Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інституту атомної та теплової енергетики

Кафедра ЦТЕ

**Методи синтезу віртуальної реальності**

Розрахунково-графічна робота

Варіант 19

Виконав: студент гр. ТР-31мп

Павський Д. О.

*(П.І.Б.)*

Перевірив: Демчишин А.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(П.І.Б., підпис)*

Київ – 2024

**Завдання**

* Потрібно взяти за основу практичне завдання 2;
* Реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні круговим способом протягом певного часу (цього разу поверхня залишається нерухомою, а джерело звуку рухається). Відтворити свою улюблену пісню у форматі mp3/ogg, маючи просторове положення джерела звуку, кероване користувачем;
* Візуалізувати положення джерела звуку за допомогою сфери;
* Додати звуковий фільтр (використовуючи інтерфейс BiquadFilterNode) за варіантом. Також необхідно додати елемент прапорця, який вмикає або вимикає фільтр. Параметри фільтра встановлюються на свій смак

**Теоритичні відомості**

**Web Audio API**

Web Audio API - це потужний інтерфейс для обробки та синтезу аудіо в веб-браузерах. Вона надає розробникам засоби для створення, обробки та керування аудіосигналами в реальному часі, що дозволяє створювати складні аудіо ефекти та інтерактивні аудіо додатки. Основні функціональні можливості Web Audio API включають створення аудіоконтекстів, додавання аудіоефектів, фільтрів та просторового аудіо. Аудіоконтекст (AudioContext) є основним об'єктом, через який здійснюється вся обробка аудіо. Він дозволяє створювати аудіо джерела (наприклад, з медіа елементів або генераторів сигналів), а також додавати різні аудіо ефекти і фільтри, об'єднуючи їх у складні графи обробки.

Важливим аспектом Web Audio API є підтримка просторового аудіо, що дозволяє створювати ефекти тривимірного звуку. Це досягається завдяки використанню об'єктів типу PannerNode, які можуть розміщувати звукові джерела у тривимірному просторі і модулювати їх інтенсивність залежно від позиції слухача.

Однією з основних переваг Web Audio API є її здатність працювати з аудіосигналами в реальному часі. Це дозволяє створювати інтерактивні аудіододатки, де користувач може впливати на звук через різні дії, такі як рух миші, натискання клавіш або взаємодію з іншими елементами веб-сторінки. В цілому, Web Audio API є потужним інструментом для веб-розробників, які хочуть додати багатий і інтерактивний аудіодосвід у свої веб-додатки. Значною перевагою використання Web Audio API є його здатність інтегрувати аудіо з іншими веб-технологіями, такими як WebGL для графіки та відео. Це дозволяє створювати поєднання, де звук, відео та графіка синхронізуються і взаємодіють. Наприклад, у веб-додатку можна створити тривимірну сцену з анімованими об'єктами, які видають звук відповідно до свого руху, використовуючи PannerNode для просторового розташування звукових джерел.

**BiquadFilterNode**

BiquadFilterNode є одним із основних компонентів Web Audio API, що дозволяє реалізовувати різноманітні фільтри для обробки аудіосигналів. Це базовий елемент для створення аудіофільтрів з двома полюсами та нулями, що забезпечує велику гнучкість у створенні аудіоефектів. BiquadFilterNode може представляти різні типи фільтрів, еквалайзерів та регуляторів тембру. Він завжди має один вхід і один вихід. BiquadFilterNode можна використовувати в різних сценаріях, наприклад, для створення ефекту еквалайзера, ревербу або для обробки аудіосигналу в реальному часі.

Основними типами фільтрів, які можна створювати за допомогою BiquadFilterNode, є lowpass, highpass, bandpass, lowshelf, highshelf, peaking, notch та allpass фільтри. Кожен з цих фільтрів має свої унікальні характеристики та використовується для певних завдань. Наприклад, lowpass фільтр пропускає низькі частоти і приглушує високі, тоді як highpass фільтр робить протилежне.

Однією з основних властивостей BiquadFilterNode є можливість налаштування частоти зрізу (frequency), добротності (Q) та підсилення (gain). Частота зрізу визначає точку, на якій фільтр починає впливати на аудіосигнал, добротність впливає на ширину смуги частот, що підлягають фільтрації, а підсилення використовується у шелфових та пікових фільтрах для збільшення або зменшення рівня сигналу.

