Міністерство освіти і науки України

Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота

з курсу “ Методи синтезу віртуальної реальності”

Варіант 16

**Виконав:**

студент 5-го курсу

групи ТР-32мп

Петричук Владислав Володимирович

**Перевірив:**

Демчишин А. А.

Київ – 2024

**Завдання:**

* повторно використовувати код із практичного завдання №2;
* ті, хто має сертифікат курсу навчання дронів FPV: реалізуйте обертання джерела звуку навколо геометричного центру ділянки поверхні по колу протягом певного часу (цього разу поверхня залишається нерухомою, а джерело звуку рухається).  Відтворюйте улюблену пісню у форматі mp3/ogg, маючи просторове розташування джерела звуку, кероване користувачем;
* візуалізувати положення джерела звуку за допомогою сфери;
* додайте звуковий фільтр (використовуйте інтерфейс BiquadFilterNode) для кожного варіанту . Додайте елемент прапорця, який вмикає або вимикає фільтр. Встановіть параметри фільтра на свій смак.

*Варіант 16 – фільтр високих частот*

**Теоретична частина:**

Основні нові інструменти програмного забезпечення, використані у даній роботі:

**WebAudio API**

WebAudio API — це загальнодоступний інтерфейс програмування додатків, який дозволяє розробникам і студентам створювати, обробляти та контролювати звук у веб-браузерах. Він надає низькорівневий доступ до аудіо, що дає змогу створювати складні звукові ефекти та проводити аналіз звуку. Це робить його ідеальним для інтерактивних додатків, музичних інструментів, ігор та інших мультимедійних проектів. WebAudio API дає можливість обробляти звукові дані в режимі реального часу, застосовуючи різноманітні ефекти, фільтри та мікшування звуків. Цей API підтримується більшістю сучасних браузерів, що забезпечує його крос-платформенну сумісність.

Основна концепція WebAudio API базується на використанні аудіоконтексту (AudioContext), який є центральним елементом для всіх звукових операцій. Аудіоконтекст дозволяє створювати джерела звуку, застосовувати аудіоефекти, підключати різні аудіовузли та керувати ними. Серед ключових компонентів WebAudio API можна виділити генератори звуку (OscillatorNode), вузли для відтворення аудіофайлів (AudioBufferSourceNode), аналізатори (AnalyserNode), а також фільтри, такі як BiquadFilterNode.

**BiquadFilterNode**

BiquadFilterNode — це один із типів аудіовузлів у WebAudio API, який використовується для обробки звуку за допомогою бі-квадратних фільтрів. Цей вузол дозволяє застосовувати до аудіосигналу різні типи фільтрів, такі як низькочастотний (low-pass), високочастотний (high-pass), смуговий (band-pass), відсічний (notch) та інші. Бі-квадратні фільтри отримали свою назву завдяки математичній структурі, яка включає квадратичні рівняння як у чисельнику, так і в знаменнику передавальної функції.

BiquadFilterNode є дуже корисним для обробки аудіо в режимі реального часу, дозволяючи налаштовувати такі параметри, як частота зрізу (frequency), добротність (Q) та посилення (gain). Це дає розробникам можливість гнучко керувати частотним спектром аудіосигналу, виділяючи або приглушуючи певні частоти відповідно до вимог застосування. Фільтри BiquadFilterNode можуть використовуватися для створення ефектів, таких як еквалізація, фільтрація шумів, модифікація тону та інших аудіо трансформацій.

У BiquadFilterNode реалізовано наступні фільтри:

* **Низькочастотний фільтр (Low-pass filter)** дозволяє пропускати частоти нижче встановленої порогової частоти (частоти зрізу), одночасно відсікаючи високі частоти. Це корисно для усунення високочастотного шуму або для створення ефекту згладжування звуку.
* **Високочастотний фільтр (High-pass filter)** пропускає частоти, які перевищують певну частоту зрізу, відфільтровуючи при цьому низькі частоти. Використовується для видалення низькочастотного шуму або підкреслення високочастотних компонентів звуку.
* **Смуговий фільтр (Band-pass filter)** дозволяє пропускати частоти в межах певного діапазону, відсікаючи ті, що знаходяться за його межами. Це корисно для виділення конкретного діапазону частот з аудіосигналу.
* **Відсічний фільтр (Notch filter або Band-stop filter)** робить протилежне до смугового фільтра: він відсікає частоти в межах певного діапазону, пропускаючи решту спектра. Використовується для усунення специфічних частотних перешкод, таких як шум від електромережі.
* **Вузькосмуговий фільтр (All-pass filter)** не змінює амплітуду частотного спектра, але змінює фазу звукового сигналу. Він може бути корисним для створення певних звукових ефектів, таких як фазова модуляція.
* **Підсилюючий фільтр низьких частот (Low-shelf filter)** збільшує або зменшує амплітуду звуку нижче встановленої частоти зрізу, не впливаючи на високі частоти. Це корисно для корекції або підсилення низьких частот в аудіосигналі.
* **Підсилюючий фільтр високих частот (High-shelf filter)** збільшує або зменшує амплітуду звуку вище певної частоти зрізу, не впливаючи на низькі частоти. Використовується для корекції або підсилення високих частот.
* **Піковий еквалайзер (Peaking filter)** збільшує або зменшує амплітуду звуку в межах певного діапазону частот, залишаючи решту спектра незмінною. Він особливо корисний для точного налаштування конкретних частотних областей.

Кожен із цих фільтрів може бути налаштований за допомогою параметрів частоти зрізу (frequency), добротності (Q) та посилення (gain), що дозволяє гнучко керувати частотним спектром звукового сигналу для досягнення бажаного ефекту.

