МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра цифрових технологій в енергетиці

Варіант - 20

Звіт з графічно-розрахункової роботи з дисципліни «Методи синтезу віртуальної реальності»

Виконав:
Студент 1-го курсу
магістратури
Групи ТР-21мп
Прачов Віталій

Прийняв:
Демчишин А. А.

Київ – 2023

Розрахунково-графічна робота (Просторове аудіо)

Вимоги:

- повторно використовувати код із практичного завдання №2
- реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні за допомогою матеріального інтерфейсу (поверхня залишається нерухомою, а джерело звуку рухається). Відтворити улюблену пісню у форматі mp3/ogg, маючи просторове розташування джерела звуку, кероване користувачем
- візуалізувати положення джерела звуку за допомогою сфери;
- додати Піковий звуковий фільтр. Додати елемент, який вмикає і вимикає фільтр.

Теоретичні відомості

AudioContext є частиною Web Audio API, яка надає можливість маніпулювати аудіо даними у веб-браузері. Він використовується для створення і керування аудіо-графом, що складається з аудіо джерел, обробників та виходів.

AudioContext дозволяє додавати аудіо джерела до аудіо-графу. Аудіо джерела можуть бути звукові файли (наприклад, MP3 або WAV), медіа-потоки або генеровані звуки. Для додавання аудіо джерела використовується метод *createMediaElementSource, createMediaStreamSource, createBufferSource* або інші методи відповідно до типу джерела.

Також AudioContext надає можливість додавати обробники для обробки аудіо сигналу. Обробники можуть включати ефекти, фільтри, еквалайзери та інші елементи обробки звуку. Для створення обробника використовується метод *create*(), типом може бути наприклад BiquadFilterNode, ConvolverNode, DelayNode тощо.

AudioContext може мати один або кілька виходів, через які звук буде відтворюватися. Зазвичай використовується вихід аудіо пристрою користувача. Щоб отримати доступ до виходу, можна використовувати метод destination або створити свій власний вихід, використовуючи метод createMediaStreamDestination.

Разом з Web Audio API, можна використовувати 3D аудіо для створення імерсійного звукового середовища. 3D аудіо дозволяє

розміщувати звукові джерела у тривимірному просторі і відтворювати їх залежно від їх положення та орієнтації у відношенні до слухача.
Основні відомості про 3D аудіо з Web Audio API:

Для розміщення звукового джерела в 3D просторі можна використовувати об'єкт PannerNode. Цей об'єкт дозволяє встановлювати позицію джерела за допомогою координат x, y та z і встановлювати його орієнтацію за допомогою векторів forwardX, forwardY, forwardZ та upX, upY, upZ.

Peaking filter - є одним з типів фільтрів, які використовуються для обробки звуку. Він дозволяє підсилити або приглушити окремі частоти в аудіо сигналі, створюючи пікову амплітудну відповідь навколо певної центральної частоти. У цьому типі фільтра значення параметрів такі:

Q: ширина смуги частот, які посилюються. Велике значення означає вузьку ширину.

frequency: центральна частота діапазону підвищення.

gain: посилення в дБ, яке буде застосовано. Якщо значення негативне, частоти послаблюються.

Опис деталей реалізації

Для опису реалізації сценарію, який включає створення сфери яка обертається та завантаження звукової доріжки, а також використання датчиків пристрою для виставлення позиції звука, необхідно розглянути етапи:

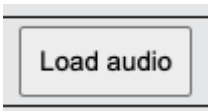
1. Завантаження звукової доріжки:
    - Створення нового об'єкта XMLHttpRequest для виконання асинхронного запиту на сервер для отримання звукового файлу.
    - Встановлення обробників подій для відстеження прогресу завантаження та успішного завершення запиту.
    - Завантаження звукового файлу за допомогою методу open та send і збереження його у змінну.
2. Створення нового AudioContext:
    - Використовується new AudioContext() для створення нового екземпляра AudioContext.
3. Створення та налаштування фільтра:
    - Використовується метод AudioContext, такий як createBiquadFilter(), для створення об'єкта фільтра.
    - Налаштування параметрів фільтра, такі як частота, пікове підсилення або приглушення та ширина піка.
4. Накладання фільтра на звук:

- Створюємо новий AudioNode для відтворення завантаженої звукової доріжки.
- Підключаємо вихідний вузол AudioNode до вхідного вузла фільтра.
- Підключаємо вихідний вузол фільтра до вхідного вузла AudioContext

5. Додаємо сферу аналогічно до попередніх робіт, та надаємо поведінку з обертання навколо фігури, у випадку якщо користувач натискає Device Orientation, обертання виконується на базі датчиків з пристрою.
6. Відповідно до координат сфери встановлюємо просторову позицію звука (*panner.setPosition)*

Інструкції користувача

1. Натиснути **Load audio** - завантаження звукової доріжки



2. Натиснути на флажок **Peaking filter** - (увімкнути/вимкнути) звуковий фільтр

3. Натиснути на флажок **Device Orientation** - (увімкнути/вимкнути) обертання сфери (візуалізація звука) за допомогою датчиків з пристрою.

Опис вихідного коду

Код з підвантаження звукової доріжки:

```
export const getSoundBuffer = (soundFileName) : Promise<unknown>  => {
  return new Promise( executor: (resolve, reject) : void  => {
    const request : XMLHttpRequest  = new XMLHttpRequest();
    request.open( method: "GET", soundFileName,  async: true);
    request.responseType = "arraybuffer";
    request.onload = function (e : ProgressEvent ) : void  {
      resolve(request.response);
    };
    request.send();
  })
}
```

Код з ініціалізації AudioContext'ту, та створення фільтру з параметрами

```
export const loadAudio = async (soundFileName) : Promise<[{volume: GainNode, source: Au...>  => {
  ctx = new AudioContext();
  const mainVolume : GainNode  = ctx.createGain();
  mainVolume.connect(ctx.destination);
  const sound : {source: AudioBufferSourceNode, volume: GainNode}  = {
    source: ctx.createBufferSource(),
    volume: ctx.createGain(),
  }
  sound.source.connect(sound.volume);
  sound.volume.connect(mainVolume);
  sound.source.loop = true;
  const soundBuffer = await getSoundBuffer(soundFileName);
  try {
    sound.buffer = await ctx.decodeAudioData(soundBuffer)
    sound.source.buffer = sound.buffer;
    sound.source.start(ctx.currentTime);
  } catch (e) {
    console.error(e)
  }
  panner = ctx.createPanner();
  filter = ctx.createBiquadFilter();
  sound.source.connect(panner);
  panner.connect(filter);
  filter.connect(ctx.destination);
  filter.type = 'peaking';
  filter.frequency.value = 1000;
  filter.gain.value = 25;
  filter.Q.value = 1;
  return [sound, panner];
}
```

Код увімкнення/вимкнення фільтру

```
export const handleFilterChange = () : void  => {
  const filterElement : HTMLElement  = document.getElementById( elementId: 'filter');
  filterElement.addEventListener( type: 'change',  listener: async (e : Event ) : Promise<void>  => {
    if (filterElement.checked) {
      panner?.disconnect()
      panner?.connect?.(filter)
      filter?.connect?.(ctx.destination)
    } else {
      panner?.disconnect()
      panner?.connect?.(ctx.destination)
    }
  });
}
```