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”

Виконав студент групи ТР-32мп  
Швагрун Андрій

Київ 2024

**Постановка задачі**

Основні завданні

* повторно використати код з практичного завдання №2;
* для тих, хто має сертифікат з курсу FPV дронів: реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні по колу в часі (поверхня при цьому залишається нерухомою, а джерело звуку рухається). Відтворити улюблену пісню у форматі mp3/ogg, при цьому просторове положення джерела звуку контролюється користувачем;
* візуалізувати положення джерела звуку за допомогою сфери;
* додати звуковий фільтр (за допомогою інтерфейсу BiquadFilterNode) відповідно до варіанту (1варіант **—** Фільтр низьких частот). Додайте елемент з прапорцем, який вмикає або вимикає фільтр. Налаштуйте параметри фільтра на свій смак.

**Теоретичні матеріали**

**Просторове аудіо в Web Audio API**

Просторове аудіо – це метод обробки звуку, який дозволяє створювати ефект присутності джерела звуку в певному місці тривимірного простору. Використання просторового аудіо може значно підвищити реалістичність віртуальних середовищ, таких як відеоігри та віртуальна реальність.

Web Audio API – це потужний інструмент для роботи зі звуком у веб-браузерах, який дозволяє створювати, обробляти та відтворювати аудіо у реальному часі. Основні компоненти Web Audio API для просторового аудіо включають:

**AudioContext**: Основний інтерфейс для роботи з аудіо. Він створює контекст, в якому виконується обробка звуку.

**PannerNode**: Нода для реалізації просторового звуку. Вона дозволяє розміщувати звукове джерело в тривимірному просторі, задаючи координати X, Y та Z.

**BiquadFilterNode**: Нода для створення різних типів аудіо фільтрів, таких як низькочастотні (lowpass), високочастотні (highpass), смугові (bandpass) та інші.

**Обертання звуку**

Обертання звуку навколо сцени досягається за допомогою зміни позиції PannerNode у просторі. Цей процес вимагає оновлення координат звукового джерела відповідно до часу. Використовуючи функції Math.sin та Math.cos, можна обчислити нові координати для звукового джерела, створюючи ефект обертання.

function moveSphere() {

let radius = 1.0;

if (startTime === null) {

startTime = performance.now();

}

let currentTime = performance.now();

let elapsedTime = (currentTime - startTime) / 1000; // Time in seconds

// Update sphere coordinates for circular motion

sphereCoords.x = radius \* Math.sin(elapsedTime);

sphereCoords.y = 0;

sphereCoords.z = radius \* Math.cos(elapsedTime); // Assuming the sphere moves in the XY plane

// Update the position of the sound panner to match the sphere's coordinates

setPannerPosition(sphereCoords.x, sphereCoords.y, sphereCoords.z);

}

**Фільтр низьких частот**

Фільтр низьких частот (Low-Pass Filter, LPF) – це електронний фільтр, який пропускає частоти, нижчі за певне значення (частота зрізу), і ослаблює частоти, вищі за це значення. В аудіо обробці, фільтр низьких частот часто використовується для згладжування сигналів, видалення шуму та покращення якості звуку.

У контексті Web Audio API, BiquadFilterNode надає простий спосіб створення низькочастотного фільтра.

**Принцип роботи**

Фільтр низьких частот працює за рахунок того, що знижує амплітуду високочастотних компонентів сигналу, залишаючи низькочастотні компоненти без змін. Це досягається за допомогою математичних алгоритмів, які обробляють вхідний сигнал і відсікають високочастотні складові.

Фільтр низьких частот дозволяє зменшити кількість високочастотних шумів і покращити якість звуку, що є важливим у різних аудіо-застосунках, від музичних плеєрів до систем обробки мови.

**Аспекти реалізації**

Реалізовано використання фільтра низьких частот для обробки звукового сигналу в аудіосистемі з використанням WebAudio API. Фільтр низьких частот дозволяє пропускати низькі частоти нижче заданої частоти зрізу, що додає м'якості і тепла звуку, зменшуючи при цьому шум на високих частотах.

Цей функціонал дозволяє покращити якість відтворення аудіо у просторовій аудіосистемі, надаючи можливість регулювати низьку частотну характеристику звуку залежно від потреб користувача. Фільтри низьких частот особливо корисні для усунення небажаних високочастотних шумів та для створення більш природного і комфортного звукового середовища.

Першим кроком для імплементації аудіосистеми є створення HTML-елемента <audio>, який містить інформацію про джерело аудіо-доріжки, в нашому випадку це файл song.mp3. Цей елемент керування дозволяє зупиняти та продовжувати відтворення аудіо-доріжки, забезпечуючи базову взаємодію користувача з аудіо-контентом.

