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 11

Виконав студент групи ТР-31мп  
Кравчук Дмитро Володимирович

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 обмеженим і потребувало використання плагінів таких як Flash або QuickTime для його відтворення. Впровадження елемента аудіо в HTML5 стало важливим проривом, дозволивши базове потокове відтворення без необхідності в додаткових плагінах. Проте ця можливість не підходить для більш вимогливих аудіо-додатків. Для розробк складних веб-ігор або інтерактивних програм потрібне досконаліше рішення. Ця специфікація має на меті інтегрувати можливості сучасних ігрових звукових двигунів, а також виконувати мікшування, обробку та фільтрацію, як це роблять сучасні настільні програми для створення звуку.

API-інтерфейси були створені з урахуванням різноманітних сценаріїв використання [webaudio-usecases]. Ідеально, система повинна підтримувати будь-який реалістичний сценарій, який може бути ефективно реалізований за допомогою оптимізованого C++ механізму, керованого через скрипти у браузері. Однак сучасне аудіо-програмне забезпечення часто має настільки розширені можливості, що деякі з них важко або неможливо відтворити за допомогою цієї системи. Наприклад, Apple Logic Audio підтримує зовнішні MIDI-контролери, різноманітні плагіни для аудіоефектів і синтезаторів, оптимізоване читання/запис аудіо на диск, інтегроване розтягнення часу і багато іншого. Проте, запропонований API буде здатен обслуговувати широкий спектр досить складних ігор і інтерактивних застосунків, включаючи музичні програми. Це стане відмінним доповненням до потужних графічних можливостей WebGL. API спроектовано з можливістю подальшого розширення функціональності.

**BiquadFilterNode** — це процесор AudioNode, який виконує обробку звуку за допомогою дуже поширених фільтрів нижчого порядку. Такі фільтри є основою для простих регуляторів тембру (баси, середні та високі частоти), графічних еквалайзерів і складніших фільтрів. Об'єднуючи кілька BiquadFilterNode, можна створювати більш складні фільтри. Параметри фільтрів, такі як частота, можна змінювати з часом, що дозволяє динамічне налаштування фільтра. Кожен BiquadFilterNode можна налаштувати на один з кількох типів фільтрів, як зазначено в IDL нижче. За замовчуванням використовується фільтр низьких частот (НЧ).

Фільтр низьких частот пропускає частоти нижче певного порогу, послаблюючи вищі частоти, і є стандартним резонансним фільтром другого порядку зі спадом 12 дБ/октаву.

Фільтр високих частот робить протилежне: пропускає високі частоти, а низькі послаблює, також виконуючи функцію резонансного фільтра другого порядку зі спадом 12 дБ/октаву.

Смуговий фільтр пропускає тільки частоти в певному діапазоні, послаблюючи частоти поза цим діапазоном, діючи як фільтр другого порядку.

Фільтр Lowshelf пропускає всі частоти, але підсилює або послаблює низькі частоти, працюючи як фільтр другого порядку.

Фільтр Highshelf, навпаки, підсилює або послаблює високі частоти, пропускаючи всі частоти і виконуючи роль фільтру другого порядку.

Піковий фільтр пропускає всі частоти, але додає підсилення або послаблення в певному діапазоні частот.

Режекторний фільтр, також відомий як смуговий або смуговідсічний, пропускає всі частоти, крім певного діапазону частот, зменшуючи їх інтенсивність.

1. **Аспекти імплементації**

У результаті виконання 3 та 4 лабораторних робіт було створено фізичний інтерфейс на основі роботи датчиків смартфона. Цей інтерфейс дозволяє користувачу обертати фігуру відповідно до показів сенсорів смартфона, враховуючи його положення в просторі.

