Міністерство освіти і науки України

Національний технічний університет України

«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”  
на тему “Spatial audio”

Варіант 12

Виконав студент групи ТР-31мп  
Кривов’язюк Михайло Олексійович

Київ 2024

1. **Завдання**

Імплементувати просторове аудіо за допомогою WebAudio HTML5 API, використовуючи код з практичного завдання 2.

Імплементувати обертання джерела звуку відносно геометричного центру поверхні з використанням дотичного інтерфейсу (смартфону, що використовує сенсори для обробки інформації, щодо положення в просторі). На відміну від попередніх робіт поверхня залишається нерухомою - рухається джерело звуку. Відтворити улюблену пісню в форматі mp3/ogg, з можливістю контролювати місцезнаходження джерела звуку в просторі користувачем.

Візуалізувати джерело звуку за допомогою сфери.

Додати звуковий фільтр за варіантом (використовуючи інтерфейс BiquadFilterNode). Додати інтерфейс користувача, який би вмикав/вимикав фільтр. Встановити параметри фільтру відповідно до вподобань.

Підготувати звіт в цифровому вигляді, який би містив необхідні частини, що сповна описують поставлені задачі та виконану роботу.

1. **Теоретичні відомості**

**WebAudio API**

До цього часу аудіо в Інтернеті було досить примітивним і його доводилося відтворювати за допомогою плагінів, таких як Flash і QuickTime. Введення елемента <audio> в HTML5 стало важливим кроком, оскільки дозволило відтворювати базове потокове аудіо без додаткових плагінів. Проте, цей елемент не достатньо потужний для більш складних аудіододатків. Для розробки складних веб-ігор або інтерактивних програм потрібне інше рішення.

Метою цієї специфікації є надання можливостей сучасних звукових ігрових движків, а також функцій мікшування, обробки та фільтрації, які доступні в сучасних настільних аудіопрограмах. Інтерфейси API були розроблені для підтримки широкого спектру випадків використання [webaudio-usecases]. В ідеалі, API повинен підтримувати будь-які сценарії, які можна ефективно реалізувати за допомогою оптимізованого C++ механізму, керованого через сценарії та виконуваного в браузері.

Проте, сучасне аудіопрограмне забезпечення для настільних комп'ютерів може мати дуже розширені функції, деякі з яких важко або неможливо реалізувати за допомогою цієї системи. Наприклад, Apple Logic Audio підтримує зовнішні MIDI-контролери, довільні аудіоефектні плагіни та синтезатори, високооптимізоване читання/запис аудіофайлів безпосередньо на диск, інтегроване розтягування часу тощо. Проте, запропонована система здатна підтримувати широкий спектр досить складних ігор та інтерактивних програм, включаючи музичні, і може стати хорошим доповненням до розширених графічних функцій, які пропонує WebGL. API розроблений з можливістю подальшого додавання розширених функцій.

**BiquadFilterNode**

BiquadFilterNode — це процесор AudioNode, який реалізує дуже поширені фільтри нижчого порядку.

Фільтри нижчого порядку є будівельними блоками базових регуляторів тембру (баси, середні та високі частоти), графічних еквалайзерів і більш розширених фільтрів. Кілька фільтрів BiquadFilterNode можна комбінувати для створення більш складних фільтрів. Параметри фільтра, такі як частота, можна змінювати з часом для розгортки фільтра тощо. Кожен BiquadFilterNode можна налаштувати як один із кількох загальних типів фільтрів, як показано в IDL нижче. Тип фільтра за замовчуванням – «НЧ».

Фільтр низьких частот пропускає частоти нижче граничної частоти та послаблює частоти вище граничної. Він реалізує стандартний резонансний фільтр низьких частот другого порядку зі спадом 12 дБ/октаву.

Фільтр високих частот є протилежністю фільтру низьких частот. Частоти вище граничної частоти пропускаються, але частоти нижче граничної послаблюються. Він реалізує стандартний резонансний фільтр високих частот другого порядку зі спадом 12 дБ/октаву.