**Опис деталей впровадження**

За основу був взятий програмний код другої лабораторної роботи, у якій реалізовано стереозображення поверхні та відмалювання зображення з веб камери на фоні

Першим кроком для виконання розрахункової роботи необхідно було створити об’єкт, в нашому випадку це сфера, яка обертається навколо поверхні, для цього необхідно було створити код, який буде генерувати точки сфери та додати його виклик у draw() функцію. Змінювати шейдер (shader.gpu) під потреби розрахунково-графічної роботи не довелось. (рис.1)

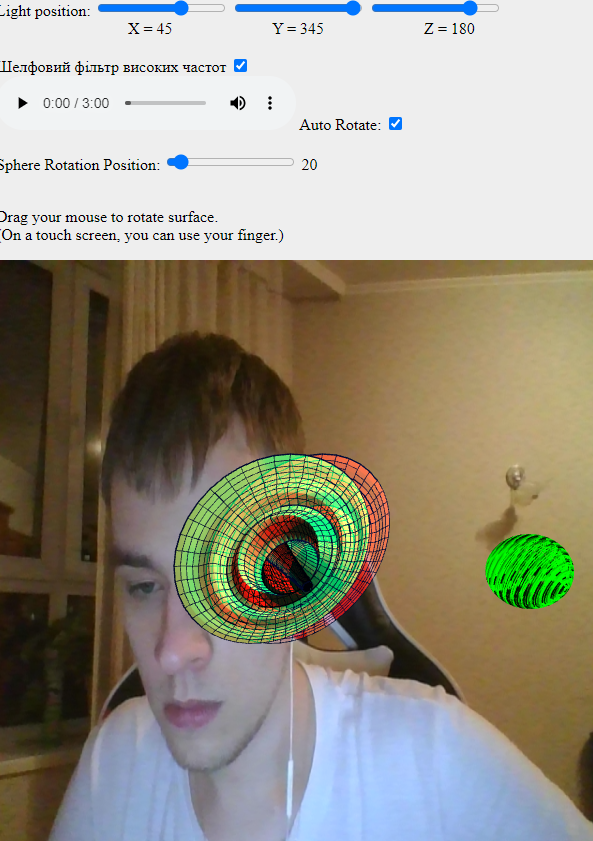


Рисунок 1 – Оберт сфери навколо поверхні

Також для контролю положення сфери було передбачення виведення поточного розташування сфери у вигляді повзунка. Також було додано елемент прапорця(checkbox) який при натисненні буде здійснювати оберт сфери автоматично з часом. (рис. 2)

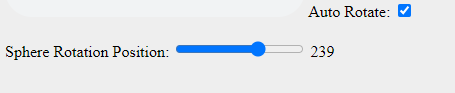


Рисунок 2 – Панель контролю положення сфери

Для контролю відтворення звіку, було додано <audio> об'єкт на HTML сторінку, також було додано елемент прапорця(checkbox) який при натисненні буде застосовувати фільтр на аудіодоріжку. (рис. 3)

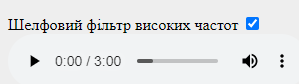


Рисунок 3 – Панель контролю аудіодоріжки

Прив’язку положення джерела звуку до положення сфери було здійснено за допомогою Web Audio API, а саме застосувавши context.createPanner(); для створення об’єкта звуку та зміни його координат відповідно координатам центра сфери.

Також в РГР необхідно було здійснити накладання фільтрів. Це було реалізовано за допомогою BiquadFIlterNode, згідно варіанту було встановлено Шельфовий фільтр високих частот або “highshelf” у налаштуваннях BiquadFIlterNode. Також було обрано довільні значення в конфігурації фільтра:

* gain = -35
* frequency = 2500

Така конфігурація дозволить зменшувати гучність на 35 dB для звуків, які перевищують частоту >2500

**Інструкції користувачу**

Головним вікном нашої програми є полотно на якому зображена поверхня та сфера, яка представляє собою джерело звуку. (рис. 4)

