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 3

Виконав студент групи ТР-32мп  
Васильцов Кирило Дмитрович

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 недавнього часу аудіо в Інтернеті було досить примітивним і вимагало використання плагінів, таких як Flash і QuickTime. Введення аудіоелемента в HTML5 стало важливим кроком, оскільки воно дозволило відтворювати прості аудіопотоки. Однак цього недостатньо для складних аудіододатків, які використовуються в сучасних веб-іграх або інтерактивних програмах. Ця специфікація має на меті забезпечити функціонал, подібний до сучасних ігрових звукових движків, а також можливості мікшування, обробки та фільтрації, які є в сучасних настільних аудіопрограмах.

Інтерфейси цього API були створені для підтримки широкого спектра випадків використання, зазначених у [webaudio-usecases]. Ідеально, API повинен підтримувати будь-який сценарій, який може бути реалізований оптимізованим механізмом C++, керованим через скрипти та запущеним у браузері. Втім, сучасне аудіопрограмне забезпечення для настільних комп'ютерів може мати дуже розширені можливості, які було б важко або неможливо відтворити з використанням цієї системи. Наприклад, Apple Logic Audio підтримує зовнішні MIDI-контролери, довільні плагіни аудіоефектів і синтезаторів, оптимізоване читання/запис аудіофайлів безпосередньо на диск, тісно інтегроване розтягування часу тощо. Незважаючи на це, запропонована система зможе підтримувати широкий спектр досить складних ігор та інтерактивних програм, включаючи музичні додатки. Це стане хорошим доповненням до розширених графічних можливостей, які надає WebGL. API розроблено таким чином, щоб у майбутньому можна було додати додаткові розширені функції.

**BiquadFilterNode**

BiquadFilterNode є процесором AudioNode, що реалізує дуже поширені фільтри нижчого порядку. Ці фільтри є основними елементами для створення базових регуляторів тембру (баси, середні та високі частоти), графічних еквалайзерів і більш складних фільтрів. Об’єднуючи кілька фільтрів BiquadFilterNode, можна створити складніші фільтри. Параметри фільтра, такі як частота, можуть змінюватися з часом, дозволяючи створювати розгортки фільтра. Кожен BiquadFilterNode можна налаштувати на один із кількох загальних типів фільтрів, як зазначено в IDL нижче. Тип фільтра за замовчуванням – низькочастотний (НЧ) фільтр.

Низькочастотний фільтр пропускає частоти нижче граничної частоти та послаблює частоти вище граничної, реалізуючи стандартний резонансний фільтр низьких частот другого порядку зі спадом 12 дБ/октаву.

Високочастотний фільтр, навпаки, пропускає частоти вище граничної частоти та послаблює частоти нижче граничної. Він також є стандартним резонансним фільтром високих частот другого порядку зі спадом 12 дБ/октаву.

Смуговий фільтр пропускає певний діапазон частот і послаблює частоти нижче та вище цього діапазону, реалізуючи смуговий фільтр другого порядку.

Фільтр Lowshelf пропускає всі частоти, але підсилює (або ослаблює) нижні частоти, реалізуючи фільтр низького рівня другого порядку.

Фільтр Highshelf є протилежністю Lowshelf-фільтру: він пропускає всі частоти, але підсилює високі частоти, реалізуючи фільтр високої полиці другого порядку.

Піковий фільтр пропускає всі частоти, але підсилює (або ослаблює) певний діапазон частот.

Режекторний фільтр, також відомий як смуговідсічний або смуговідкидний фільтр, навпаки, пропускає всі частоти, крім певного діапазону частот.