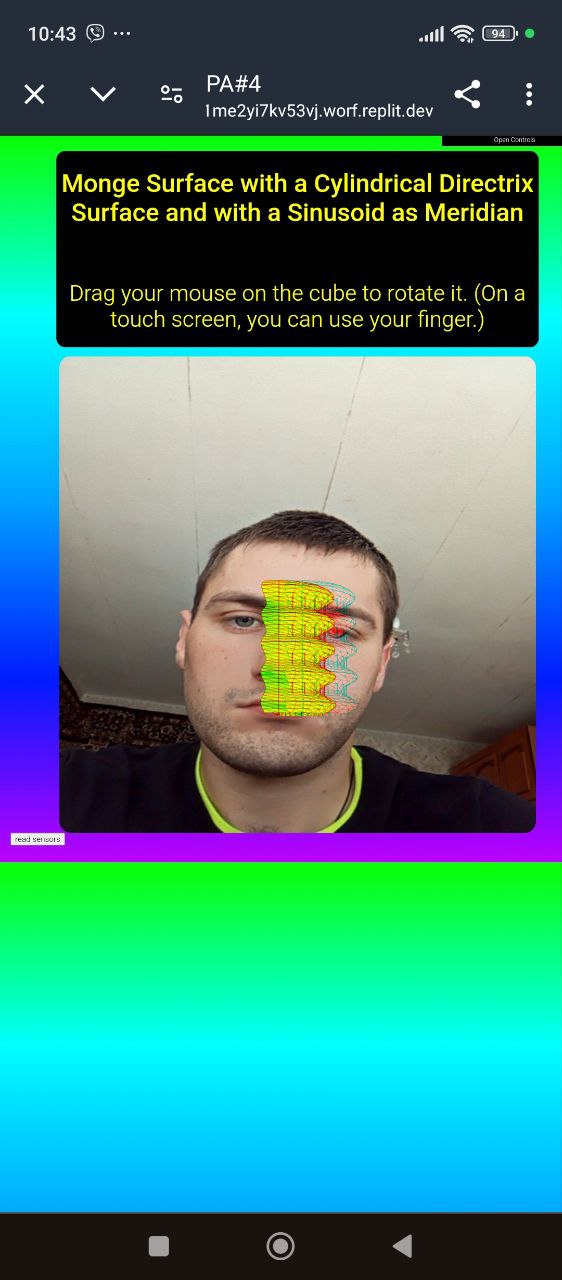


Рисунок 3.1 – Інтерфейс лабораторної роботи №4

В майбутньому цей функціонал дозволить керувати положенням джерела просторового аудіо всередині програми. Першим кроком для реалізації просторового аудіо є створення HTML-елемента <audio>, який в моєму випадку містить аудіофайл у форматі mp3 за шляхом “./song/song.mp3”. Цей елемент керування дає змогу зупиняти та відновлювати відтворення аудіофайлу. Далі, за допомогою JavaScript і WebAudio API, було створено аудіоконтекст (AudioContext). Для цього контексту було створено та підключено три основних об'єкти:

* джерело звуку (MediaElementSource)
* об'єкт для обробки просторового аудіо (Panner)
* звуковий фільтр (BiquadFilter)

Згідно з варіантом було обрано шелфовий фільтр низьких частот.

Наступним кроком було налаштування параметрів цього фільтра. BiquadFilter має три параметри для налаштування:

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

Особливості низькочастотного шелфового фільтра полягають у наступному:

* frequency визначає верхню межу частот, на які буде застосовуватися підсилення або послаблення.
* Q не використовується для цього типу фільтра.
* gain вказує рівень підсилення або послаблення у дБ. Від'ємне значення призводить до послаблення частот.

Джерело звуку, яке обробляється об'єктом класу Panner, представлено у WebGL контексті у вигляді сфери. Ця сфера може переміщуватися, змінюючи параметри об'єкта Panner, щоб створювати ефект переміщення джерела звуку при прослуховуванні стерео аудіо. Таким чином, звук змінюється відповідно до позиції сфери в координатній системі.

