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 20

Виконав студент групи ТР-31мп  
Пархомчук Дмитро Юрійович

Київ 2024

1. **Завдання**

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

Забезпечити обертання джерела звуку відносно геометричного центру поверхні з використанням сенсорів смартфону для визначення його положення у просторі. На відміну від попередніх завдань, поверхня залишається нерухомою, а рухається саме джерело звуку. Відтворити улюблену пісню у форматі mp3/ogg з можливістю користувача контролювати місцезнаходження джерела звуку у просторі.

Візуалізувати джерело звуку за допомогою сфери.

Додати звуковий фільтр відповідно до варіанту (використовуючи інтерфейс BiquadFilterNode). Створити інтерфейс користувача, який дозволятиме вмикати та вимикати фільтр, а також налаштовувати його параметри згідно з уподобаннями.

Підготувати цифровий звіт, що містить всі необхідні частини для повного опису поставлених задач і виконаної роботи.

1. **Теоретичні відомості**

**WebAudio API**

Аудіо в Інтернеті до недавнього часу було досить примітивним, і його потрібно було відтворювати за допомогою плагінів, таких як Flash і QuickTime. Введення аудіоелемента в HTML5 є важливим кроком, оскільки він дозволяє відтворювати базове потокове аудіо. Проте, цей елемент не достатньо потужний для роботи зі складнішими аудіододатками. Для створення складних веб-ігор або інтерактивних програм потрібне інше рішення. Метою цієї специфікації є забезпечення можливостей сучасних ігрових звукових рушіїв, а також виконання завдань мікшування, обробки та фільтрації, які присутні в сучасних настільних програмах для створення звуку.

Інтерфейси API були розроблені з урахуванням широкого спектру випадків використання [webaudio-usecases]. Ідеальний API повинен підтримувати будь-який сценарій використання, який можна реалізувати за допомогою оптимізованого механізму на C++, керованого через скрипт і виконуваного в браузері. Однак сучасне аудіопрограмне забезпечення для комп’ютерів може мати дуже розширені можливості, деякі з яких важко або неможливо створити за допомогою цієї системи. Apple Logic Audio є прикладом такої програми, що підтримує зовнішні MIDI-контролери, довільні плагіни аудіоефектів і синтезаторів, високоефективне читання/запис аудіофайлів прямо на диск, тісно інтегроване розтягування часу тощо. Тим не менш, запропонована система буде здатна підтримувати широкий спектр досить складних ігор та інтерактивних програм, включаючи музичні. Це може стати хорошим доповненням до розширених графічних можливостей, які пропонує WebGL. API розроблено так, щоб у майбутньому можна було додати розширені можливості.

**BiquadFilterNode**

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

Ці фільтри є основою для базових регуляторів тембру (баси, середні та високі частоти), графічних еквалайзерів і складніших фільтрів. Комбінуючи кілька фільтрів BiquadFilterNode, можна створити більш складні фільтри. Параметри фільтра, такі як частота, можна змінювати з часом для динамічного налаштування тощо. Кожен BiquadFilterNode можна налаштувати як один із кількох типів фільтрів, як показано в IDL нижче. Тип фільтра за замовчуванням — «НЧ».

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

Фільтр високих частот, навпаки, пропускає частоти вище граничної, а частоти нижче граничної послаблює. 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. Він реалізує смуговий фільтр другого порядку.

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

Фільтр Highshelf, навпаки, пропускає всі частоти, але підсилює високі частоти. Він реалізує фільтр високого рівня другого порядку.

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

Режекторний фільтр (також відомий як смуговий стоп-фільтр) пропускає всі частоти, крім певного діапазону.

