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 - 2

Виконав:
Студент 1-го курсу
групи ТР-31мп, ІАТЕ
Булига Микола
Олександрович

Київ - 2023

Завдання

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

Вимоги:

- Перевикористати код з практичної роботи №2.
- Реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні за допомогою матеріального інтерфейсу (поверхня при цьому залишається нерухомою, а джерело звуку рухається). Відтворювати улюблену пісню у форматі mp3/ogg, при цьому просторове положення джерела звуку контролюється користувачем;
- Візуалізувати джерело звуку у вигляді сфери.
- Додати звуковий фільтр за варіантом. Додати «галочку», яка вмикає чи вимикає фільтр.

Теоретичні відомості

Web Audio API представляє собою потужний інструмент, що дозволяє розробникам маніпулювати та синтезувати звук у веб-додатках. Вона надає набір інтерфейсів та об'єктів, що дозволяють створювати, змінювати та маршрутизувати аудіосигнали в реальному часі. Один з ключових аспектів API веб-аудіо полягає у його здатності обробляти аудіо та керувати ним за допомогою модульного підходу.

Серед численних об'єктів, доступних у API веб-аудіо, широко використовувані є:

- AudioContext;
- MediaElementSourceNode;
- PannerNode
- BiquadFilterNode;

AudioContext

Центральний елемент, який представляє граф обробки аудіо і служить точкою входу для створення та підключення аудіовузлів. Цей об'єкт дозволяє розробникам отримувати доступ до різноманітних методів і властивостей для керування відтворенням аудіо, маршрутизацією та ефектами. Ініціалізуючи об'єкт AudioContext за допомогою коду "context = new AudioContext();", ми створюємо основу для конвеєра обробки звуку.

MediaElementSourceNode

Використовується для отримання аудіоданих з елементів медіа веб-сторінки, таких як елементи <audio> або <video>. Цей об'єкт служить джерелом аудіо, яке можна підключити до інших аудіовузлів для подальшої обробки або маршрутизації.

PannerNode

Відповідає за просторове позиціонування та панорамування звуку. Він імітує тривимірний звук, регулюючи положення, орієнтацію та швидкість звукових джерел у віртуальному тривимірному просторі.

BiquadFilterNode

Реалізує різні типи цифрових фільтрів, такі як низькочастотні, високочастотні, смугові та пікові фільтри. Це дозволяє розробникам формувати частотну характеристику аудіосигналу, змінюючи його тембр та застосовуючи ефекти, такі як вирівнювання або резонанс.

У підсумку, 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 об'єкти, такі як MediaElementSourceNode, PannerNode та BiquadFilterNode, пропонують спеціальні функції для отримання аудіоданих, розміщення звуку у тривимірному просторі та застосування ефектів цифрової фільтрації. За допомогою цих об'єктів та можливостей Web Audio API розробники можуть створювати захоплюючі та інтерактивні аудіо в Інтернеті.

Виконання завдання

Для початку завантажимо аудіо з яким будемо працювати в розрахунковій роботі за допомогою вище описаного Web Audio API. Аудіо, яке буде програватися, буде завантажено за допомогою XMLHttpRequest класу з віддаленого GitHub репозиторію. А використовуючи Web Audio API буде створено аудіо контекст, фільтр та панер, які будуть підключені одні до одного, щоб відтворювати звук згідно заданих налаштувань.

```
function createAudio() {
   audioContext = new (window.AudioContext || window.webkitAudioContext)();
   audioSource = audioContext.createBufferSource();
   createHighpassFilter();
   createAudioPanner();


   const request = new XMLHttpRequest();
   request.open("GET", "https://raw.githubusercontent.com/Sykess3/WebGL-basics/CGW/sound.mp3", true);
   request.responseType = "arraybuffer";
   request.onload = () => {
      const audioData = request.response;
      audioContext.decodeAudioData(audioData, (buffer) => {
         audioSource.buffer = buffer;
         if (useFilter) {
            audioSource.connect(audioFilter);
            audioFilter.connect(audioPanner);
         } else {
            audioSource.connect(audioPanner);
         }
         audioPanner.connect(audioContext.destination);
```

```
        audioSource.loop = true;
        audioSource.start(0);
      }, (err) => {alert(err)});
  };
  request.send();
}
```

Згідно варіанту необхідно було реалізувати фільтр високих частот. Для цього необхідно створити BiquadFilter використовуючи AudioContext та встановити цьому фільтру бажаний тип.

```
function createHighpassFilter() {
  audioFilter = audioContext.createBiquadFilter();
  audioFilter.type = "highpass";
  audioFilter.frequency.value = 1000;
  audioFilter.Q.value = 1;
}
```