Смуговий фільтр пропускає діапазон частот і послаблює частоти нижче та вище цього діапазону частот. Він реалізує смуговий фільтр другого порядку.

Фільтр Lowshelf пропускає всі частоти, але додає підсилення (або ослаблення) нижніх частот. Він реалізує фільтр низького рівня другого порядку.

Фільтр Highshelf є протилежністю фільтру Lowshelf і пропускає всі частоти, але додає посилення до вищих частот. Він реалізує фільтр високої полиці другого порядку

Піковий фільтр пропускає всі частоти, але додає підсилення (або ослаблення) до діапазону частот.

Режекторний фільтр (також відомий як смуговий або смуговий фільтр) є протилежністю смуговому фільтру. Він дозволяє пропускати всі частоти, крім набору частот.

1. **Аспекти імплементації**

В результаті виконання 3 та 4 лабораторних робіт було розроблено матеріальний інтерфейс: смартфон, який дозволяє користувачу обертати фігуру згідно показів сенсорів смартфона, щодо його відносного положення в просторі.

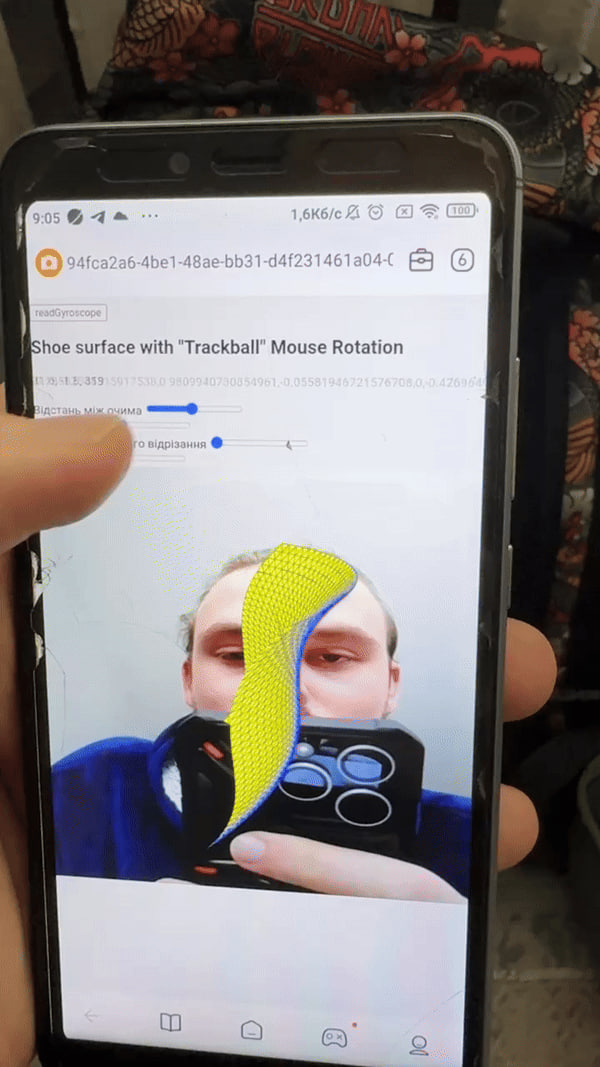


Рисунок 3.1 - скріншот 3 або 4 лаби

Реалізував обертання джерела звуку навколо геометричного центру ділянки поверхні круговим способом протягом певного часу (цього разу поверхня залишається нерухомою, а джерело звуку рухається). Джерело звуку візуалізовано у вигляді сферичної геометрії

В подальшому цей функціонал дозволить керувати положенням джерела звуку просторового аудіо всередині програми.