1. **Аспекти імплементації**

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

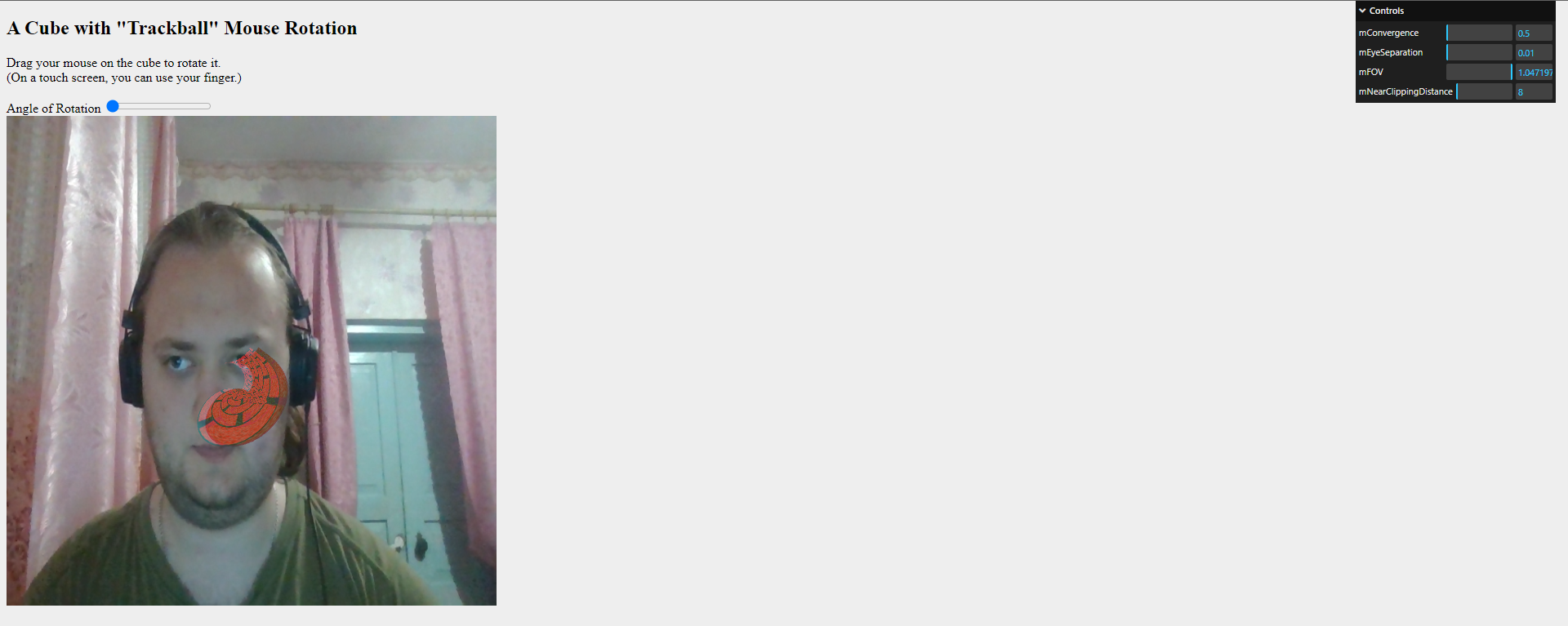


Рисунок 3.1 - скріншот 2 лаби

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 “music” 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 фільтру 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.