Також необхідно створити Panner, його буде використано для можливості керування аудіо та розміщення його в просторі.

```
function createAudioPanner() {
  audioPanner = audioContext.createPanner();
  audioPanner.panningModel = "HRTF";
  audioPanner.distanceModel = "inverse";
  audioPanner.refDistance = 1;
  audioPanner.maxDistance = 1000;
  audioPanner.rolloffFactor = 1;
  audioPanner.coneInnerAngle = 360;
  audioPanner.coneOuterAngle = 0;
  audioPanner.coneOuterGain = 0;
}
```

Для можливості вимикання та вмикання фільтра під час програвання звуку необхідно додати прапорець та приєднати/від’єднати фільтр відповідно до встановленого значення в цьому полі.

```
function setupUseFilterEvent() {
    const checkbox = document.getElementById('useFilter');
    checkbox.addEventListener('change', (event) => {
        if (event.target.checked) {
            useFilter = true;
            if (audioContext) {
                audioSource.disconnect();
                audioPanner.disconnect();
                audioSource.connect(audioFilter);
                audioFilter.connect(audioPanner);
                audioFilter.connect(audioContext.destination);
            }
        } else {
            useFilter = false;
            if (audioContext) {
                audioSource.disconnect();
                audioPanner.disconnect();
                audioSource.connect(audioPanner);
                audioPanner.connect(audioContext.destination);
            }
        }
    });
}
```

Для можливості змінювати звук в просторі було використано даний код:

```
function ExecuteAnimation(){
   if(!isAnimating){
      return;
   }
   let deltaTime = 10 / fps;
   sphereRotation.x = (Math.sin(currentAnimationTime / 500 * 100) * ModelRadius / 5);
   sphereRotation.y = (Math.cos(currentAnimationTime / 500 * 100) * ModelRadius / 5) * 1.2;
   sphereRotation.z = (Math.sin(currentAnimationTime / 500 * 100) * ModelRadius / 5) +
(Math.cos(currentAnimationTime / 500 * 100) * ModelRadius / 4);


   audioPanner.setPosition(sphereRotation.x, sphereRotation.y, 0);
   audioPanner.setOrientation(0,0,0);


   currentAnimationTime += deltaTime;
   setTimeout(() => {
      reqAnim = window.requestAnimationFrame(ExecuteAnimation);
   }, deltaTime);
}
```

Для відображення позиції звука в просторі було додано сферу, яка змінює своє положення разом із звуком.

Вказівки для користувача

Користувач має можливість увімкнути аудіо за допомогою кнопки “Play” див. рисунок 1.