Першим кроком для імплементації просторового аудіо є створення HTML-елементу <audio>, який містить інформацію про джерело аудіо-доріжки в моєму випадку це “Baby\_Lasagna\_Rim\_Tim\_Tagi\_Dim” в форматі mp3. Цей елемент керування дозволить зупиняти та продовжувати відтворення аудіо-доріжки. Після цього за допомогою JavaScript, а саме WebAudio API було створено обʼєкт аудіоконтексту (AudioContext), для якого було створено та підʼєднано 3 основних обʼєкти:

* джерело звуку (MediaElementSource)
* обʼєкт обробки просторового аудіо (Panner)
* звуковий фільтр (BiquadFilter)

Згідно з варіантом було обрано Шелфовий фільтр високих частот.

Наступним кроком було встановлено параметри обраного фільтра. Взагалі, BiquadFilter має 3 параметри для налаштування:

* frequency (частота)
* Q (ширина смуги)
* gain (підсилення)

Особливість Шелфового фільтру високих частот в тому, що

для highshelf фільтру

frequency - нижня межа частот, на яких застосовується підсилення (або ослаблення).

Q - не використовується в цьому типі фільтра.

gain - підсилення, у дБ, яке буде застосовано. Якщо значення від'ємне, частоти послаблюються.

Джерело звуку, обробка якого в просторі здійснюється обʼєктом класу Panner зображено у WebGL контексті у вигляді сфери, щоб може переміщуватись - відповідно змінюючи параметри обʼєкту Panner, щоб при прослуховуванні аудіо зі стерео звуком був ефект переміщення джерела звуку відповідно до місцезнаходження сфери в системі координат.

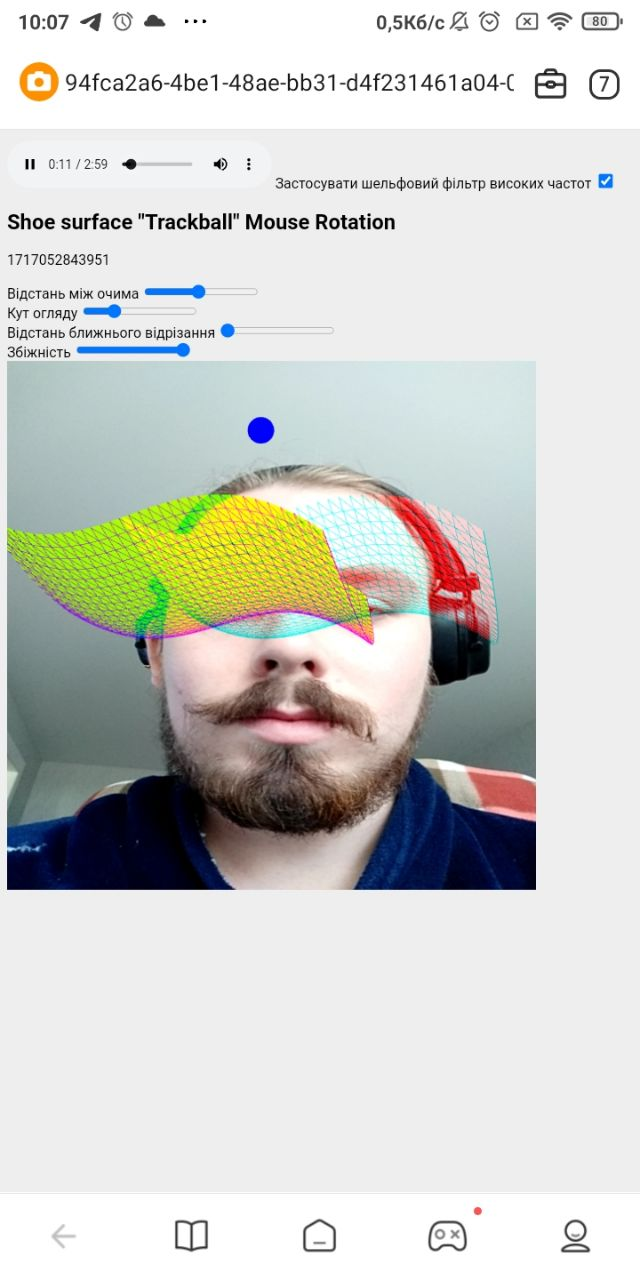


Рисунок 3.2 - скріншот РГР

1. **Інструкція користувача**

Для налаштування стерео зображення використовуються 4 слайдери, в який можна налаштувати наступні параметри:

* Convergence (збіжність)
* Eye separation (відстань між очима)
* Field of view (поле зору)
* Near clipping distance (відстань ближньої площини, що відсікає зображену на екрані геометрію)

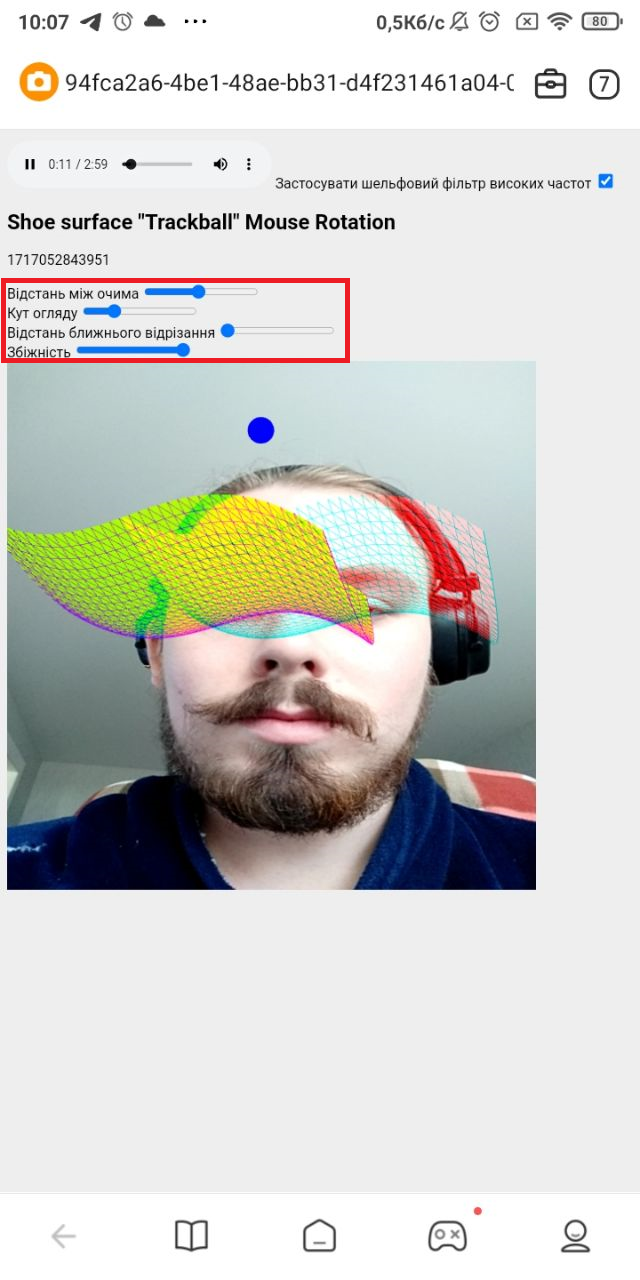


Рисунок 4.1 - скрішот слайдерів

Згідно до встановлених параметрів стерео зображення фігури виглядає дещо інакше

