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 17

Виконала студентка групи ТР-32мп  
Правдохіна Крістіна Артурівна

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**

Аудіо в Інтернеті до цього моменту було досить примітивним і донедавна доводилося доставляти через плагіни, такі як Flash і QuickTime. Введення елемента аудіо в HTML5 є дуже важливим, оскільки дозволяє відтворювати базове потокове аудіо. Але він недостатньо потужний, щоб працювати зі складнішими аудіододатками. Для складних веб-ігор або інтерактивних програм потрібне інше рішення. Ціль цієї специфікації полягає в тому, щоб включити можливості сучасних ігрових звукових движків, а також деякі завдання мікшування, обробки та фільтрації, які є в сучасних настільних програмах створення звуку.

Інтерфейси програмного інтерфейсу були розроблені з урахуванням широкого спектру випадків використання [webaudio-usecases]. В ідеалі він повинен мати можливість підтримувати будь-який варіант використання, який можна розумно реалізувати за допомогою оптимізованого механізму C++, який керується через сценарій і запускається в браузері. Тим не менш, сучасне аудіопрограмне забезпечення для настільних комп’ютерів може мати дуже розширені можливості, деякі з яких було б важко або неможливо створити за допомогою цієї системи. Apple Logic Audio є однією з таких програм, яка підтримує зовнішні MIDI-контролери, довільні плагіни аудіоефектів і синтезаторів, високооптимізоване читання/запис аудіофайлів прямо на диск, тісно інтегроване розтягування часу тощо. Тим не менш, запропонована система буде цілком здатна підтримувати широкий спектр досить складних ігор та інтерактивних програм, включаючи музичні. І це може бути гарним доповненням до розширеніших графічних функцій, які пропонує WebGL. API розроблено таким чином, щоб пізніше можна було додати розширені можливості.