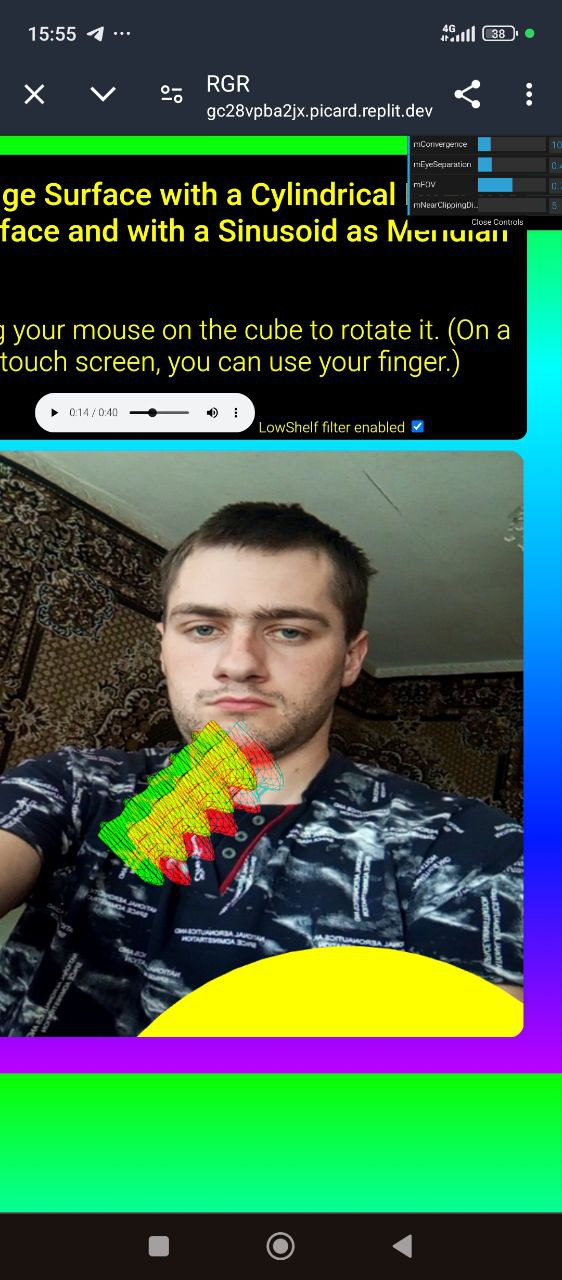


Рисунок 3.2 – Вигляд РГР

1. **Інструкція користувача**

Для регулювання стерео зображення використовуються чотири слайдери, за допомогою яких можна налаштувати такі параметри:

* Збіжність (Convergence)
* Відстань між очима (Eye separation)
* Поле зору (Field of view)
* Відстань до ближньої площини відсічення (Near clipping distance), що визначає, яка частина геометрії буде видимою на екрані.

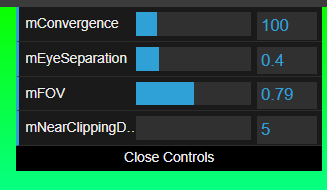


Рисунок 4.1 – Слайдери налаштування стерео зображення

Після зміни цих налаштувань фігура має дещо інший вигляд:

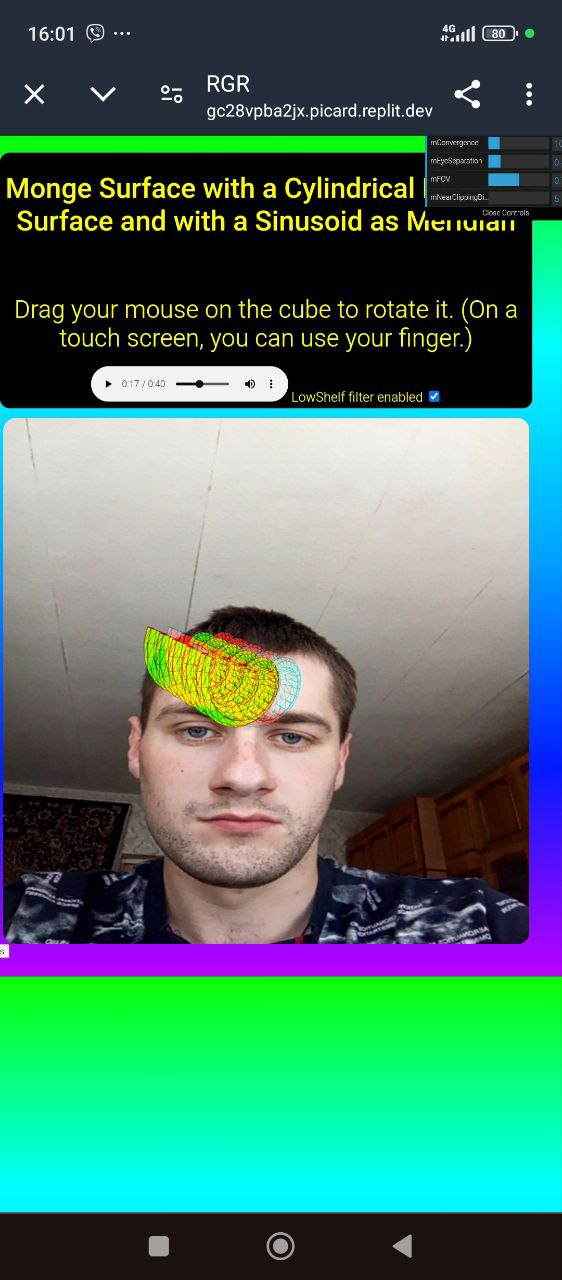


Рисунок 4.2 – Початкові налаштування фігури

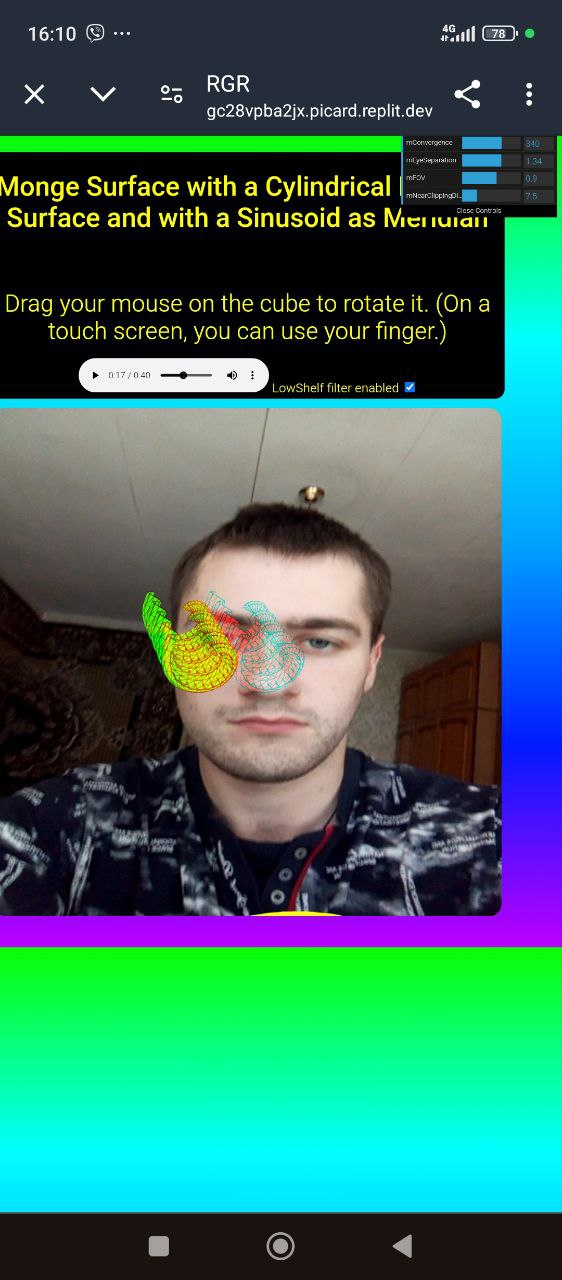


Рисунок 4.3 – Змінені налаштування фігури

Раніше згаданий HTML-елемент audio надає можливість керувати аудіо-доріжкою, включаючи зупинку та відновлення відтворення, перемотування до бажаного моменту та регулювання гучності.

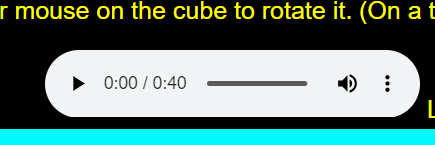


Рисунок 4.4 – Елемент керування аудіо відтворення

"Чекбокс" надає можливість керувати активністю звукового фільтру: вмикаючи та вимикаючи його. Залежно від стану фільтру можна відчути зміну у звучанні.



Рисунок 4.5 – Елемент управління станом звукового фільтру

1. **Код програми**

function setupAudio() {

audio = document.getElementById('song');

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 = 'lowshelf';

biquadFilter.Q.value = 0.8;

biquadFilter.frequency.value = 808;

biquadFilter.gain.value = 8;

context.resume();

}

})

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

console.log('pause');

context.resume();

})

}

function initAudio() {

setupAudio();

let radioButton = document.getElementById('lowshelf');

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

if (radioButton.checked) {

panner.disconnect();

panner.connect(biquadFilter);

biquadFilter.connect(context.destination);

} else {

panner.disconnect();

panner.connect(context.destination);

}

});

audio.play();

}