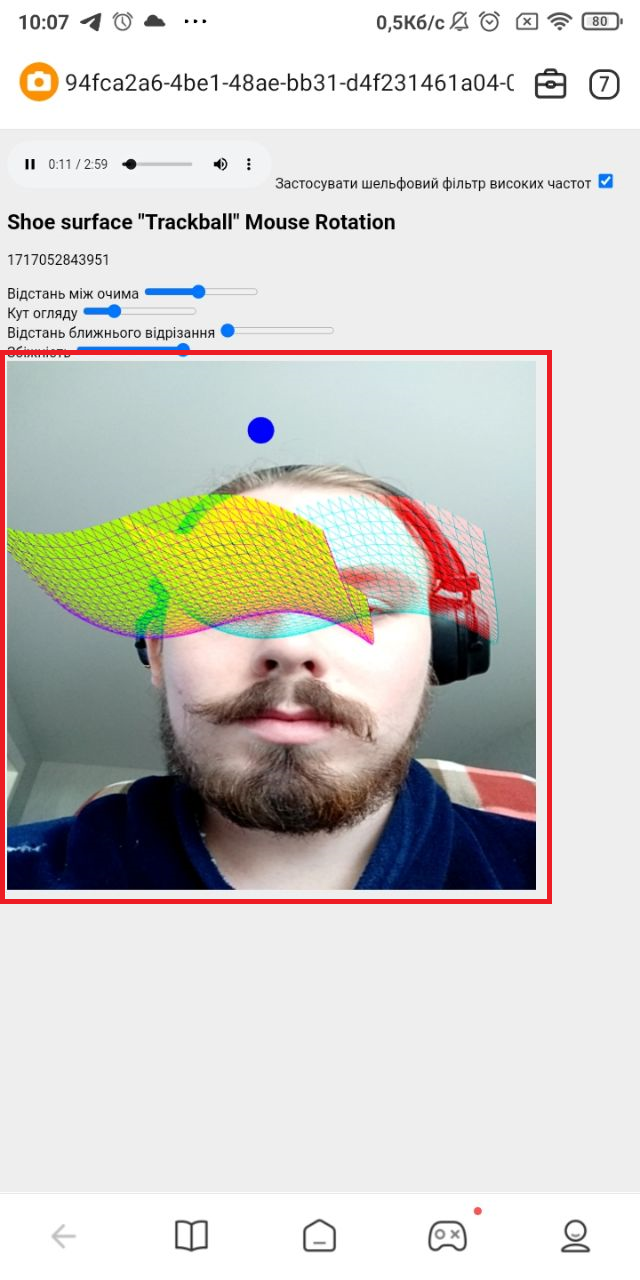


Рисунок 4.2 - скріншот фігури з початковими налаштуваннями

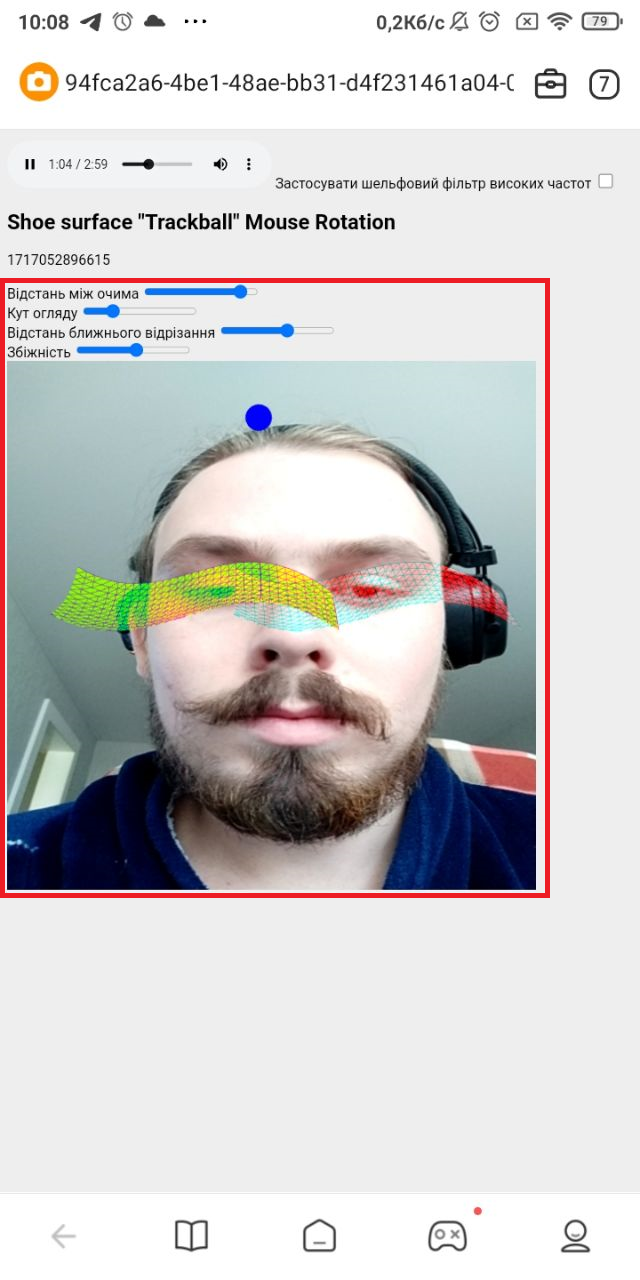


Рисунок 4.3 - скріншот фігури зі зміненими налаштуваннями

Вище згаданий HTML-елемент audio дозволяє керувати аудіо-доріжкою, а саме зупиняти