**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**

Реалізовано 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

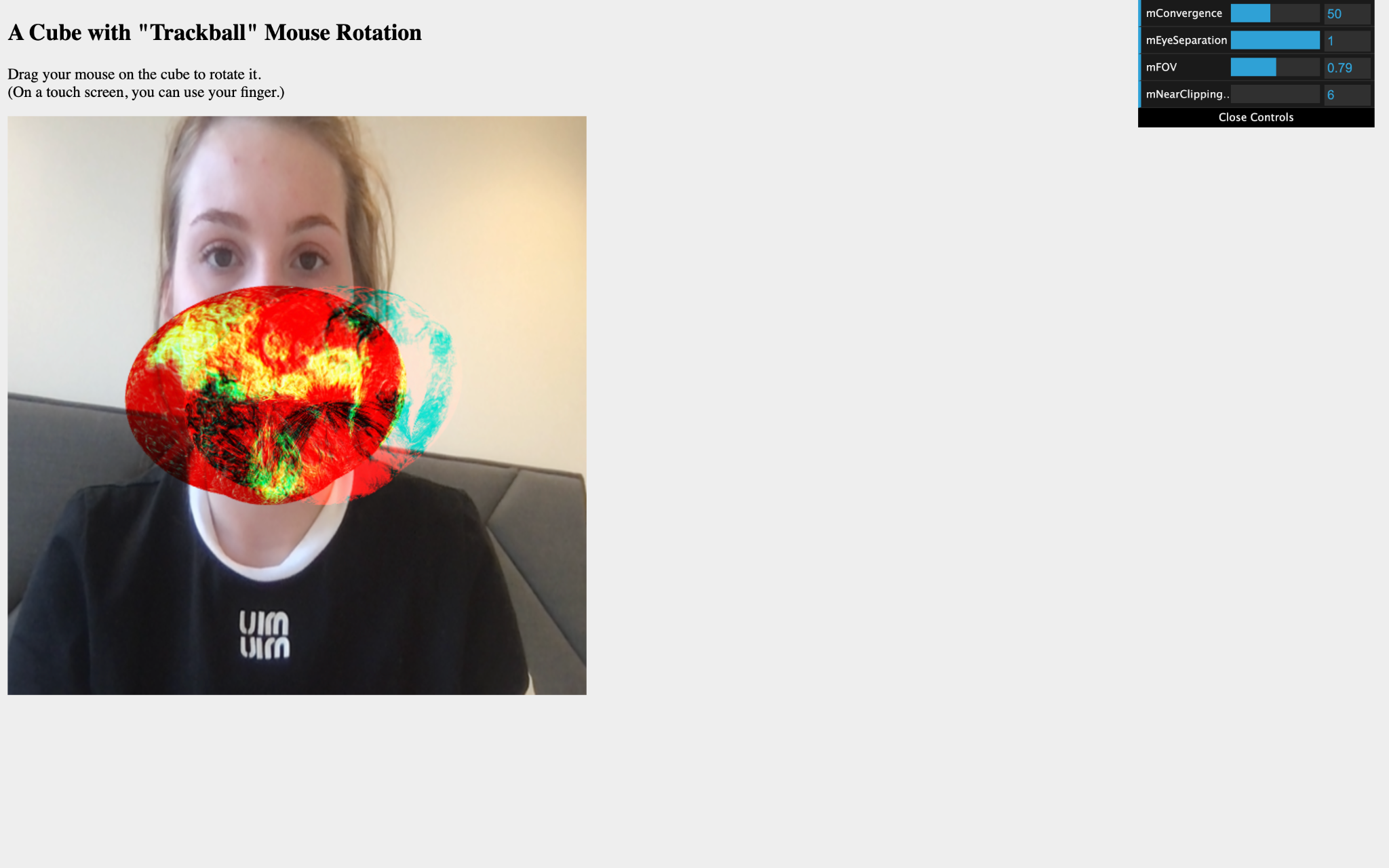


Рисунок 3.1 - веб-інтерфейс 2 лабораторної роботи

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 DOROFEEVA.mp3 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 смуговий фільтр.

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

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

Особливість смугового фільтра в тому, що

frequency - центр смуги частот

Q - контролює ширину смуги. Ширина стає вужчою зі збільшенням значення Q.

gain - не використовується в цьому типі фільтра

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.

