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»
Варіант 5

Виконав: Грицюк А.О.
Студент групи ТР-21мп

Київ 2023

Зміст

Завдання

Тема роботи: Звук у просторі. Імлементувати звук у просторі за допомогою WebAudio HTML5 API

Вимоги:

- Перевикористати код з практичної роботи №2.
- Імплементувати обертання джерела звуку навколо геометричного центру поверхні за допомогою матеріального інтерфейсу. Програвати улюблену пісню у форматі mp3/ogg, змінюючи розташування джерела звуку відповідно до введення користувача.
- Візуалізувати джерело звуку у вигляді сфери.
- Додати звуковий фільтр за варіантом. Додати «галочку», яка вмикає ти вимикає фільтр. Задати параметри фільтру за смаком.

Теоретичні відомості

Web Audio API — це потужний інструмент, який дозволяє розробникам маніпулювати та синтезувати звук у веб-додатках. Він надає набір інтерфейсів і об’єктів, які дозволяють створювати, модифікувати та маршрутизувати аудіосигнали в реальному часі. Одним із ключових аспектів API веб-аудіо є його здатність обробляти аудіо та керувати ним за допомогою модульного підходу, який дозволяє створювати складні конвеєри обробки аудіо.

Серед багатьох об’єктів, доступних у API веб-аудіо, три широко використовувані – це AudioContext, MediaElementSourceNode, PannerNode і BiquadFilterNode. Давайте заглибимося в кожен із цих об’єктів і дослідимо їхні ролі та функції.

AudioContext:

AudioContext представляє граф обробки аудіо та діє як центральний центр для створення та підключення аудіо вузлів. Він служить точкою входу для доступу та керування аудіофункціями, які надає Web Audio API. Створюючи екземпляр AudioContext, розробники отримують доступ до широкого спектру методів і властивостей для керування відтворенням аудіо, маршрутизацією та ефектами.

У фрагменті коду “context = new AudioContext();” ініціалізується об’єкт AudioContext, який служить основою для конвеєра обробки звуку.

MediaElementSourceNode:

MediaElementSourceNode використовується для отримання аудіоданих із медіа-елементів HTML, таких як <audio> або <video>. Він являє собою джерело аудіо, яке можна підключити до інших аудіо вузлів для подальшої обробки або маршрутизації. Використовуючи MediaElementSourceNode, розробники можуть включати існуючі медіа-елементи в екосистему Web Audio API і застосовувати різні звукові ефекти або маніпуляції.

У коді “source = context.createMediaElementSource(audio);” створює MediaElementSourceNode, де змінна audio посилається на елемент HTML

<audio>. Це дозволяє обробляти аудіодані з указаного медіа-елемента за допомогою Web Audio API.

PannerNode:

PannerNode відповідає за просторове позиціонування та панорамування звуку. Він імітує тривимірне аудіо, регулюючи положення, орієнтацію та швидкість аудіоджерела у віртуальному 3D-просторі. Цей об’єкт дозволяє розробникам створювати ефект занурення в аудіо, коли звук виходить із певних напрямків, створюючи відчуття глибини та руху.

У коді “panner = context.createPanner();” створює PannerNode, який можна підключити до звукового графіка. PannerNode можна використовувати для керування положенням і рухом аудіоджерела, забезпечуючи динамічне розподілення звуку в просторі.

BiquadFilterNode:

BiquadFilterNode реалізує різні типи цифрових фільтрів, наприклад фільтри низьких частот, високочастотні, смугові та пікові фільтри. Це дозволяє розробникам формувати частотну характеристику аудіосигналу, змінюючи його тембр і застосовуючи такі ефекти, як вирівнювання або резонанс. BiquadFilterNode пропонує параметри для керування частотою зрізу, посиленням і якістю фільтра.

У коді “biquadFilter = context.createBiquadFilter();” створює BiquadFilterNode. Після підключення до аудіографа цей вузол можна використовувати для застосування ефектів фільтрації до аудіосигналу, покращуючи або змінюючи його спектральні характеристики.

Підсумовуючи, Web Audio API надає потужний набір об’єктів, які дозволяють розробникам маніпулювати та обробляти звук у веб-додатках. AudioContext діє як основний інтерфейс, тоді як такі об’єкти, як MediaElementSourceNode, PannerNode і BiquadFilterNode, пропонують спеціальні функції для вилучення аудіоданих, розміщення звуку у віртуальному просторі та застосування ефектів цифрової фільтрації. Використовуючи ці об’єкти та можливості Web Audio API, розробники можуть створювати захоплюючі та інтерактивні аудіо в Інтернеті.