та продовжувати відтворення, перемотувати на потрібний час, керувати гучністю.

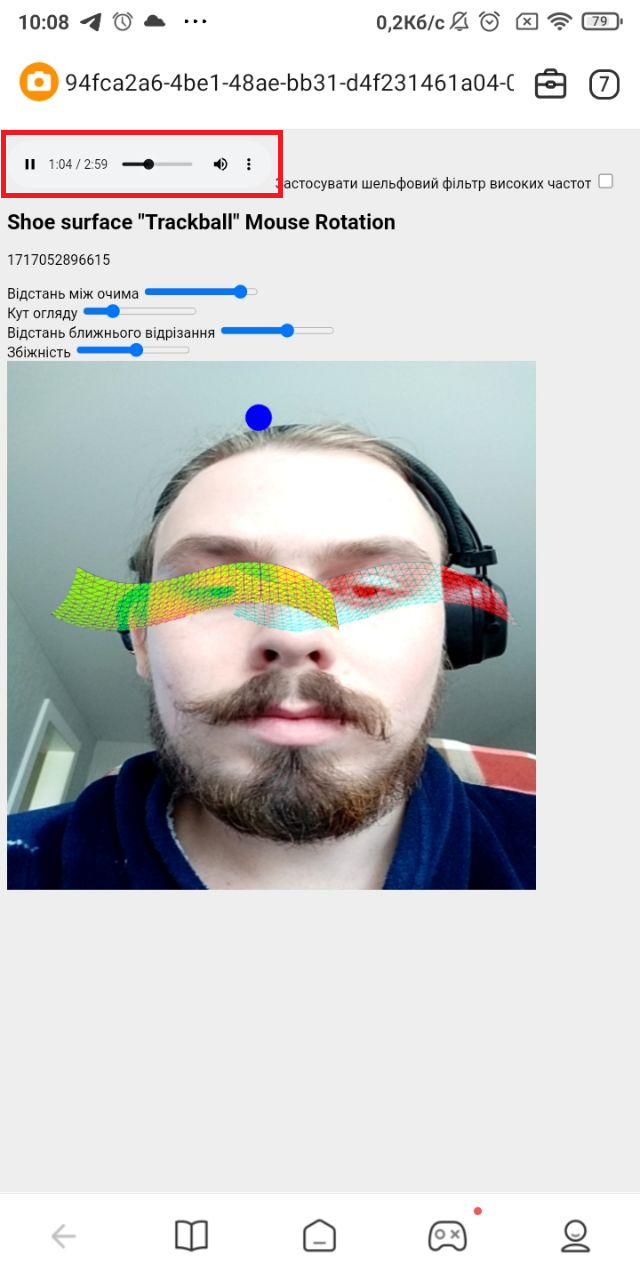


Рисунок 4.4 - скріншот з UI-елементом плеєра

“Чекбокс” дозволяє керувати станом звукового фільтру: вмикати та вимикати його. В залежності від стану фільтру можна помітити зміну звучання.