Після цього, за допомогою JavaScript, а саме WebAudio API, було створено об'єкт аудіо контексту (AudioContext), який служить основою для всіх аудіо-операцій. До цього контексту було під'єднано три основних об'єкти:

* Джерело звуку (MediaElementSource): Це джерело звуку, яке бере аудіо з HTML-елемента <audio>. Воно забезпечує передачу аудіо даних до аудіо контексту для подальшої обробки.
* Об'єкт обробки просторового аудіо (Panner): Цей об'єкт використовується для розташування звуку у тривимірному просторі. Він дозволяє створювати ефекти обертання та переміщення звуку навколо користувача, що робить аудіо досвід більш реалістичним і захоплюючим.
* Звуковий фільтр (BiquadFilter): Це багатофункціональний фільтр, який може бути налаштований для виконання різних типів фільтрації. Для нашої реалізації було обрано тип фільтра lowpass.

**Реалізація фільтра низьких частот**

Для реалізації фільтра низьких частот було обрано тип фільтра lowpass. Наступним кроком було встановлення параметрів обраного фільтра. Загалом, BiquadFilter має три параметри для налаштування:

* frequency (частота зрізу): Це частота, вище якої частоти послаблюються. Вибір правильної частоти зрізу є ключовим для досягнення бажаного звукового ефекту.
* Q (ширина смуги): Цей параметр контролює, наскільки піковим буде відгук на частоті зрізу. Велике значення Q робить відповідь більш піковою, що може бути корисним для акцентування певних частотних діапазонів.
* gain (підсилення): У випадку фільтра низьких частот цей параметр не використовується, оскільки основна функція фільтра - це послаблення частот вище частоти зрізу.

**Візуалізація джерела звуку**

Джерело звуку, обробка якого в просторі здійснюється об'єктом класу Panner, зображено у WebGL контексті у вигляді сфери. Сфера може переміщуватись, змінюючи параметри об'єкта Panner, щоб при прослуховуванні аудіо зі стерео звуком був ефект переміщення джерела звуку відповідно до місцезнаходження сфери в системі координат.

**Обертання звуку**

Обертання звуку реалізується шляхом зміни положення джерела звуку у тривимірному просторі. Об'єкт Panner має методи для встановлення координат джерела звуку, що дозволяє створювати динамічні звукові сцени. Наприклад, звук може переміщуватися навколо слухача, що створює ефект обертання.

**Інструкція користувача**Для налаштування просторового зображення поверхні на панелі керування доступні 4 слайдери, якими можна налаштувати наступні параметри:

1. Відстань між очима
2. Кут огляду
3. Відстань ближнього відрізання.
4. Збіжність

Також на панелі присутні плеєр відтворення аудіозапису та чекбокс для увімкнення/вимкнення звукового фільтру.

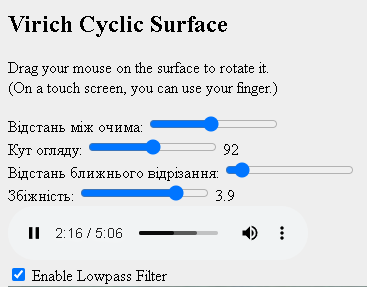


Рисунок 1 - Панель керування

Приклад зміни даних параметрів:





Рисунки 2 – 3 — Приклад зміни параметрів 3-D відображення

**Код програми:**function initializeAudioControls() {  
 const audioElement = *document*.getElementById('audio');  
 const lowpassCheckbox = *document*.getElementById('lowpass');  
  
 function initializeAudio() {  
 *audioContext* = new (*window*.*AudioContext* || *window*.webkitAudioContext)();  
 *track* = *audioContext*.createMediaElementSource(audioElement);  
  
 *panner* = *audioContext*.createPanner();  
 *panner*.panningModel = 'HRTF';  
 *panner*.distanceModel = 'inverse';  
 *panner*.refDistance = 1;  
 *panner*.maxDistance = 10000;  
 *panner*.rolloffFactor = 1;  
 *panner*.coneInnerAngle = 360;  
 *panner*.coneOuterAngle = 0;  
 *panner*.coneOuterGain = 0;  
  