Особливості виконання завдання

В ході другої лабораторної роботи було імплементовано можливість обертати поверню під назвою «Мінімальна поверхня Річмонда» у стерео вигляді за допомогою акселерометра: обертаючи телефон – відповідно оберталась і фігура. Демонстрацію застосування програми можна обачити на рисунку 3.1.

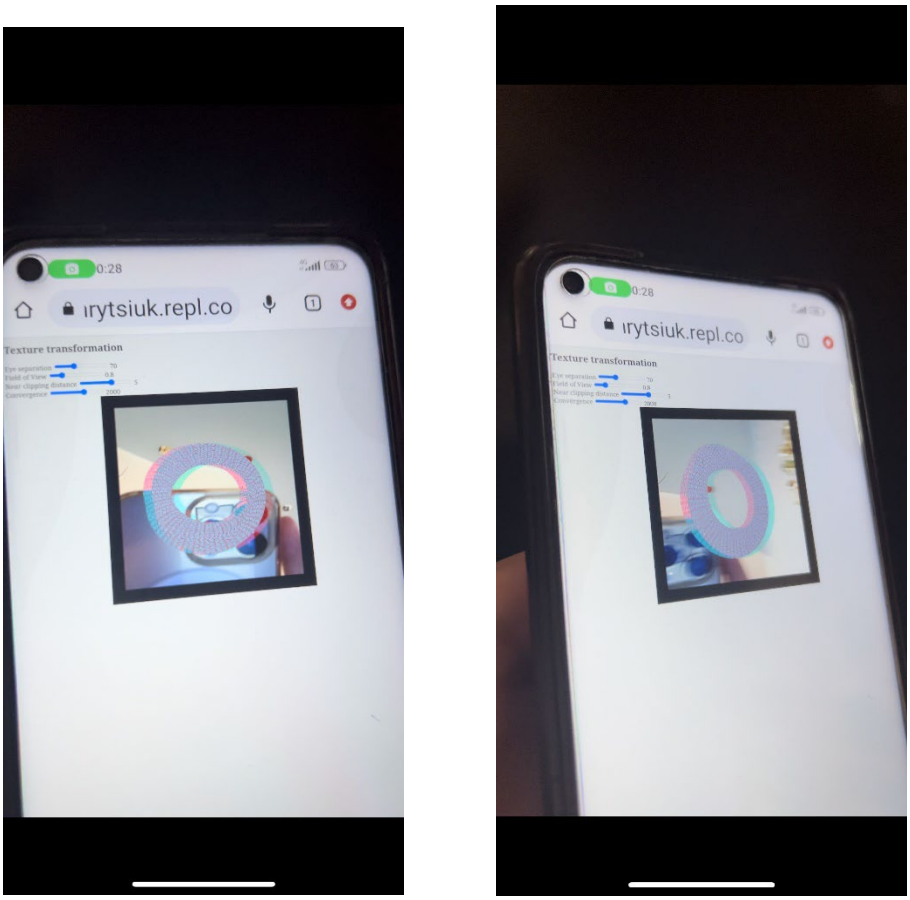

Рис. 3.1 Демонстрація застосування програми, отриманої в ході другої лабораторної

За допомогою Web Audio API, а саме документації представленої на сторінці https://webaudio.github.io/web-audio-api/ було імплементовано основну частину завдання розрахунково-графічної роботи.

В ході виконання лабораторної роботи необхідно було спочатку створити об’єкт аудіоконтексту, що дозволяє отримати доступ до Web Audio API.

Для виконання роботи було також обрано аудіо-файл формату mp3 і представлено його на веб-сторінці через HTML-елемент <audio>.

Наступним кроком було створити джерело аудіо передавши аудіо-елемент в конструктор.

Також необхідно було створити об’єкт panner в контексті, для подальшої маніпуляції звуком, зокрема позицією, що буде змінюватися по обертанню телефоном(джерело звуку буде знаходитися на умовній відстані 2 від центру в сторону відповідну повороту телефону в просторі).

Важливою частиною завдання було застосувати фільтр до вихідного звуку. За варіантом було імпелментовано шелфовий фільтр високих частот з параметрами вказанами в розділі 5.

Далі потрібно було поєднати об’єкту, передавші відповідні об’єкти іншим. Було додано eventListener, що відповдає за зупинку та продовження програвання аудіо-файлу.
Крім цього необхідно було створити поле для увімкнення та вимкнення фільтру, а також додати інший eventListener для перемикання фільтра по перемиканню вище вказаного поля.

Оновлення позиції звуку через переміщення об’єкту panner було імплементовані в основній функції під назвою draw.