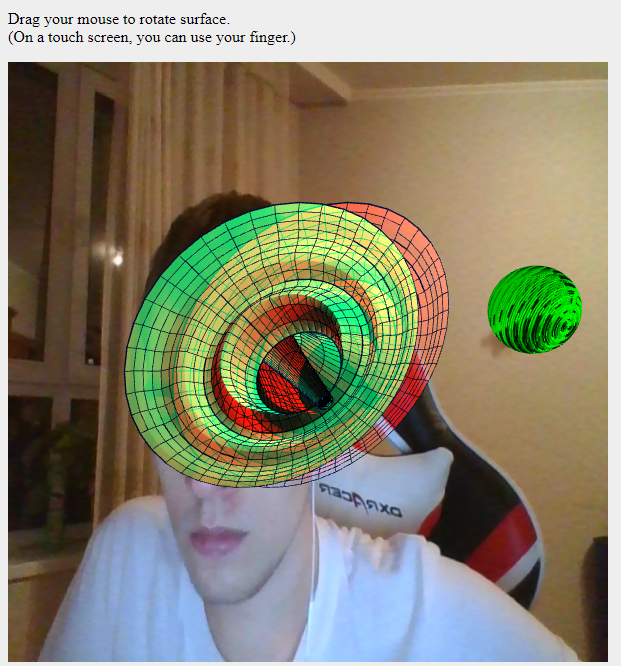


Рисунок 4 – Полотно з стереозображенням поверхні та сферою(джерелом звуку)

Користувач може обертати відмальовану сцену за допомогою рухів вказівника по полотну з затисненою ЛКМ.

Над полотном знаходиться панель керування, яка складається з трьох блоків:

* Налаштування поверхні та стереозображення
* Налаштуванням освітлення
* Налаштування аудіо
* Керування джерелом звуку

Схема панелі керування зображена на Рисунку 5

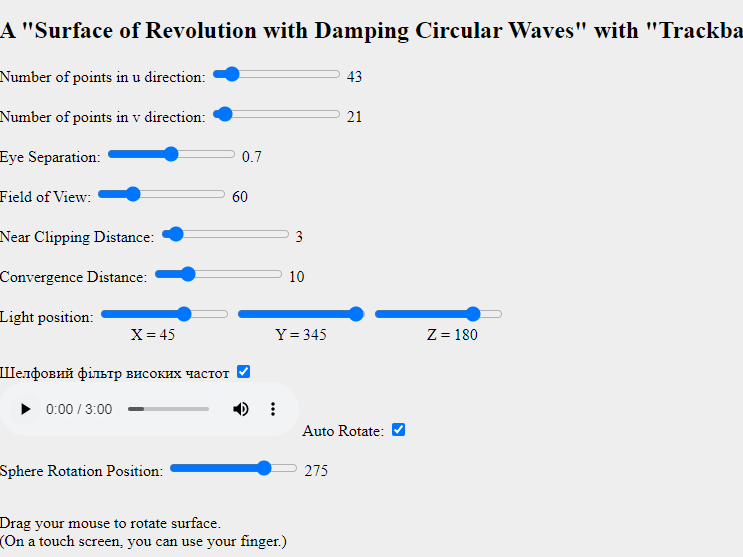


Рисунок 5 – Панель керування

В налаштуваннях поверхні та стереозображення користувач може змінювати якість поверхні, що зможе збільшити час виконання інших дій.

Далі користувач може робити налаштування стереозображення, змінюючи такі параметри як:

* Відстань між очим (Eye separation)
* Поле зору (Field of View)
* Близьку відстань відсікання (Near clipping Distance)
* Відстань конвергенції (Convergence Distance)

В налаштуваннях освітлення користувач може задавати координати з якого відбувається прожекторне освітлення.

В налаштуваннях аудіо користувач може запускати/зупиняти та перемотувати звукову доріжку, яка буде слідувати за положенням сфери та накладати Шелфовий фільтр високих частот на неї. Поточні налаштування фільтру стишують високі частоти на 35 dB.

І в кінці панелі керування знаходиться керування джерелом звуку, тут користувач може увімкнути автоматичне переміщення джерела звуку або вимкнути його і налаштовувати позицію вручну змінюючи значення рухом повзунка Sphere Rotation Position.