1. **Аспекти імплементації**

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

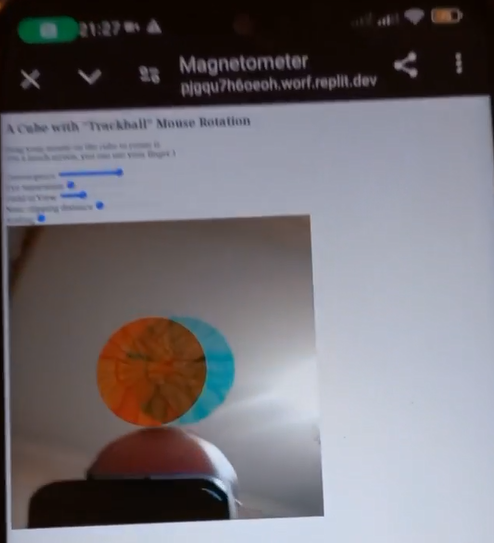


Рисунок 3.1 - скріншот виконання коду 4 лабараторної лабараторної

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

Першим кроком для реалізації просторового аудіо є створення HTML-елемента <audio>, який містить інформацію про джерело аудіодоріжки. У моєму випадку це “Frederic Chopin Nocturne In E Flat Major, Op. 9, No. 2” у форматі mp3. Цей елемент керування дозволить зупиняти та продовжувати відтворення аудіодоріжки. Після цього, за допомогою JavaScript, зокрема WebAudio API, було створено об'єкт аудіоконтексту (AudioContext), для якого було створено та підключено три основних об'єкти:

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

Згідно з варіантом було обрано піковий фільтр

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

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

Особливістю пікового фільтра є те, що в ньому frequency визначає центральну частоту, на якій застосовується підсилення, Q контролює ширину смуги підсилених частот; велике значення вказує на вузьку смугу, а gain встановлює підсилення в децибелах (дБ); якщо значення від'ємне, частоти послаблюються.

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

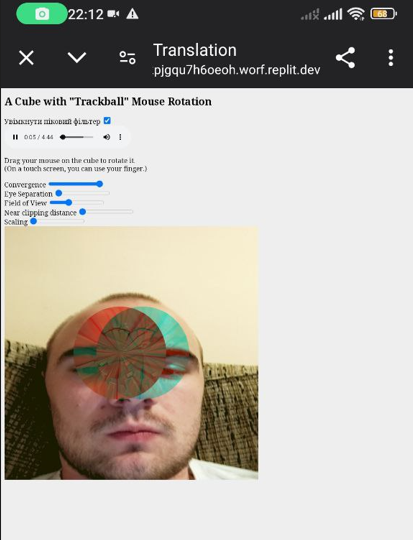


Рисунок 3.2 – скріншот виконання коду РГР

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)

