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-31мп  
Мороз Єлизавета Олександрівна

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 часу було досить примітивним і донедавна вимагало використання плагінів, таких як Flash і QuickTime. Впровадження аудіоелемента в HTML5 є значущим, оскільки дозволяє відтворювати базове потокове аудіо. Однак, воно недостатньо потужне для складніших аудіододатків. Для складних веб-ігор або інтерактивних програм потрібне інше рішення. Метою цієї специфікації є включення можливостей сучасних ігрових звукових движків, а також завдань мікшування, обробки та фільтрації, які є у сучасних настільних програмах для створення звуку.

Інтерфейси API розроблені з урахуванням широкого спектру випадків використання [webaudio-usecases]. Ідеально, API повинен підтримувати будь-який варіант використання, який можна розумно реалізувати за допомогою оптимізованого механізму C++, керованого через сценарій і запущеного в браузері. Однак сучасне аудіопрограмне забезпечення для настільних комп’ютерів може мати дуже розширені можливості, деякі з яких важко або неможливо реалізувати за допомогою цієї системи. Apple Logic Audio є однією з таких програм, яка підтримує зовнішні MIDI-контролери, довільні аудіоефекти і синтезатори, високооптимізоване читання/запис аудіофайлів на диск, тісно інтегроване розтягування часу тощо. Попри це, запропонована система здатна підтримувати широкий спектр досить складних ігор та інтерактивних програм, включаючи музичні. Вона може бути гарним доповненням до розширених графічних функцій, які пропонує WebGL. API розроблений таким чином, щоб у майбутньому можна було додати додаткові можливості.

**BiquadFilterNode**

BiquadFilterNode — це процесор AudioNode, що реалізує поширені фільтри нижчого порядку. Такі фільтри слугують будівельними блоками для базових регуляторів тембру (баси, середні та високі частоти), графічних еквалайзерів та складніших фільтрів. Кілька фільтрів BiquadFilterNode можна комбінувати для створення більш складних фільтрів. Параметри фільтра, наприклад частоту, можна змінювати з часом для різних ефектів. 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 неї. Він реалізує стандартний резонансний фільтр низьких частот другого порядку зі спадом 12 дБ/октаву.

Фільтр високих частот є протилежністю фільтру низьких частот. Він пропускає частоти вище граничної частоти та послаблює частоти нижче неї. 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 додає підсилення (або ослаблення) до нижніх частот. Він реалізує фільтр низького рівня другого порядку.

Фільтр Highshelf є протилежністю фільтру Lowshelf, пропускаючи всі частоти, але додаючи підсилення до високих частот. Він реалізує фільтр високої полиці другого порядку.

Піковий фільтр пропускає всі частоти, але додає підсилення (або ослаблення) до певного діапазону частот.

Режекторний фільтр (також відомий як смуговий або смуговий фільтр) є протилежністю смуговому фільтру. Він пропускає всі частоти, крім певного діапазону 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.

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 “ Disturbed – Legion of Monsters” 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 низьких частот

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

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

Особливість фільтру низьких частот в тому, що

frequency - встановлює частоту зрізу

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.

A screenshot of a cellphone

Description automatically generated

Рисунок 3.1 – РГР з мобільного телефону

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)

A screenshot of a computer

Description automatically generated

Рисунок 4.1 - скрішот слайдерів

Згідно до встановлених параметрів стерео зображення фігури виглядає дещо інакше

A screenshot of a phone

Description automatically generated

Рисунок 4.2 - фігура з початковими налаштуваннями

A screenshot of a video

Description automatically generated

Рисунок 4.3 - фігура зі зміненими налаштуваннями

За допомогою HTML-елемента audio можна керувати аудіо-відтворенням:

Пуск/пауза: Розпочинати та зупиняти відтворення аудіо.

Перемотування: Швидко переміщатися по аудіозапису до потрібного моменту.

Регулювання гучності: Збільшувати або зменшувати гучність звуку.

Фільтр:

Вмикання/вимикання: За допомогою "чекбокса" можна активувати або деактивувати звуковий фільтр.

Вплив на звучання: Залежно від стану фільтра, звучання аудіо може змінюватися.

A screenshot of a phone

Description automatically generated

Рисунок 4.4 - скріншот з UI-елементом плеєра, чекбоксу

1. **Код програми**

let audio = null, ctx, src, fltr, panner;

function setupAudio() {

audio = document.getElementById('msc');

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

if (!ctx) {

ctx = new AudioContext();

src = ctx.createMediaElementSource(audio);

panner = ctx.createPanner();

fltr = ctx.createBiquadFilter();

src.connect(panner);

panner.connect(fltr);

fltr.connect(ctx.destination);

fltr.type = 'lowpass';

fltr.Q.value = 0.5;

fltr.frequency.value = 1234;

ctx.resume();

}

})

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

console.log('pause');

ctx.resume();

})

}

function initAudio() {

setupAudio();

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

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

if (radioButton.checked) {

panner.disconnect();

panner.connect(fltr);

fltr.connect(ctx.destination);

} else {

panner.disconnect();

panner.connect(ctx.destination);

}

});

audio.play();

}