**Зразок вихідного коду**

Модель поверхні

function Model(name) {

    this.name = name;

    this.iVertexBuffer = gl.createBuffer();

    this.iNormalBuffer = gl.createBuffer();

    this.count = 0;

    this.BufferData = 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.NormalBufferData = function (normalsList) {

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

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

        this.count = normalsList.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.iNormalBuffer);

        gl.vertexAttribPointer(shProgram.iAttribNormal, 3, gl.FLOAT, false, 0, 0);

        gl.enableVertexAttribArray(shProgram.iAttribNormal);

        gl.drawArrays(gl.TRIANGLES, 0, this.count);

    }

    this.DrawVisualDetails = 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.LINES, 0, this.count);

    }

}

Модель сфери

function Sphere(name) {

    this.iVertexBuffer = gl.createBuffer();

    this.name = name;

    this.count = 0;

    this.BufferData = 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.generateSphereData = function(centerX, centerY, centerZ, r) {

        const vertexPositionData = CreateSphereSurface(centerX, centerY, centerZ, r);

        this.BufferData(vertexPositionData);

    }

    this.Draw = function(projectionViewMatrix, angle) {

        let translation = m4.translation(Math.cos(angle) \* 2, 0, Math.sin(angle) \* 2);

        let modelMatrix = m4.multiply(translation, spaceball.getViewMatrix());

        let modelViewProjection = m4.multiply(projectionViewMatrix, modelMatrix);

        gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, modelViewProjection);

        gl.uniform4fv(shProgram.iColor, [0.0, 1.0, 0.0, 1.0]);

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

        gl.vertexAttribPointer(shProgram.iAttribVertex, 3, gl.FLOAT, false, 0, 0);

        gl.enableVertexAttribArray(shProgram.iAttribVertex);

        gl.drawArrays(gl.LINE\_STRIP, 0, this.count);

    }

}

Модель зображення веб камери

function WebCameraImageModel(name){

    this.name = name;

    this.iVertexBuffer = gl.createBuffer();

    this.iTextureBuffer = gl.createBuffer();

    this.texture = gl.createTexture();

    this.count = 0;

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

    this.BufferData = 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.TextureBufferData = function (vertices) {

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

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

    }

    this.Draw = function () {

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

        gl.vertexAttribPointer(shProgram.iAttribVertex, 3, gl.FLOAT, false, 0, 0);

        gl.enableVertexAttribArray(shProgram.iAttribVertex);

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

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

        gl.enableVertexAttribArray(shProgram.aTexCoord);

        gl.uniform1i(shProgram.uTexture, 0);

        gl.drawArrays(gl.TRIANGLES, 0, this.count);

    }

}

draw() функція

function draw(animate = false) {

    gl.clearColor(0, 0, 0, 1);

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

    gl.uniform1f(shProgram.iUseColor, false);

    /\* Set up the stereo camera system \*/

    initStereoCamera();

    // General settings

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

    let viewMatrix = spaceball.getViewMatrix();

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

    let rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0.7);

    let translateToPointZero = m4.translation(0, 0, -5);

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

    let matAccum1 = m4.multiply(translateToPointZero, matAccum0);

    // Web Camera texture

    gl.uniform1f(shProgram.iUseTexture, true);

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

    gl.texImage2D(

        gl.TEXTURE\_2D,

        0,

        gl.RGBA,

        gl.RGBA,

        gl.UNSIGNED\_BYTE,

        webCamElement

    );

    gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, m4.identity());

    webCamModel.Draw();

    gl.clear(gl.DEPTH\_BUFFER\_BIT)

    gl.uniform1f(shProgram.iUseTexture, false);

    // Left eye

    stereoCamera.ApplyLeftFrustum();

    gl.colorMask(true, false, false, true); // Only draw red channel

    let projection = stereoCamera.projection;

    let modelView = stereoCamera.modelView;

    /\* Multiply the projection matrix times the modelview matrix to give the

       combined transformation matrix, and send that to the shader program. \*/

    let modelViewProjection = m4.multiply(projection, m4.multiply(modelView, matAccum1));

    gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, modelViewProjection);

    settingUpLightingScene(modelViewProjection);

    surface.Draw();

    drawMesh();

    // Clear depth buffer to prepare for right eye rendering

    gl.clear(gl.DEPTH\_BUFFER\_BIT);

    // Right eye

    stereoCamera.ApplyRightFrustum();

    gl.colorMask(false, true, true, true); // Only draw green and blue channels

    projection = stereoCamera.projection;

    modelView = stereoCamera.modelView;

    /\* Multiply the projection matrix times the modelview matrix to give the

       combined transformation matrix, and send that to the shader program. \*/

    modelViewProjection = m4.multiply(projection, m4.multiply(modelView, matAccum1));

    gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, modelViewProjection);

    settingUpLightingScene(modelViewProjection);

    surface.Draw();

    drawMesh();

    gl.colorMask(true, true, true, true); // Restore color mask to default

    // Animate the sphere

    gl.uniform1f(shProgram.iUseColor, true);

    let angle;

    if (autoRotate) {

        const time = Date.now() \* 0.001;

        angle = time \* (360 / (2 \* Math.PI)); // Convert time to degrees

        rotationPosition = angle % 360;

        document.getElementById('rotationSlider').value = rotationPosition;

        document.getElementById('rotationValue').innerText = Math.floor(rotationPosition);