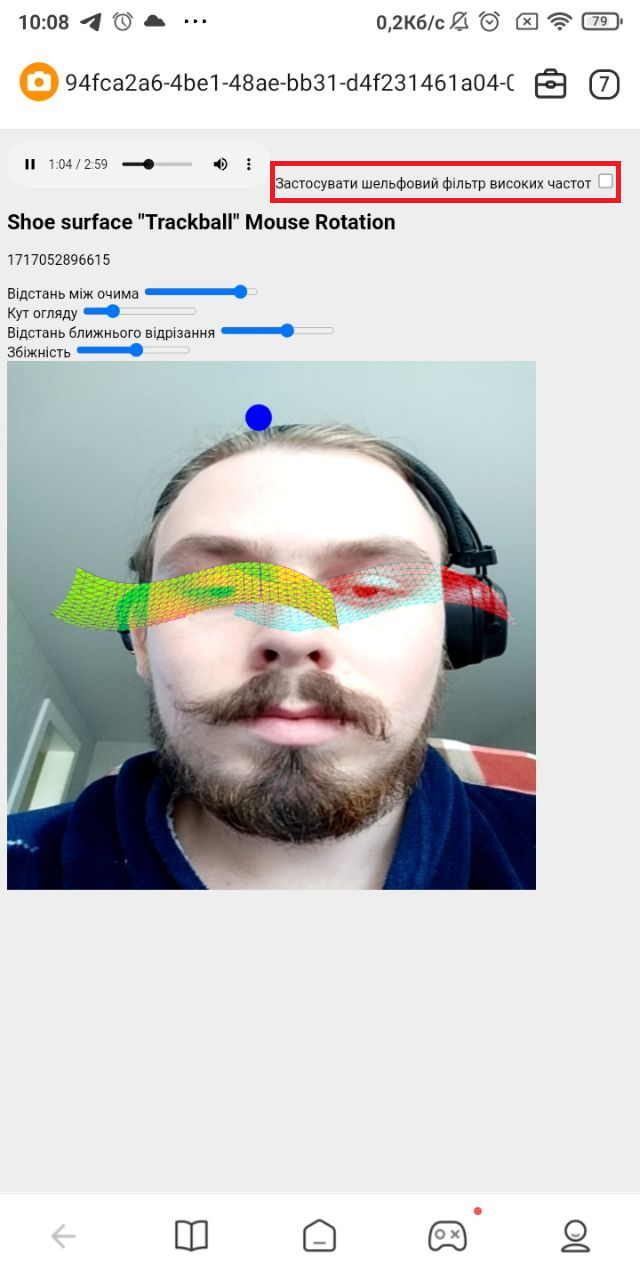


Рисунок 4.5 - скріншот з чекбоксом

1. **Код програми**

let context,

audio,

source,

biquadFilter,

panner;

function setupAudio() {

audio = document.getElementById('audio');

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

if (!context) {

context = new AudioContext();

source = context.createMediaElementSource(audio);

panner = context.createPanner();

biquadFilter = context.createBiquadFilter();

source.connect(panner);

panner.connect(biquadFilter);

biquadFilter.connect(context.destination);

biquadFilter.type = 'highshelf';

biquadFilter.gain.value = 25;

biquadFilter.frequency.value = 100;

context.resume();

}

})

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

console.log('pause');

context.resume();

})

}

function initAudio() {

setupAudio();

const peakingEnabled = document.getElementById('filterCheckbox');

peakingEnabled.addEventListener('change', function() {

if (peakingEnabled.checked) {

panner.disconnect();

panner.connect(biquadFilter);

biquadFilter.connect(context.destination);

} else {

panner.disconnect();

panner.connect(context.destination);

}

});

audio.play();

}

function anglesToVector(alpha, beta, gamma) {

// Convert angles to radians

const alphaRad = (alpha \* Math.PI) / 180;

const betaRad = (beta \* Math.PI) / 180;

const gammaRad = (gamma \* Math.PI) / 180;

// Define the initial vector along the x-axis

let vector = [0, 1, 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;

}