МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра ЦТЕ

**ЗВІТ**

розрахунково-графічної роботи

з дисципліни: «Методи синтезу віртуальної реальності»

Варіант №5

**Виконав:**

Ст. гр. ТР-22мп

Головакін М. А.

**Перевірив:**

Демчишин А. А.

Київ – 2023

# Постановка завдання

**Мета**: навчитись роботі з просторовим звуком та налаштуванням звукових фільтрів у webGL використовуючи WebAudio HTML5 API.

**Вимоги:**

* Повторно використати код з практичного завдання №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 положення джерела звуку за допомогою сфери;
* додати Шелфовий фільтр високих частот для 5 варіанту (використовуючи інтерфейс BiquadFilterNode). Додати елемент, який вмикає або вимикає фільтр. Налаштувати параметри фільтра на свій смак.

# Теоретичні відомості

Звуковий контекст (AudioContext) є центральним елементом веб-застосунку, що працює з просторовим звуком і звуковими фільтрами за допомогою WebAudio HTML5 API. Цей контекст відповідає за створення, налаштування та маніпулювання аудіооб'єктами. Він служить посередником між джерелами звуку, ефектами та виходом звуку.

Звукові вузли (AudioNode) використовуються для створення та керування звуковими об'єктами. Вони можуть бути звуковими джерелами (AudioSourceNode), які відтворюють аудіофайли або генерують звук в реальному часі, звуковими ефектами (AudioEffectNode), які застосовують обробку і зміну до звукового сигналу, або звуковими виходами (AudioDestinationNode), які відправляють звуковий сигнал на аудіообладнання пристрою.

Для створення ефектів просторового звучання можна використовувати параметри позиції (x, y, z) для розміщення звукових джерел у тривимірному просторі. Це дозволяє створювати реалістичні ефекти звуку, де звук може рухатися зліва направо, вгору або назад відносно слухача. Для цього можна використовувати методи setPosition() і setVelocity() для налаштування координат та швидкості звукових джерел.

WebAudio API також надає можливість налаштування звукових фільтрів для обробки звукового сигналу. Наприклад, фільтри можуть використовуватися для зміни частотного відгуку або створення ефектів, таких як ехо або реверберація. Звукові фільтри можуть бути додані до звукових вузлів за допомогою методу connect(), що дозволяє створювати послідовні або паралельні ланцюжки обробки звуку.

Загалом, використання WebAudio HTML5 API у поєднанні з WebGL дозволяє створювати вражаючі веб-додатки з просторовим звучанням та високоякісним аудіовізуальним досвідом. Це відкриває широкі можливості для створення інтерактивних аудіо-візуальних презентацій, ігор та інших веб-додатків, що використовують багатогранний звуковий контент.

Шелфовий фільтр високих частот є одним з типів звукових фільтрів, які можна використовувати у WebAudio HTML5 API для обробки звукового сигналу. Цей фільтр дозволяє змінювати амплітуду звукових частот вище заданої точки відсічення (кросовер-частоти).

Шелфовий фільтр високих частот має два основні параметри:

1. Cutoff Frequency: Це точка, де фільтр починає зменшувати амплітуду звукових частот. Всі частоти вище кросовер-частоти будуть приглушені або знижені у рівні амплітуди.
2. Slope: Це параметр, який визначає, наскільки швидко амплітуда знижується після кросовер-частоти. Вища значення похилості призводить до більш різкого зниження амплітуди після кросовер-частоти.

Застосування шелфового фільтра високих частот дозволяє контролювати наявність високочастотних складових у звуковому сигналі. Наприклад, його можна використовувати для зменшення шуму, зайвого шипіння або непотрібних високочастотних складових у звуковому потоці.

У WebAudio API шелфовий фільтр високих частот можна створити за допомогою класу BiquadFilterNode. Необхідно встановити значення типу фільтра на "highshelf" і налаштувати кросовер-частоту та похилість за допомогою параметрів frequency, Q та gain.

# Деталі реалізації

Перше з чого почалась робота – це з налаштування шелфового фільтру високих частот. Була обрана відповідна звукова доріжка та накладено фільтр.