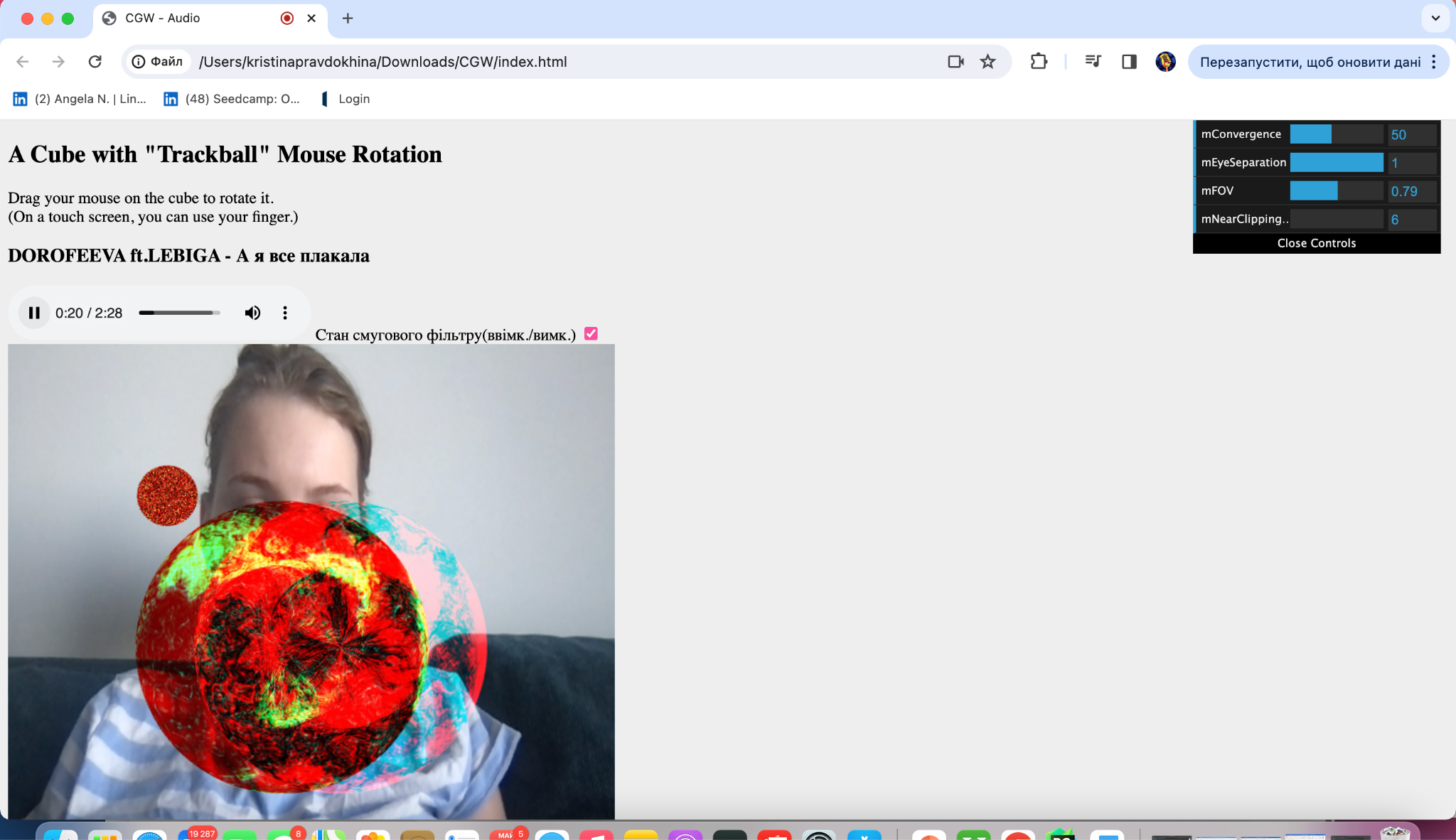


Рисунок 3.2 - веб-інтерфейс РГР

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)

