cos(u);

            vertexList.push(x, y, z);

            textureList.push(getU(u), getV(v));

            const xNext = radius \* Math.sin(u + stepU) \* Math.cos(v + stepV);

            const yNext = radius \* Math.sin(u + stepU) \* Math.sin(v + stepV);

            const zNext = radius \* Math.cos(u + stepU);

            vertexList.push(xNext, yNext, zNext);

            textureList.push(getU(u + stepU), getV(v + stepV));

        }

    }

    return {

        verticesSphere: vertexList,

        texturesSphere: textureList,

    };

};

/\* 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.iNormalVertex = gl.getAttribLocation(prog, "normal");

    shProgram.iTextureCoords = gl.getAttribLocation(prog, "texcoord");

    shProgram.iProjectionMatrix = gl.getUniformLocation(prog, "ProjectionMatrix");

    shProgram.iModelViewMatrix = gl.getUniformLocation(prog, "ModelViewMatrix");

    shProgram.iTexture = gl.getUniformLocation(prog, "u\_texture");

    sphere = new Model("Sphere");

    const { verticesSphere, texturesSphere } = CreateSphereData(5);

    sphere.BufferData(verticesSphere, texturesSphere);

    surface = new Model('Surface');

    surface.BufferData(CreateSurfaceData()[0], CreateSurfaceData()[2]);

    surface2 = new Model('Surface2');

    surface2.BufferData(

        [0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0],

        [1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0])

    loadSphereTexture();

    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;

    camera = new StereoCamera(

        2500,

        70.0,

        1,

        0.5,

        5,

        100

    );

    try {

        canvas = document.getElementById("webglcanvas");

        gl = canvas.getContext("webgl");

        video = document.createElement('video');

        video.setAttribute('autoplay', true);

        window.vid = video;

        getWebcam();

        webCamText = CreateWebCamTexture();

        if (!gl) {

            throw "Browser does not support WebGL";

        }

    }

    catch (e) {

        console.log(e);

        document.getElementById("canvas-holder").innerHTML =

            "<p>Sorry, could not get a WebGL graphics context.</p>";

        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;

    }

    const xPosInput = document.getElementById("xPos");

    const yPosInput = document.getElementById("yPos");

    const zPosInput = document.getElementById("zPos");

    xPosInput.addEventListener("input", draw);

    yPosInput.addEventListener("input", draw);

    zPosInput.addEventListener("input", draw);

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

    if ("DeviceOrientationEvent" in window) {

        window.addEventListener("deviceorientation", handleOrientation);

    } else {

        console.log("Device orientation not supported");

    }

    audio = document.getElementById("audio");

    audio.addEventListener("pause", () => {

        audioContext.resume();

    });

    audio.addEventListener("play", () => {

        if (!audioContext) {

        audioContext = new (window.AudioContext || window.webkitAudioContext)();

        audioSource = audioContext.createMediaElementSource(audio);

        audioPanner = audioContext.createPanner();

        audioFilter = audioContext.createBiquadFilter();

        var from = 300;

        var to = 30000;

        var geometricMean = Math.sqrt(from \* to);

        audioPanner.panningModel = "HRTF";

        audioPanner.distanceModel = "linear";

        audioFilter.type = "bandpass";

        audioFilter.frequency.value = geometricMean;

        audioFilter.Q.value = geometricMean / (to - from);

        audioSource.connect(audioPanner);

        audioPanner.connect(audioFilter);

        audioFilter.connect(audioContext.destination);

        audioContext.resume();

        }

    });

    let filterOn = document.getElementById("filterCheckbox");

    filterOn.addEventListener("change", function () {

        if (filterOn.checked) {

        audioPanner.disconnect();

        audioPanner.connect(audioFilter);

        audioFilter.connect(audioContext.destination);

        } else {

        audioPanner.disconnect();

        audioPanner.connect(audioContext.destination);

        }

    });

    draw2();

    LoadTexture();

}

function LoadTexture() {

    texture = gl.createTexture();

    gl.bindTexture(gl.TEXTURE\_2D, texture);

    gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED\_BYTE,

        new Uint8Array([0, 0, 255, 255]));

    var image = new Image();

    image.crossOrigin = "anonymous"

    image.src = "https://files.cults3d.com/uploaders/13286518/illustration-file/78ac4637-9420-4c13-90ca-1f73381973ac/hummer.jpg";

    image.addEventListener('load', function () {

        gl.bindTexture(gl.TEXTURE\_2D, texture);

        gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, image);

        gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_S, gl.REPEAT);

        gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_T, gl.REPEAT);

        gl.generateMipmap(gl.TEXTURE\_2D);

        console.log("Texture is loaded!");

        draw();

    });

}

const loadSphereTexture = () => {

    const image = new Image();

    image.crossOrigin = "anonymous";

    image.src = "https://www.manytextures.com/download/18/texture/jpg/256/stone-wall-256x256.jpg";

    image.addEventListener("load", () => {

        sphereTex = gl.createTexture();

        gl.bindTexture(gl.TEXTURE\_2D, sphereTex);

        gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, image);

        gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.LINEAR);

        gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MAG\_FILTER, gl.LINEAR);

        console.log("Sphere texture is loaded!")

    });

};

const handleOrientation = (event) => {

    orientationEvent.alpha = event.alpha;

    orientationEvent.beta = event.beta;

    orientationEvent.gamma = event.gamma;

};