У даному випадку було створено контекст аудіодоріжки, паннер та фільтр. Вони були поєднані між собою. Після було налаштовано шелфовий фільтр високих частот з кросовер-частотою 500 *Гц* та похилістю 10 *дБ/октаву*.

source = audioContext.createMediaElementSource(audio);

panner = audioContext.createPanner();

filter = audioContext.createBiquadFilter();

source.connect(panner);

panner.connect(filter);

filter.connect(audioContext.destination);

filter.type = 'highshelf';

filter.frequency.value = 500;

filter.gain.value = 10;

Далі необхідно було створити сферу, яка відображає положення звуку в просторі. Будемо використовувати схожу функцію до *createSurfaceData*. Ми створюємо масив вершин *vertexList* для генерації сферичної сітки. Обчислюються координати (*x, y, z*) для кожної вершини. Використовується змінна *radius* для масштабування координат.

const vertexList = [];

for (let l = 0; l <= latitudeBands; l++) {

const theta = (l \* Math.PI) / latitudeBands;

const sinTheta = Math.sin(theta);

const cosTheta = Math.cos(theta);

for (let l = 0; l <= longitudeBands; l++) {

const phi = (l \* 2 \* Math.PI) / longitudeBands;

const sinPhi = Math.sin(phi);

const cosPhi = Math.cos(phi);

const x = radius \* cosPhi \* sinTheta;

const y = radius \* cosTheta;

const z = radius \* sinPhi \* sinTheta;

vertexList.push(x, y, z);

}

}

return { vertexList: vertexList, normalList: null, textureList: null };

Наступним кроком було написати автоматичне переміщення нашої кругової сфери. Маємо декілька змінних функції, які описують обертання. *rotationTick* вказує поточне значення оберту анімації, а *lengthOfRotation* визначає загальну кількість обертів, яку необхідно зробити для повного руху сфери, *radius* визначає радіус сфери.

У функції обчислюється кут angle на основі *rotationTick* та *lenghtOfRotation*. Далі ми використовуємо тригонометричні функції для визначення координат *x* та *z*.

function moveCircleSphere(rotationTick, lengthOfRotation, distanceFromCenterX, centerZ, radius) {

const angle = (rotationTick / lengthOfRotation) \* Math.PI \* 2;

const xmusicSphere = distanceFromCenterX + Math.cos(angle) \* radius;

const zmusicSphere = centerZ + Math.sin(angle) \* radius;

}

# Інструкція використання

Програму можна завантажити, використовуючи *git clone <посилання на репозиторій>* та відкрити *index.html*, однак є можливість перейти на сайт, який вказаний у README.md.

Відкривши сайт, маємо опис основних елементів управління (рис. 1):

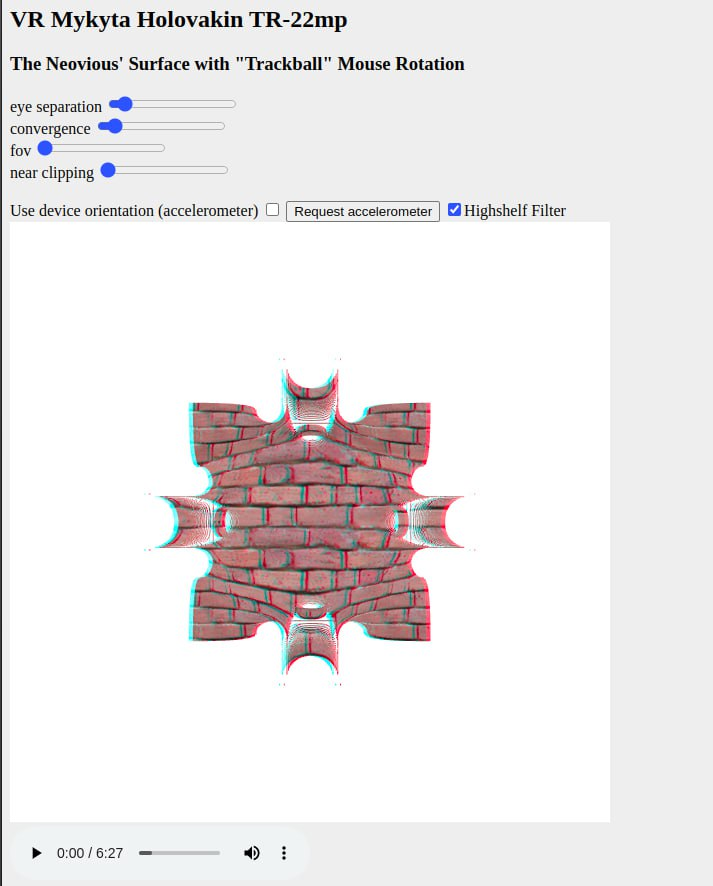


Рис. 1. Головний екран програми

Користувач може налаштувати елементи управління під свій смак, а саме розведення очей (червоного та синього кольору), конвергенція (сходимість кольорів), fov, відсікання ближнього плану. Також присутня можливість вмикання оберту фігури за допомогою акселерометра, однак це доступно лише на телефонах марки Apple.