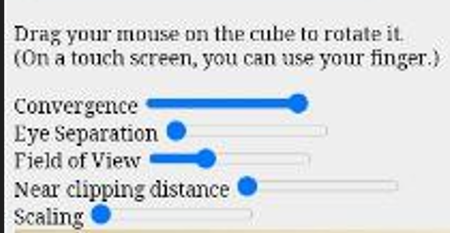


Рисунок 4.1 – Слайдери

Згідно до встановлених параметрів стерео зображення фігури виглядає дещо інакше

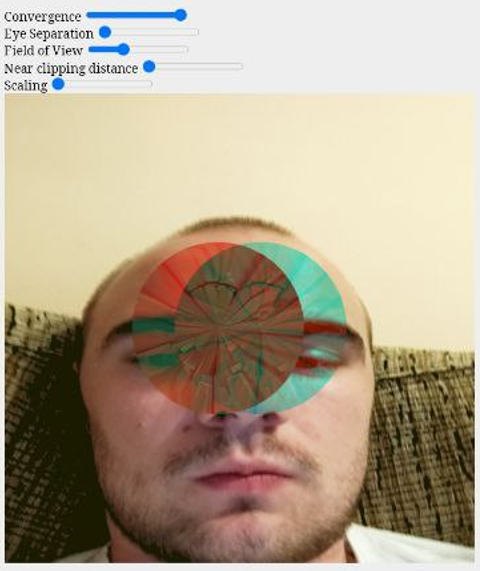


Рисунок 4.2 – Значення слайдерів за замовчуванням

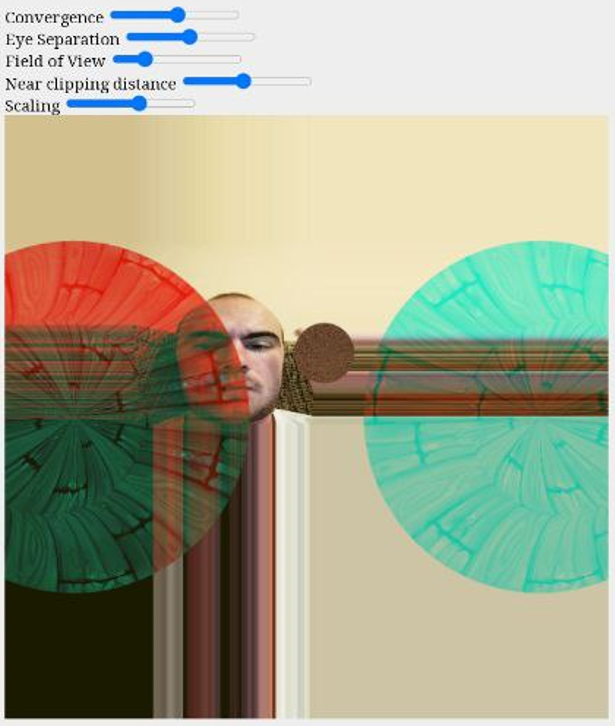


Рисунок 4.3 – Зміненні значення слайдерів

Вищезгаданий HTML-елемент audio дозволяє керувати аудіодоріжкою, зокрема зупиняти та продовжувати відтворення, перемотувати на потрібний час та керувати гучністю.



Рисунок 4.4 - Плеєр

"Чекбокс" дозволяє керувати станом звукового фільтру, дозволяючи користувачу вмикати та вимикати його. Залежно від стану фільтру можна помітити зміну звучання.



Рисунок 4.5 – “чекбокс” пікового фільтру

1. **Код програми**

let context, audio, source, biquadFilter, panner;

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 = 'peaking';

biquadFilter.gain.value = 9;

biquadFilter.Q.value = 0.99;

context.resume();

}

})

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

console.log('pause');

context.resume();

})

}

function initAudio() {

setupAudio();

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

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();

}

unction Grap\_Model(name) {

this.name = name;

this.iVertexBuffer = gl.createBuffer();

this.iNormalBuffer = gl.createBuffer();

this.iTextureBuffer = gl.createBuffer();

this.count = 0;

this.BufferData = function(vertices, normales, textures) {

gl.bindBuffer(gl.ARRAY\_BUFFER, this.iVertexBuffer);

gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(vertices), gl.STREAM\_DRAW);

gl.bindBuffer(gl.ARRAY\_BUFFER, this.iNormalBuffer);

gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(normales), gl.STREAM\_DRAW);

gl.bindBuffer(gl.ARRAY\_BUFFER, this.iTextureBuffer);

gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(textures), gl.STREAM\_DRAW);

this.count = vertices.length / 3;

};

this.Draw = function() {

gl.bindBuffer(gl.ARRAY\_BUFFER, this.iVertexBuffer);

gl.vertexAttribPointer(sh\_prog.iAttribVertex, 3, gl.FLOAT, false, 0, 0);

gl.enableVertexAttribArray(sh\_prog.iAttribVertex);

gl.bindBuffer(gl.ARRAY\_BUFFER, this.iNormalBuffer);

gl.vertexAttribPointer(sh\_prog.iAttribNormal, 3, gl.FLOAT, true, 0, 0);

gl.enableVertexAttribArray(sh\_prog.iAttribNormal);

gl.bindBuffer(gl.ARRAY\_BUFFER, this.iTextureBuffer);

gl.vertexAttribPointer(sh\_prog.iAttribTexture, 2, gl.FLOAT, true, 0, 0);

gl.enableVertexAttribArray(sh\_prog.iAttribTexture);

gl.drawArrays(gl.TRIANGLES, 0, this.count);

};

}