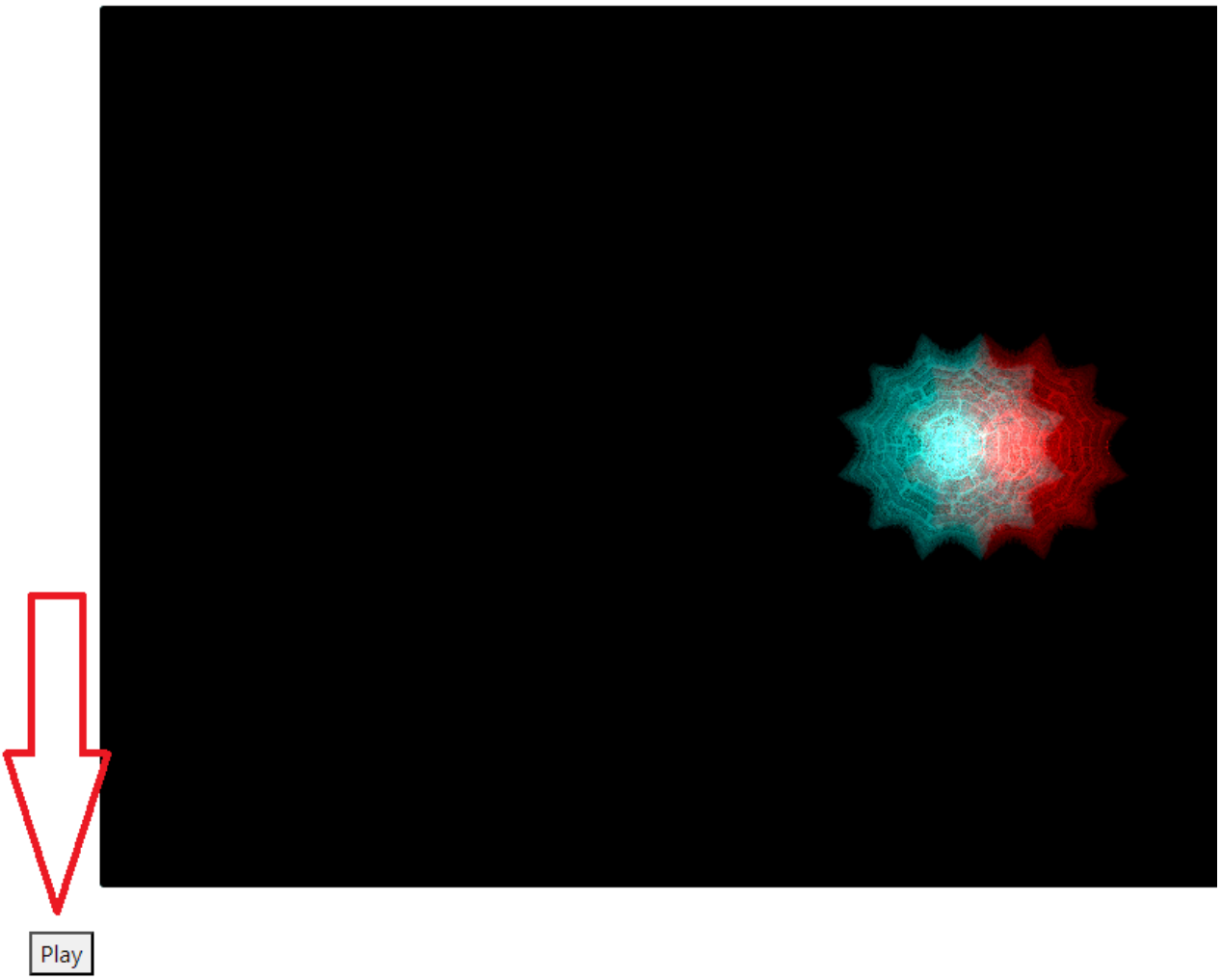


Рисунок 1 - Увімкнення аудіо

Користувач може увімкнути/вимкнути фільтр високих частот за допомогою прапорця зображеного на рисунку 2.

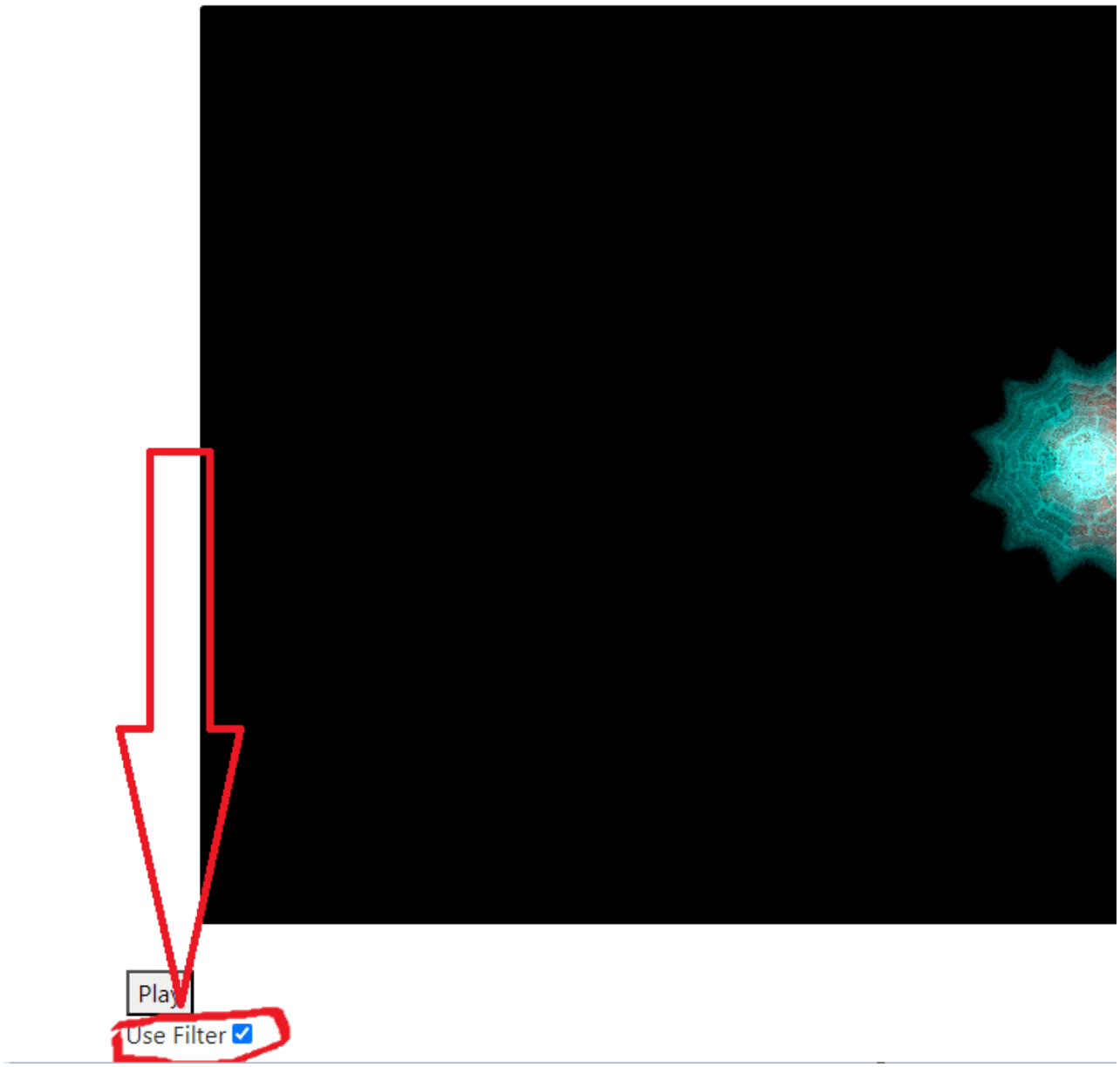
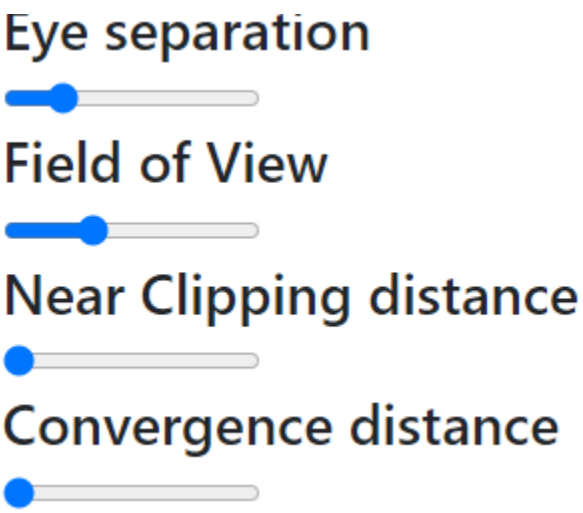