Загалом, WebAudio API та BiquadFilterNode є необхідними інструментами для розробників, які дозволяють створювати складні та інтуїтивно зрозумілі аудіорішення у веб-додатках, забезпечуючи багатий користувацький досвід і високу якість звуку.

**Опис деталей впровадження**

Основа для виконання роботи була взята з програмного коду другої лабораторної роботи. Для реалізації завдання була створена сфера, використовуючи код з попередніх робіт, а також функцію створення сфери з практичного завдання курсу попереднього семестру, що передував розробці програми поточного курсу.

Згідно із завданням, необхідно було реалізувати обертання джерела звуку навколо центру фігури протягом часу. Для позначення джерела звуку було вирішено використати моделювання сфери. Через простоту реалізації шейдеру протягом всього навчального курсу не було необхідності змінювати його в цій роботі, тому нові зміни в цьому аспекті не були внесені. Замість цього вдалося вдало використати вже напрацьований матеріал.

Для реалізації просторового звуку було використано WebAudio HTML5 API. Музикальну мелодію підібрано на власний розсуд у відповідності до вимог завдання. Щоб уникнути проблем з доступом до файлу з музикою на різних пристроях та браузерах, її було завантажено на Github, і для програвання при запуску програми реалізовано завантаження через посилання.

Аудіофільтр, згідно з варіантом завдання, був обраний для низьких частот і відповідно реалізований. Джерело звуку, оброблене об'єктом класу Panner, представлено в контексті WebGL у вигляді сфери, яка може переміщуватися, змінюючи параметри об'єкта Panner. Це забезпечує ефект переміщення звукового джерела при прослуховуванні стерео аудіо, що відповідає положенню сфери в системі координат.

**Інструкції користувача**

З попередніх робіт для налаштування відображення використовуються 4 параметри:

1. Point of view - поле обзору
2. Convergence – збіжність
3. Near-clipping distance - відстань ближньої площини, що відсікає моделювання
4. Separation - відстань між очима

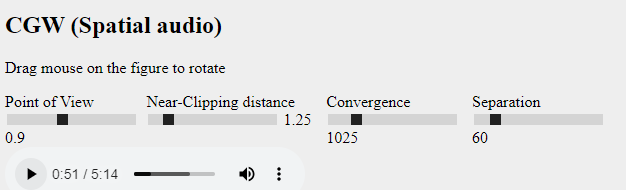


Рис. 1 – Налаштування

Audio-елемент, який дозволяє відтворювати аудіо окремо від кнопки ініціалізації та регулювати timecode аудіо.

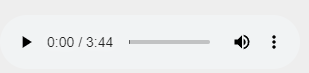


Рис. 2 – Аудіо доріжка

Кнопка запуску фільтру високих частот



Рис. 3 – Кнопка запуску фільтру

Флажок для вмикання та вимикання застосування фільтру високих частот.



Рис. 4 – Флажок фільтру

Загальний вигляд роботи:

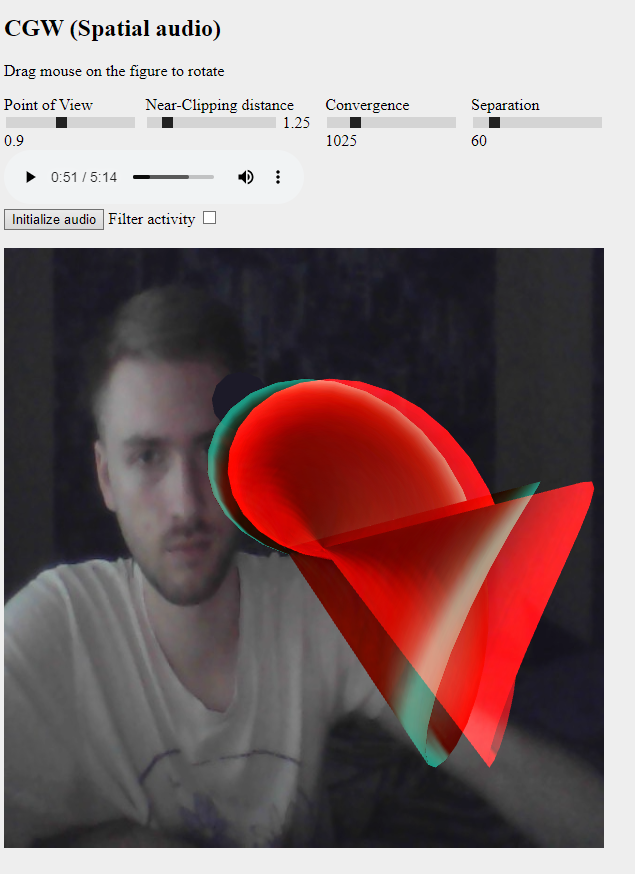


Рис. 5 – Головна сторінка

**Код програми**

Функція initAudio() з фільтром високих частот:

function initAudio() {

    filterNow = document.getElementById('filterState');

    audio = document.getElementById('audioContext');

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

        if (!context) {

            context = new (window.AudioContext || window.webkitAudioContext)();

            audiosource = context.createMediaElementSource(audio);

            highpassFilter = context.createBiquadFilter();

            panner = context.createPanner();

            audiosource.connect(panner);

            panner.connect(highpassFilter);

            highpassFilter.connect(context.destination);

            highpassFilter.type = 'highpass';

            highpassFilter.Q.value = 1;

            highpassFilter.frequency.value = 350;

            context.resume();

        }

    });

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

        context.suspend();

    });

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

        if (filterNow.checked) {

            panner.disconnect();

            panner.connect(highpassFilter);

            highpassFilter.connect(context.destination);

        } else {

            panner.disconnect();

            panner.connect(context.destination);

        }

    });

    audio.play();

}