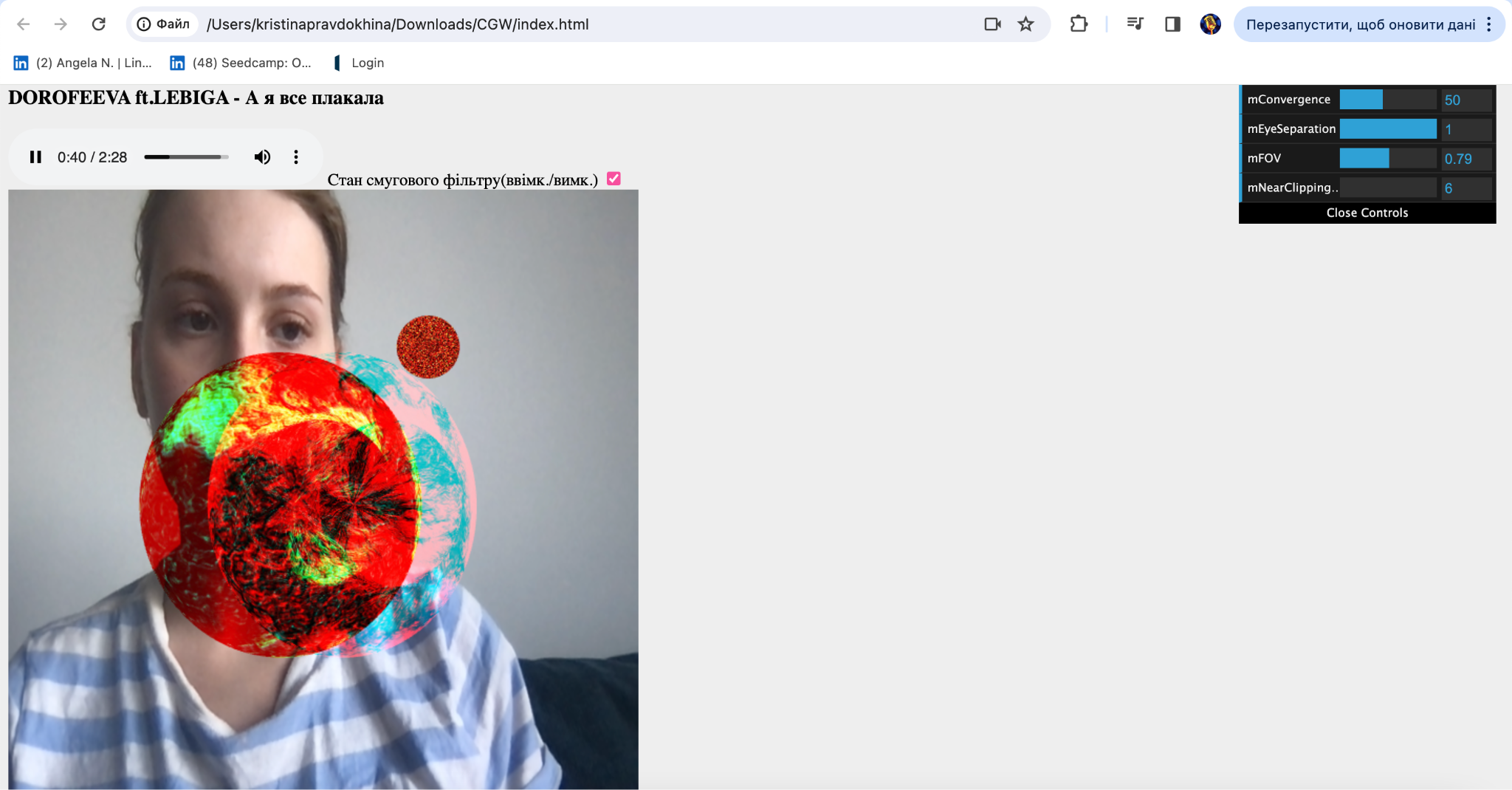


Рисунок 4.1 - слайдери, що керують стерео зображенням

Згідно до встановлених параметрів стерео зображення фігури виглядає дещо інакше

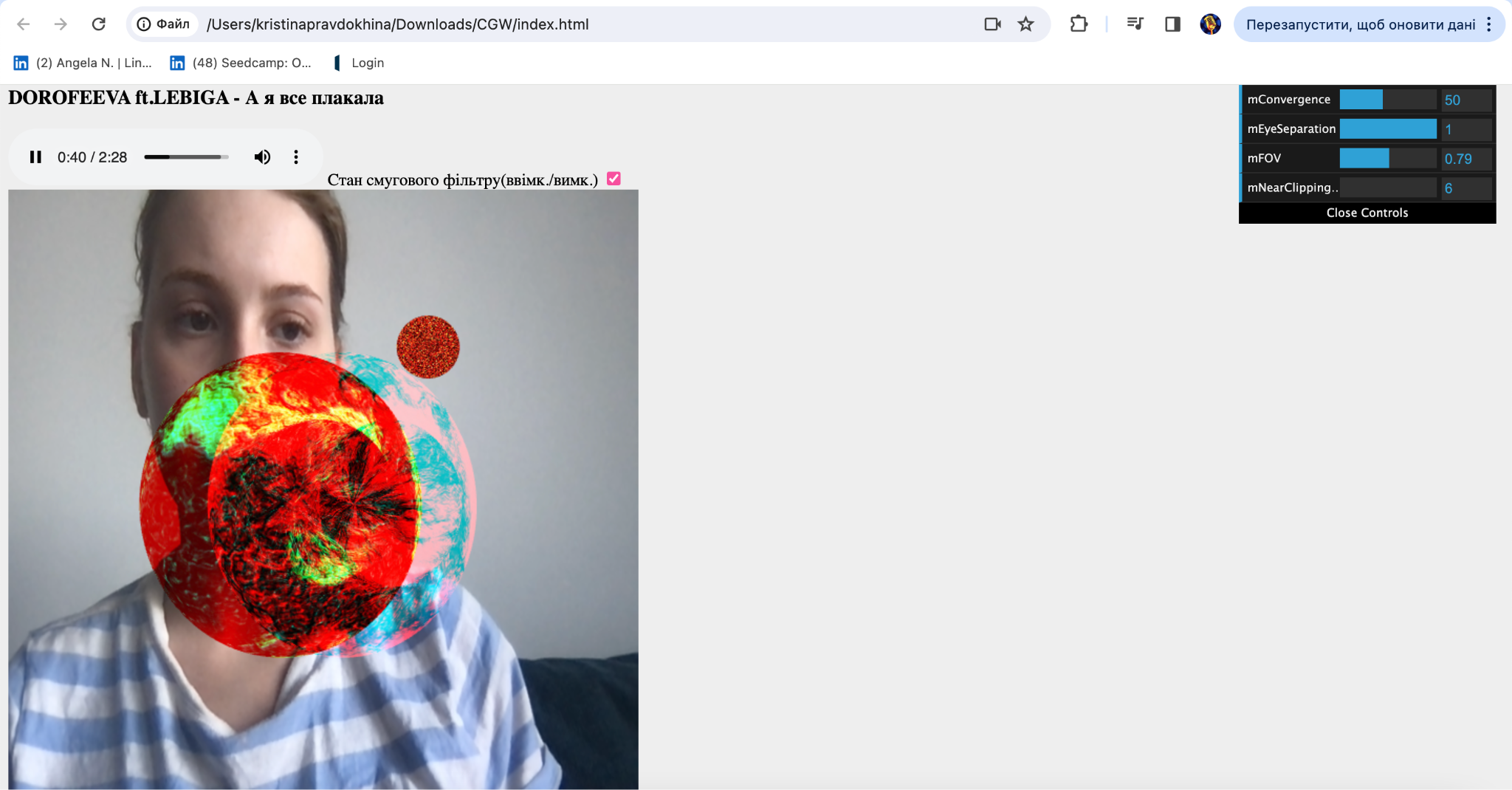


Рисунок 4.2 - скріншот фігури з початковими налаштуваннями

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

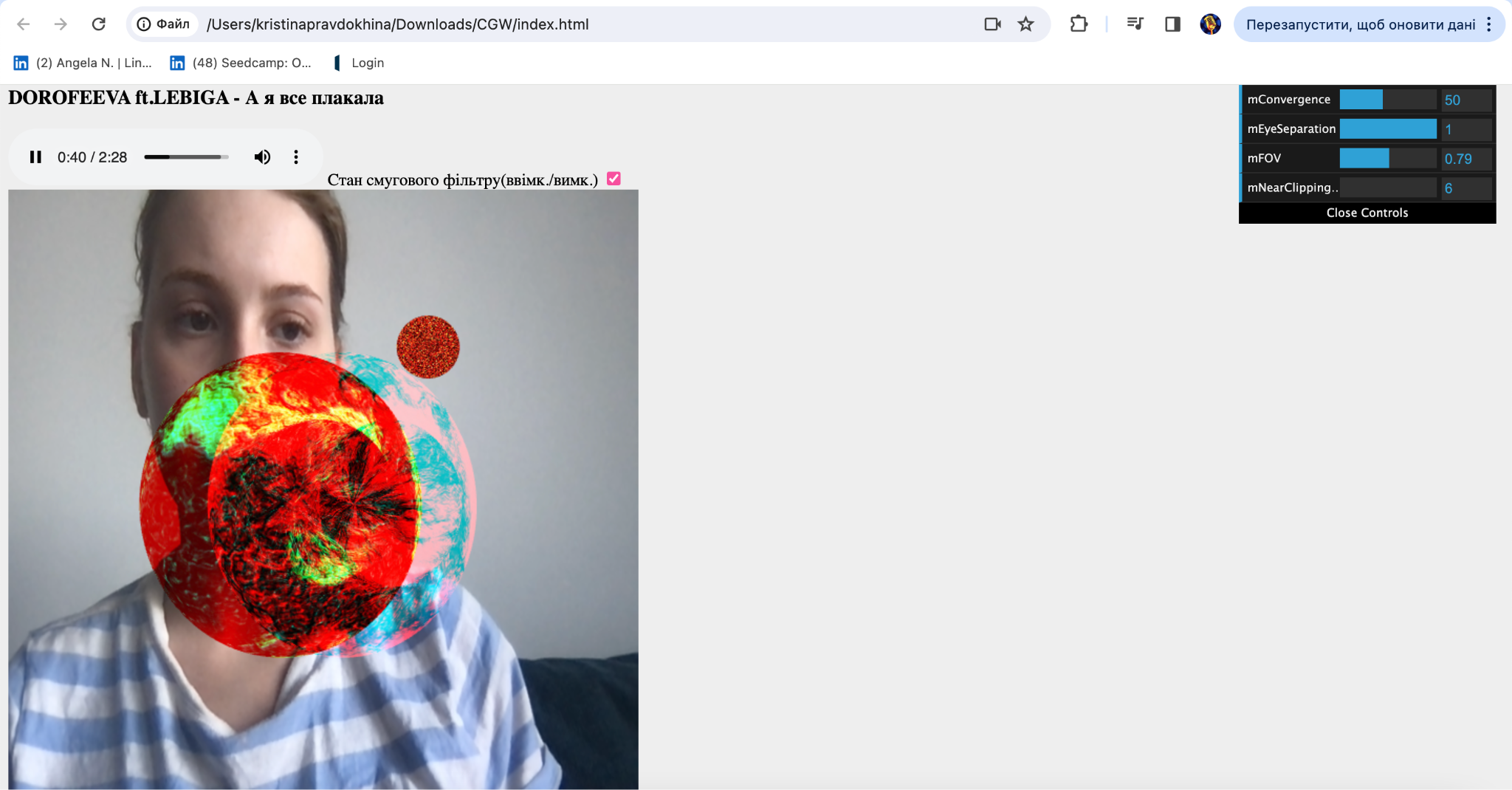


Рисунок 4.4 - скріншот з UI-елементом плеєра

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

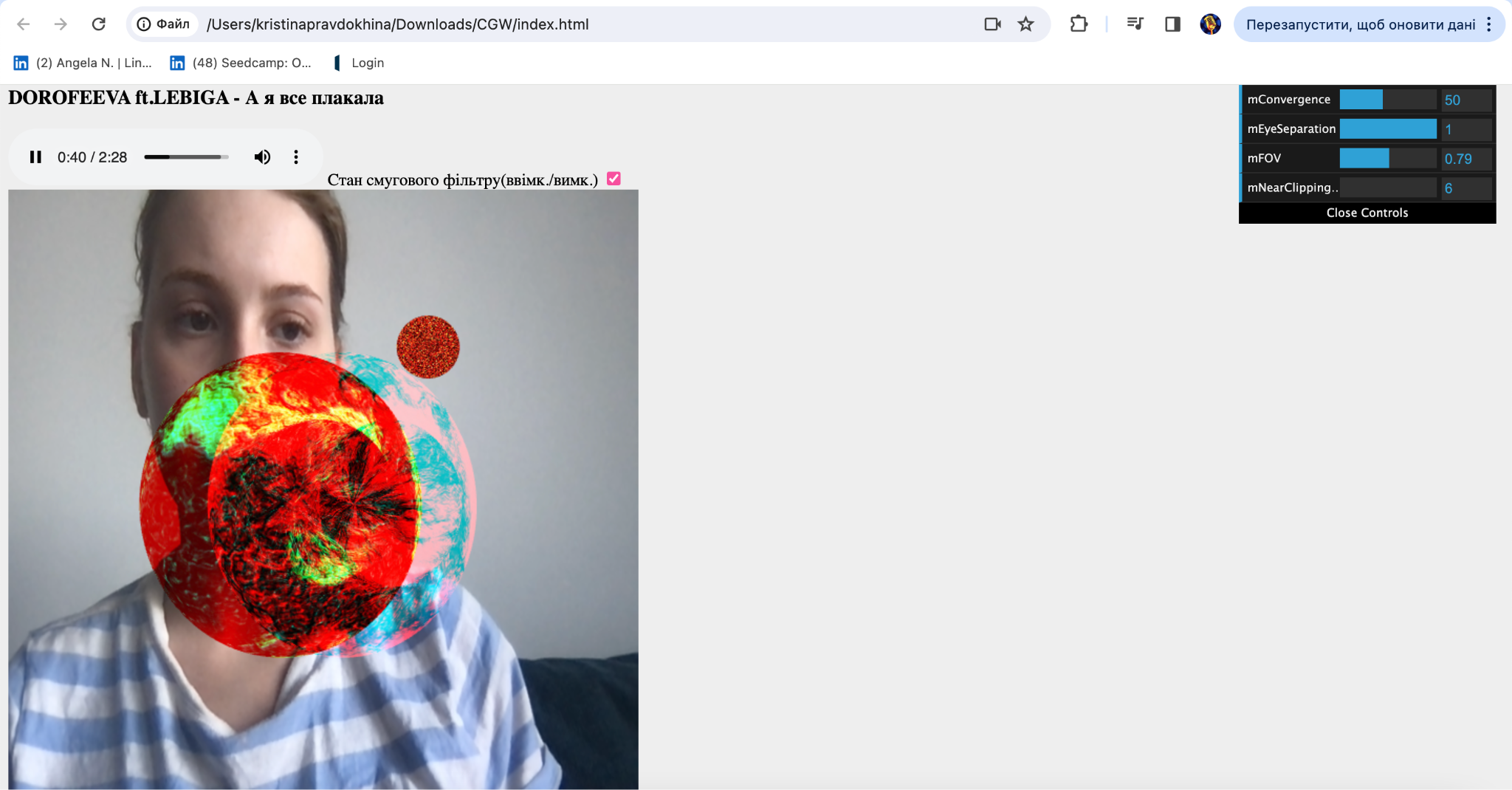


Рисунок 4.5 - скріншот з чекбоксом

1. **Код програми**

let filterCbx;

let audio;