Вказівки користувачу

Користувач може керувати переміщенням умовної сфери, що показує користувачу умовне місцезнаходження джерела звуку.

Відповідне переміщення можна побачити на рисунках 4.1 та 4.2

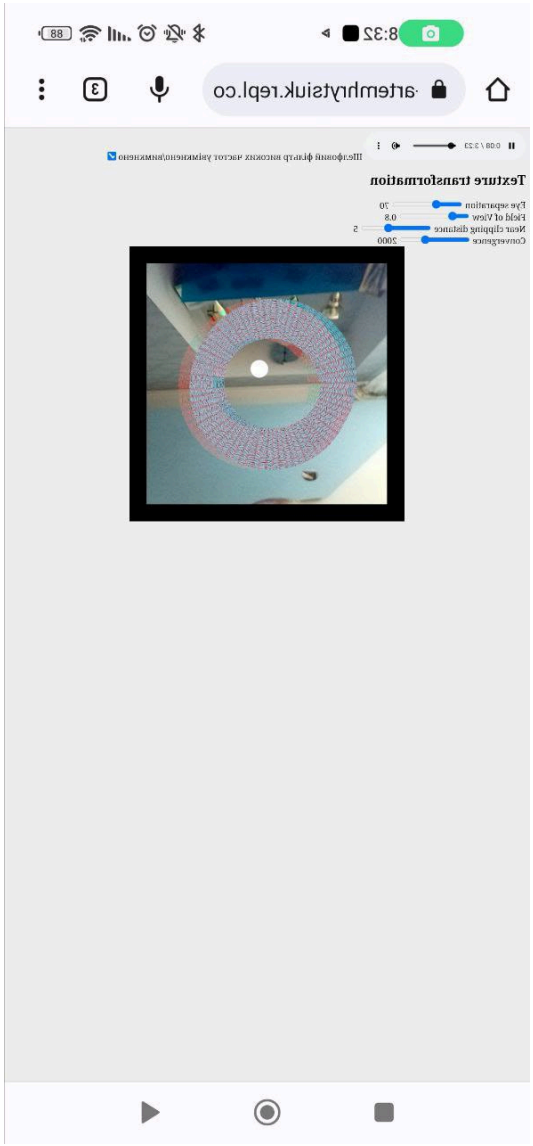

Рисунок 4.1 Матеріальний інферфейс повернуто догори

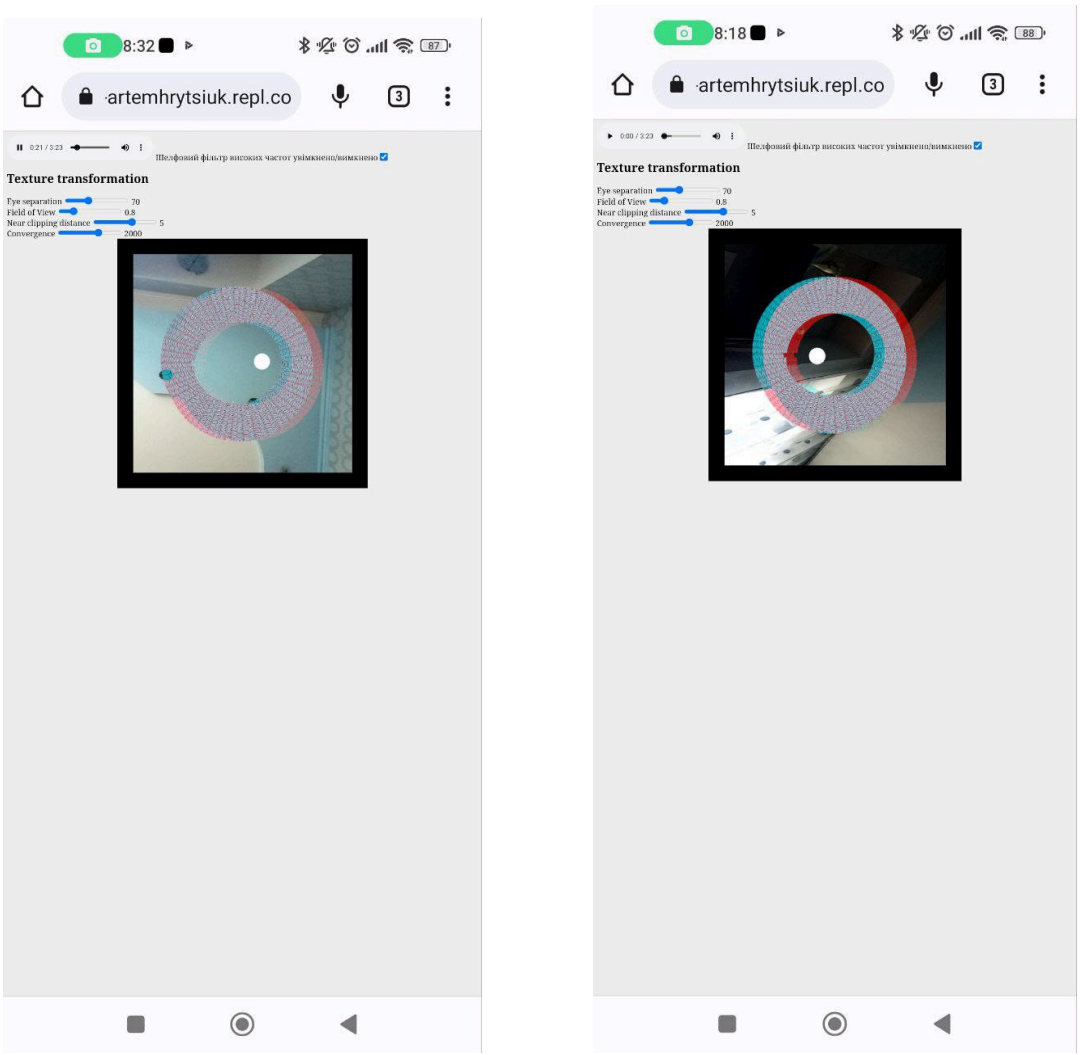

Рисунок 4.2 Матеріальний інтерфейс повернуто вбік

При обертанні телефону сфера переміщується навколо фігури. З переміщенням сфера створюється ефект переміщення джерела звуку, який найкраще відчувається в навушниках та аудіо стерео системах.
Окрім іншого на сторінці представлено елементи інтерфейсу для зміни параметрів стерео-зображення, а саме значення eye separation, field of view, near clipping distance та convergence. Дані слайдери можна побачити на рисунку 4.3

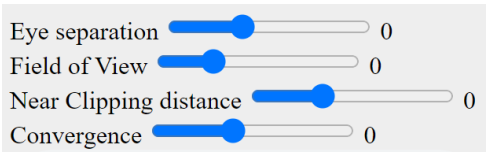

Рисунок 4.3 Слайдери для зміни параметрів стерео зображення

На сторінці можна побачити елементи управлінням аудіо-файлом: перемотка, пауза, продовження, керування гучністю, див. рисунок 4.4.

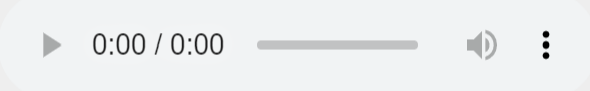

Рисунок 4.4 Елемент управління аудіо-файлом

Було також створено елемент «чекбокс» для увімкнення та вимкнення фільтру, див. рисунок 4.5.

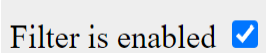

Рисунок 4.5 Чекбокс перемикання стану фільтру

Лістинг програми

Код функцій для налаштування та ініціалізації аудіо та акселерометра

```
const acl = new Accelerometer({ frequency: 60 });
acl.addEventListener("reading", () => {