 *lowpassFilter* = *audioContext*.createBiquadFilter();  
 *lowpassFilter*.type = 'lowpass';  
 *lowpassFilter*.frequency.value = 1000;  
  
 updateAudioRouting();  
 }  
  
 function updateAudioRouting() {  
 if (lowpassCheckbox.checked) {  
 *track*.disconnect();  
 *track*.connect(*lowpassFilter*).connect(*panner*).connect(*audioContext*.destination);  
 } else {  
 *track*.disconnect();  
 *track*.connect(*panner*).connect(*audioContext*.destination);  
 }  
 }  
  
 lowpassCheckbox.addEventListener('change', updateAudioRouting);  
  
 audioElement.addEventListener('play', function () {  
 if (!*audioContext*) {  
 initializeAudio();  
 }  
 });  
  
 audioElement.play().catch(() => {  
 *console*.log('Audio playback prevented. User interaction required.');  
 });  
}

function moveSphere() {  
 let radius = 1.0;  
 if (*startTime* === null) {  
 *startTime* = *performance*.now();  
 }  
 let currentTime = *performance*.now();  
 let elapsedTime = (currentTime - *startTime*) / 1000; // Time in seconds  
  
 // Update sphere coordinates for circular motion  
 *sphereCoords*.x = radius \* *Math*.sin(elapsedTime);  
 *sphereCoords*.y = 0;  
 *sphereCoords*.z = radius \* *Math*.cos(elapsedTime); // Assuming the sphere moves in the XY plane  
 // Update the position of the sound panner to match the sphere's coordinates  
 setPannerPosition(*sphereCoords*.x, *sphereCoords*.y, *sphereCoords*.z);  
}

function drawSphere(modelViewProjection) {  
 moveSphere();  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, *sphere*.positionBuffer);  
 *gl*.vertexAttribPointer(*shProgram*.iAttribVertex, 3, *gl*.*FLOAT*, false, 0, 0);  
 *gl*.enableVertexAttribArray(*shProgram*.iAttribVertex);  
  
 *gl*.bindBuffer(*gl*.*ELEMENT\_ARRAY\_BUFFER*, *sphere*.indexBuffer);  
  
 *gl*.uniform4fv(*shProgram*.iColor, [1, 0, 0, 1]); // Sphere color (red)  
 *gl*.uniformMatrix4fv(*shProgram*.iModelViewProjectionMatrix, false, m4.translation(*sphereCoords*.x, *sphereCoords*.y, *sphereCoords*.z));  
  
 *gl*.drawElements(*gl*.*TRIANGLES*, *sphere*.vertexCount, *gl*.*UNSIGNED\_SHORT*, 0);  
}

function CreateSphereData() {  
 const latitudeBands = 30;  
 const longitudeBands = 30;  
 const radius = 0.05;  
  
 const vertexPositionData = [];  
 const indexData = [];  
  
 for (let latNumber = 0; latNumber <= latitudeBands; latNumber++) {  
 const theta = latNumber \* *Math*.PI / latitudeBands;  
 const sinTheta = *Math*.sin(theta);  
 const cosTheta = *Math*.cos(theta);  
  
 for (let longNumber = 0; longNumber <= longitudeBands; longNumber++) {  
 const phi = longNumber \* 2 \* *Math*.PI / longitudeBands;  
 const sinPhi = *Math*.sin(phi);  
 const cosPhi = *Math*.cos(phi);  
  
 const x = cosPhi \* sinTheta;  
 const y = cosTheta;  
 const z = sinPhi \* sinTheta;  
  
 vertexPositionData.push(radius \* x);  
 vertexPositionData.push(radius \* y);  
 vertexPositionData.push(radius \* z);  
 }  
 }  
  
 for (let latNumber = 0; latNumber < latitudeBands; latNumber++) {  
 for (let longNumber = 0; longNumber < longitudeBands; longNumber++) {  
 const first = (latNumber \* (longitudeBands + 1)) + longNumber;  
 const second = first + longitudeBands + 1;  
 indexData.push(first);  
 indexData.push(second);  
 indexData.push(first + 1);  
  
 indexData.push(second);  
 indexData.push(second + 1);  
 indexData.push(first + 1);  
 }  
 }  
  
 const positionBuffer = *gl*.createBuffer();  
 *gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, positionBuffer);  
 *gl*.bufferData(*gl*.*ARRAY\_BUFFER*, new *Float32Array*(vertexPositionData), *gl*.*STATIC\_DRAW*);  
  
 const indexBuffer = *gl*.createBuffer();  
 *gl*.bindBuffer(*gl*.*ELEMENT\_ARRAY\_BUFFER*, indexBuffer);  
 *gl*.bufferData(*gl*.*ELEMENT\_ARRAY\_BUFFER*, new *Uint16Array*(indexData), *gl*.*STATIC\_DRAW*);  
  
 return {  
 positionBuffer: positionBuffer,  
 indexBuffer: indexBuffer,  
 vertexCount: indexData.length  
 };  
}