Рисунок 2 - Прапорець для увімкнення/вимкненн фільтру високих частот

Також представлені елементи управління стерео зображенням:



Приклад вихідного коду

```
let audioContext;

let audioSource;

function createAudio() {

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

    audioSource = audioContext.createBufferSource();

    createHighpassFilter();

    createAudioPanner();


    const request = new XMLHttpRequest();

    request.open("GET",
"https://raw.githubusercontent.com/Sykess3/WebGL-basics/CGW/sound.mp3", true);

    request.responseType = "arraybuffer";

    request.onload = () => {
```

```
        const audioData = request.response;
        audioContext.decodeAudioData(audioData, (buffer) => {
            audioSource.buffer = buffer;
            if (useFilter) {
                audioSource.connect(audioFilter);
                audioFilter.connect(audioPanner);
            } else {
                audioSource.connect(audioPanner);
            }
            audioPanner.connect(audioContext.destination);
            audioSource.loop = true;
            audioSource.start(0);
        }, (err) => {alert(err)});
    };
    request.send();
}

let audioFilter;
function createHighpassFilter() {
    audioFilter = audioContext.createBiquadFilter();
    audioFilter.type = "highpass";
    audioFilter.frequency.value = 1000;
    audioFilter.Q.value = 1;
}
```

```
let audioPanner;

function createAudioPanner() {

    audioPanner = audioContext.createPanner();

    audioPanner.panningModel = "HRTF";

    audioPanner.distanceModel = "inverse";

    audioPanner.refDistance = 1;

    audioPanner.maxDistance = 1000;

    audioPanner.rolloffFactor = 1;

    audioPanner.coneInnerAngle = 360;

    audioPanner.coneOuterAngle = 0;

    audioPanner.coneOuterGain = 0;

}


function setupUseFilterEvent() {

    const checkbox = document.getElementById('useFilter');

    checkbox.addEventListener('change', (event) => {

        if (event.target.checked) {

            useFilter = true;

            if (audioContext) {

                audioSource.disconnect();

                audioPanner.disconnect();

                audioSource.connect(audioFilter);

                audioFilter.connect(audioPanner);

                audioFilter.connect(audioContext.destination);

            }

        } else {
```

```
            useFilter = false;
            if (audioContext) {
                audioSource.disconnect();
                audioPanner.disconnect();
                audioSource.connect(audioPanner);
                audioPanner.connect(audioContext.destination);
            }
        }
    });
}
```