  console.log(`Acceleration along the X-axis ${acl.x}`);
  console.log(`Acceleration along the Y-axis ${acl.y}`);
  console.log(`Acceleration along the Z-axis ${acl.z}`);
});
acl.start();
let audio = null;
let audioContext;
let source;
let panner;
let filter;

function AudioSetup() {
  audio = document.getElementById('audio');
  audio.addEventListener('play', () => {
    console.log('play');
    if (!audioContext) {
      audioContext = new AudioContext();
      source = audioContext.createMediaElementSource(audio);
      panner = audioContext.createPanner();
      filter = audioContext.createBiquadFilter();
      // Connect audio nodes
      source.connect(panner);
      panner.connect(filter);
```

```
    filter.connect(audioContext.destination);
    // highshelf filter parameters
    filter.type = 'highshelf';
    filter.Q.value = 10;
    filter.frequency.value = 500;
    filter.gain.value = 18;
    audioContext.resume();
   }
  })
  audio.addEventListener('pause', () => {
   console.log('pause');
   audioContext.resume();
  })
}
function startAudio() {
  AudioSetup();
  let filterCheckbox = document.getElementById('filterCheckbox');
  filterCheckbox.addEventListener('change', function() {
   if (filterCheckbox.checked) {
    // Connect filter when checkbox is checked
    panner.disconnect();
    panner.connect(filter);
    filter.connect(audioContext.destination);
   } else {
    // Disconnect filter when checkbox is unchecked
    panner.disconnect();
    panner.connect(audioContext.destination);
   }
  });
```