Навколо відбувається обертання сфери:

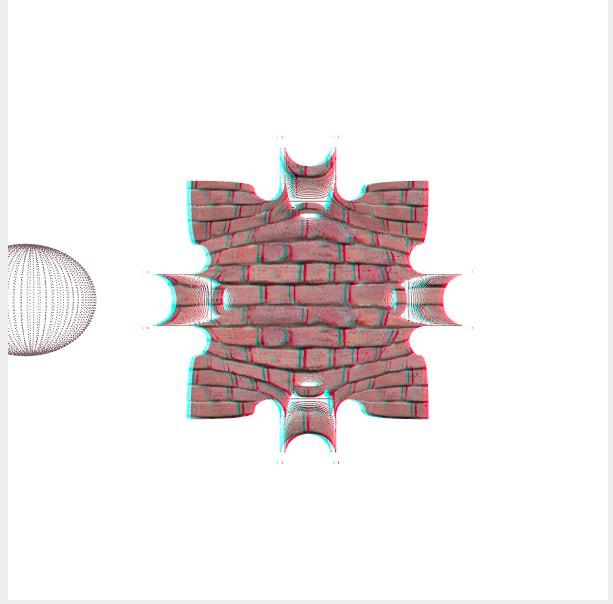


Рис. 2. Переміщення сфери навколо фігури

Сфера мімікує джерело звуку. Переміщуючись навколо поверхні Неовіуса, вона створює ефект просторового звуку.

Також є можливість ввімкнути та вимкнути шелфовий фільтр високих частот:



Рис. 3. Елемент керування шелфовим фільтром

Керування звуковою доріжкою:



Рис. 4. Елемент керування звуковою доріжкою

# Код програми

**Обчислення вершин сфери:**

function createSphere(radius, latitudeBands, longitudeBands) {

const vertexList = [];

for (let l = 0; l <= latitudeBands; l++) {

const theta = (l \* Math.PI) / latitudeBands;

const sinTheta = Math.sin(theta);

const cosTheta = Math.cos(theta);

for (let l = 0; l <= longitudeBands; l++) {

const phi = (l \* 2 \* Math.PI) / longitudeBands;

const sinPhi = Math.sin(phi);

const cosPhi = Math.cos(phi);

const x = radius \* cosPhi \* sinTheta;

const y = radius \* cosTheta;

const z = radius \* sinPhi \* sinTheta;

vertexList.push(x, y, z);

}

}

return { vertexList: vertexList, normalList: null, textureList: null };

}

**Налаштування аудіо та фільтру:**

function StartAudio() {

audio = document.getElementById('audio');

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

if (!audioContext) {

audioContext = new window.AudioContext();

source = audioContext.createMediaElementSource(audio);

panner = audioContext.createPanner();

filter = audioContext.createBiquadFilter();

source.connect(panner);

panner.connect(filter);

filter.connect(audioContext.destination);

filter.type = 'highshelf';

filter.frequency.value = 500;

filter.gain.value = 10;

audioContext.resume();

}

});

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

console.log('pause');

audioContext.resume();

});

let filterCheckbox = document.getElementById('filterCheckbox');

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

if (filterCheckbox.checked) {

panner.disconnect();

panner.connect(filter);

filter.connect(audioContext.destination);

} else {

panner.disconnect();

panner.connect(audioContext.destination);

}

});

}

**Обертання сфери навколо поверхні:**

function moveCircleSphere(rotationTick, lengthOfRotation, distanceFromCenterX, centerY, radius) {

const angle = (rotationTick / lengthOfRotation) \* Math.PI \* 2;

xmusicSphere = distanceFromCenterX + Math.cos(angle) \* radius;

zmusicSphere = centerY + Math.sin(angle) \* radius;

}