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 30

Виконав студент групи ТР-31мп  
Яковенко Олександр Ігорович

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**

WebAudio API дозволяє обробляти операції над аудіо за допомогою спеціального аудіоконтексту (аудіоконтексту) і спроектовано з використанням модульної маршрутизації (модульної маршрутизації). Базові операції виконуються за допомогою аудіо вузлів (аудіовузлів), які об'єднуються разом, формуючи аудіо-маршрутизатор \*таблицю (граф маршрутизації аудіо). Нескілько джерел - з різними видами поточних схем - підтримуються навіть зсередини простого контексту. Ця модульна концепція забезпечує гнучкість у створенні складних функцій для динамічних ефектів.

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**

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 виконання 3 ЛР

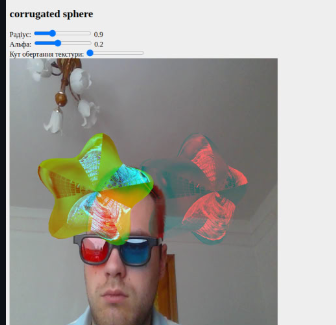


Рисунок 3.1 - Скріншот виконання 2 ЛР

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 “hymn” 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 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 фільтру високих частот в тому, що frequency - гранична частота, нижче якої частоти послаблюються

Q - контролює, наскільки піковим буде відгук на частоті зрізу. Велике значення робить відповідь більш піковою.

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.

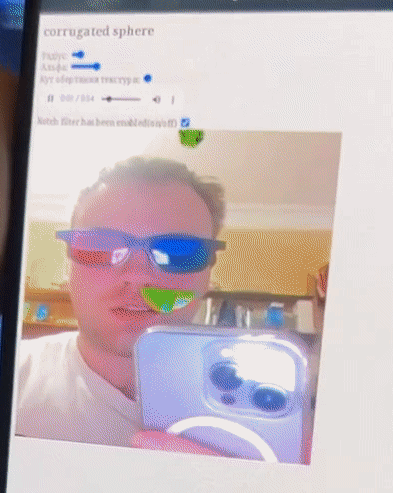


Рисунок 3.2 - Скріншот виконання CGW

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

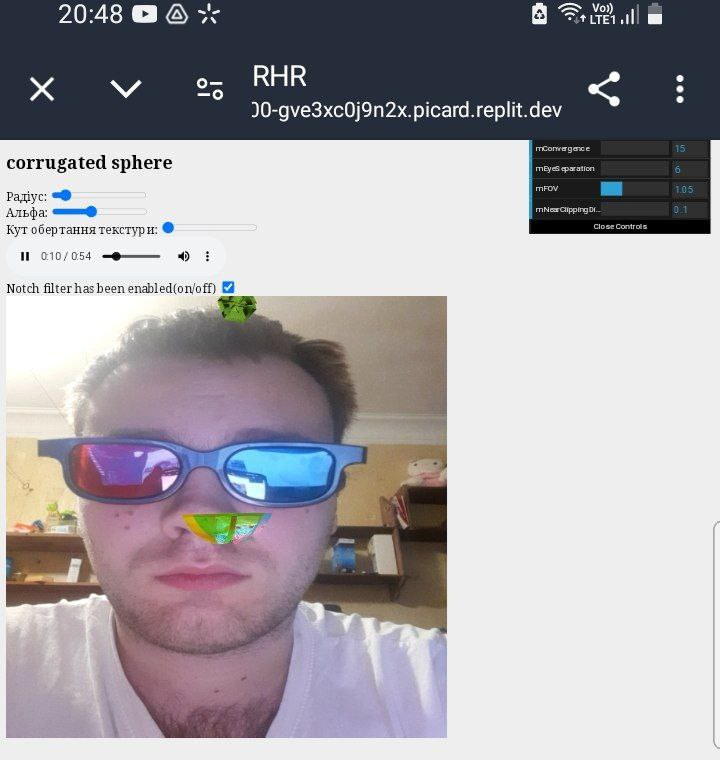


Рисунок 4.2 - скріншот фігури з початковими налаштуваннями

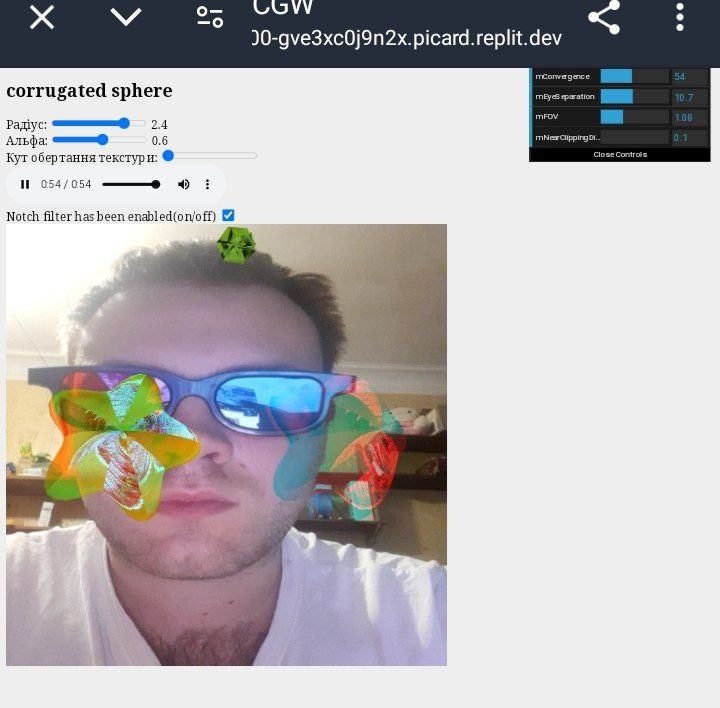


Рисунок 4.3 - скріншот фігури зі зміненими налаштуваннями

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



Рисунок 4.4 - UI-елемент плеєра

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



Рисунок 4.5 - чекбокс

1. **Код програми**

**function audioContextInitialization() {**

**audio = document.getElementById('idAudi');**

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

**if (!context) {**

**context = new AudioContext();**

**source = context.createMediaElementSource(audio);**

**panner = context.createPanner();**

**source.connect(panner);**

**highpass = context.createBiquadFilter();**

**panner.connect(highpass);**

**highpass.connect(context.destination);**

**highpass.type = 'highpass';**

**highpass.Q.value = 0.78;**

**highpass.frequency.value = 7000;**

**highpass.gain.value = 1; // 1 db as gain is not used in reject(notch) filter**

**context.resume();**

**}**

**})**

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

**console.log('pause');**

**context.resume();**

**})**

**let highState = document.getElementById('highState');**

**highState.addEventListener('change', function() {**

**if (highState.checked) {**

**panner.disconnect();**

**panner.connect(highpass);**

**highpass.connect(context.destination);**

**} else {**

**panner.disconnect();**

**panner.connect(context.destination);**

**}**

**});**

**audio.play();**

**}**

**function getVectorFromAngles() {**

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

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

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

**let vector = [0, 1, 0];**

**const rotZ = [**

**[Math.cos(gammaRad), -Math.sin(gammaRad), 0],**

**[Math.sin(gammaRad), Math.cos(gammaRad), 0],**

**[0, 0, 1]**

**];**

**vector = multiplyMatrixVector(rotZ, vector);**

**const rotY = [**

**[Math.cos(betaRad), 0, Math.sin(betaRad)],**

**[0, 1, 0],**

**[-Math.sin(betaRad), 0, Math.cos(betaRad)]**

**];**

**vector = multiplyMatrixVector(rotY, vector);**

**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;**

**}**