    } else {

        angle = rotationPosition;

    }

    const radians = angle \* Math.PI / 180;

    sphere.generateSphereData(Math.cos(radians) \* 2, 0, Math.sin(radians) \* 2, sphereRadius);

    sphere.Draw(modelViewProjection, radians);

    gl.uniform1f(shProgram.iUseColor, false);

    if (animate) {

        window.requestAnimationFrame(() => draw(true));

    }

}

shader.gpu

// Vertex shader

const vertexShaderSource = `

attribute vec3 vertex;

attribute vec3 normal;

varying vec3 normalInterp;

varying vec3 fragPos;

varying vec3 lightDir;

uniform mat4 ModelViewProjectionMatrix;

uniform mat4 NormalMatrix;

uniform vec3 lightPos;

varying vec2 v\_texCoord;

attribute vec2 a\_texcoord;

void main() {

    gl\_Position = ModelViewProjectionMatrix \* vec4(vertex, 1.0);

    fragPos = vec3(ModelViewProjectionMatrix \* vec4(vertex, 1.0));

    normalInterp = normalize(vec3(NormalMatrix \* vec4(normal, 0.0)));

    lightDir = normalize(lightPos - fragPos);

    v\_texCoord = a\_texcoord;

    gl\_PointSize = 50.0;

}

`;

// Fragment shader

const fragmentShaderSource = `

#ifdef GL\_FRAGMENT\_PRECISION\_HIGH

   precision highp float;

#else

   precision mediump float;

#endif

varying vec3 normalInterp;

varying vec3 fragPos;

varying vec3 lightDir;

uniform bool useColor;

uniform bool useTexture;

uniform vec4 color;

uniform sampler2D u\_texture;

varying vec2 v\_texCoord;

void main() {

    if(useTexture){

        vec4 texturedColor = texture2D(u\_texture, v\_texCoord);

        gl\_FragColor = texturedColor;

    }else{

        vec3 ambientColor = vec3(0.1, 0.1, 0.05);

        vec3 diffuseColor = vec3(1.0, 1.0, 0.5);

        vec3 specularColor = vec3(0.0, 0.0, 0.0);

        vec3 normal = normalize(normalInterp);

        vec3 lightDirNorm = normalize(lightDir);

        vec3 viewDir = normalize(-fragPos);

        float lambertian = max(dot(normal, lightDirNorm), 0.0);

        float specular = 0.0;

        if (lambertian > 0.0) {

            vec3 reflectDir = reflect(-lightDirNorm, normal);

            float specAngle = max(dot(reflectDir, viewDir), 0.0);

            specular = pow(specAngle, 32.0);

        }

        vec3 lighting = ambientColor + (lambertian \* diffuseColor) + (specular \* specularColor);

        if(useColor){

            gl\_FragColor = color \* vec4(lighting, 1.0);

        }else{

            gl\_FragColor = vec4(lighting, 1.0);

        }

    }

}

`;

initAudio()

function initAudio() {

    audio = document.getElementById('audio');

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

        if (!context) {

            context = new AudioContext();

            audioSrc = context.createMediaElementSource(audio);

            panner = context.createPanner();

            biquadFilter = context.createBiquadFilter();

            audioSrc.connect(panner);

            panner.connect(biquadFilter);

            biquadFilter.connect(context.destination);

            // High-shelf filter (variant 19)

            biquadFilter.type = 'highshelf';

            biquadFilter.frequency.setValueAtTime(2500, context.currentTime); // for sound with frequency more then

            biquadFilter.gain.setValueAtTime(-35, context.currentTime); // -N amount of dB

            context.resume();

        }

    });

    audio.addEventListener('pause', () => context.resume());

    applyFilter = document.getElementById('check-filter');

    applyFilter.addEventListener('change', () => {

        if (applyFilter.checked) {

            panner.disconnect();

            panner.connect(biquadFilter);

            biquadFilter.connect(context.destination);

        } else {

            panner.disconnect();

            panner.connect(context.destination);

        }

    });

}