let src;

let filter;

let stereoPanner;

let audioCtx;

let audioSrfc;

function initA() {

filterCbx = document.getElementById('filterCbx');

audio = document.getElementById('audioCtx');

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

if (!audioCtx) {

audioCtx = new AudioContext();

src = audioCtx.createMediaElementSource(audio);

stereoPanner = audioCtx.createPanner();

filter = audioCtx.createBiquadFilter();

src.connect(stereoPanner);

stereoPanner.connect(filter);

filter.connect(audioCtx.destination);

filter.type = 'bandpass';

filter.frequency.value = 10000;

filter.Q.value = 1;

// filter.gain.value = 1; // not used for bandpass filter according to the documentation

audioCtx.resume();

}

})

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

console.log('pause');

audioCtx.resume();

})

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

if (filterCbx.checked) {

stereoPanner.disconnect();

stereoPanner.connect(filter);

filter.connect(audioCtx.destination);

} else {

stereoPanner.disconnect();

stereoPanner.connect(audioCtx.destination);

}

});

audio.play();

}

function CreateSphere(radius) {

let vertexList = [];

let lon = -Math.PI; // longitude from -PI to PI

let lat = -Math.PI \* 0.5; // latitude from -PI/2 to PI/2

const STEP = 0.1;

while (lon < Math.PI) {

while (lat < Math.PI \* 0.5) {

let v1 = getVFromLonLat(radius, lon, lat);

let v2 = getVFromLonLat(radius, lon + STEP, lat);

let v3 = getVFromLonLat(radius, lon, lat + STEP);

let v4 = getVFromLonLat(radius, 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 getVFromLonLat(radius, u, v) {

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

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

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

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

}