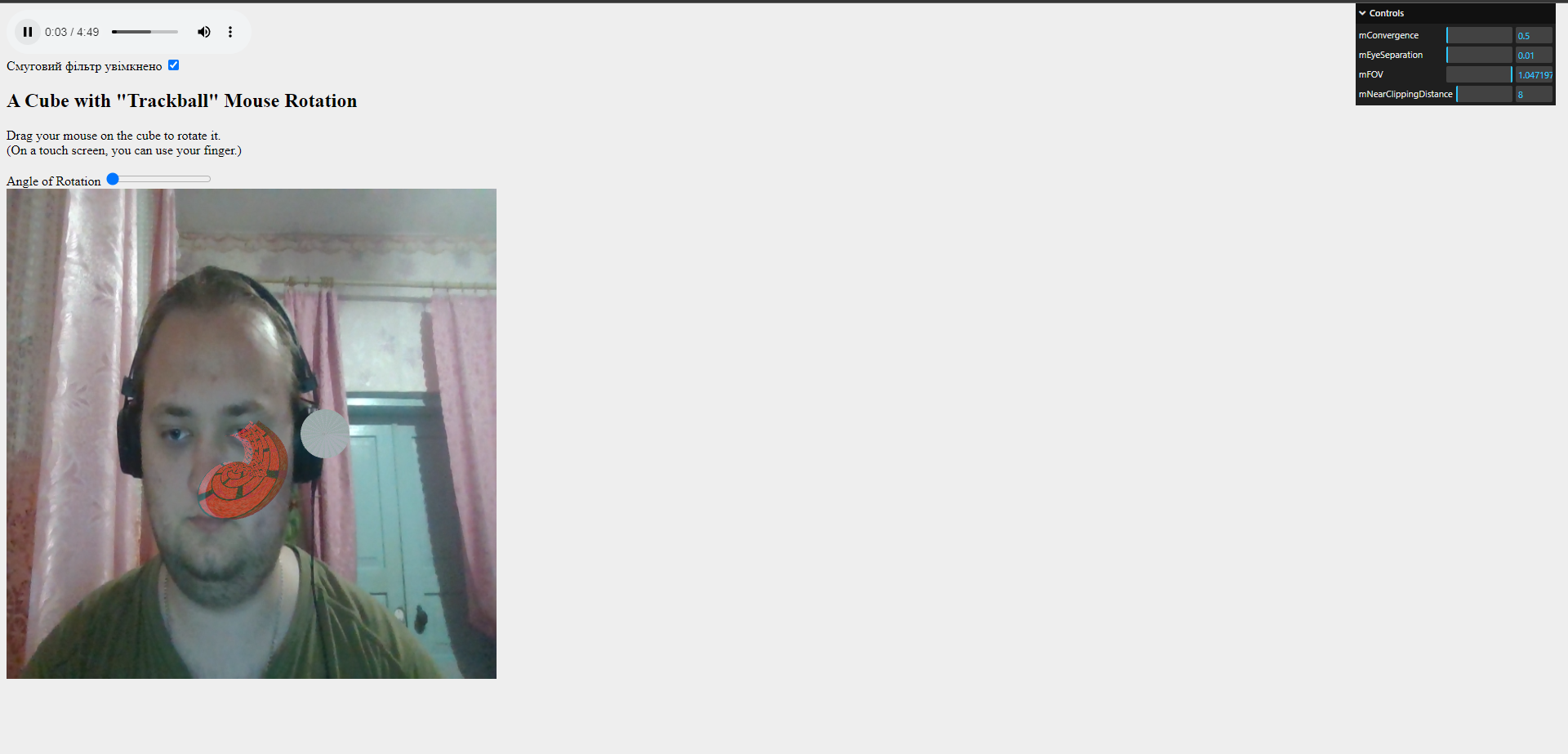


Рисунок 3.2 - скріншот РГР

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)

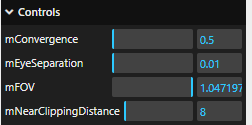


Рисунок 4.1 - скріншот слайдерів

Згідно до встановлених параметрів стерео зображення фігури виглядає дещо інакше

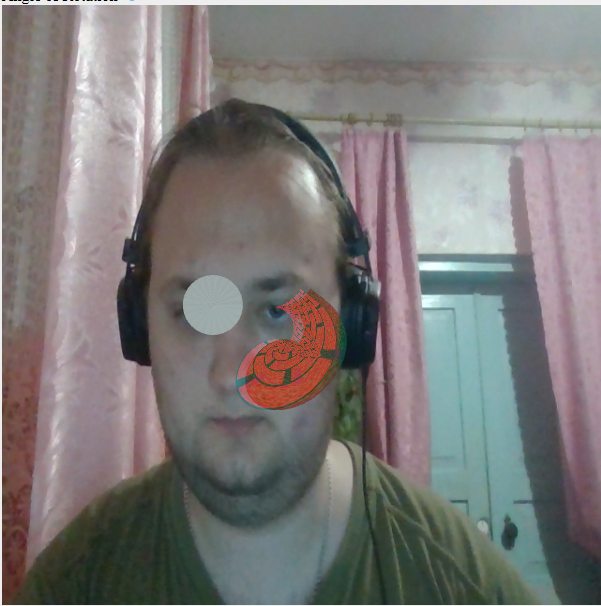


Рисунок 4.2 - скріншот фігури з початковими налаштуваннями



Рисунок 4.3 - скріншот фігури зі зміненими налаштуваннями

Вище згаданий HTML-елемент audio дозволяє керувати аудіо-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 filterOnOff;

let audioElement;

let sourcePointer

let smuhovyiFilter;

let stereoHelper;

let audioContext;

function setupListenersToElements() {

filterOnOff = document.getElementById('filterOnOff');

audioElement = document.getElementById('audioElement');

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

if (!audioContext) {

audioContext = new AudioContext();

sourcePointer = audioContext.createMediaElementSource(audioElement);

stereoHelper = audioContext.createPanner();

smuhovyiFilter = audioContext.createBiquadFilter();

sourcePointer.connect(stereoHelper);

stereoHelper.connect(smuhovyiFilter);

smuhovyiFilter.connect(audioContext.destination);

smuhovyiFilter.type = 'bandpass';

smuhovyiFilter.frequency.value = 1000;

smuhovyiFilter.Q.value = 1;

audioContext.resume();

}

})

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

console.log('pause');

audioContext.resume();

})

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

if (filterOnOff.checked) {

stereoHelper.disconnect();

stereoHelper.connect(smuhovyiFilter);

smuhovyiFilter.connect(audioContext.destination);

} else {

stereoHelper.disconnect();

stereoHelper.connect(audioContext.destination);

}

});

audioElement.play();

}