**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імЕНІ Ігоря Сікорського»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

**ЗВІТ**

з розрахунково графічної роботи

з дисципліни « Методи синтезу віртуальної реальності»

Варіант 21

                                                                    Виконала студентка групи ТР-41мп

Чиж Єлизавета

Перевірив: Демчишин А.А.

                                              Київ – 2025 рік

**Завдання**

Повторно використайте код із практичного завдання №2.

Реалізуйте обертання джерела звуку навколо геометричного центру ділянки поверхні за допомогою тактильного інтерфейсу (цього разу поверхня залишається нерухомою, а джерело звуку рухається). Відтворіть вашу улюблену пісню у форматі MP3/OGG, контролюючи просторове положення джерела звуку користувачем.

Візуалізуйте положення джерела звуку за допомогою сфери.

Додайте звуковий фільтр (використайте інтерфейс BiquadFilterNode) відповідно до варіанта. Додайте елемент-прапорець, який вмикає або вимикає фільтр. Налаштуйте параметри фільтра на ваш смак.

**7, 14, 21, 28**Шелфовий фільтр низьких частот

**Теоретичні відомості**

Просторове аудіо (Spatial Audio) – це технологія, що дозволяє відтворювати звуки таким чином, щоб вони сприймалися слухачем з певного напрямку та відстані у віртуальному 3D-просторі. Це створює більш імерсивний та реалістичний звуковий досвід, імітуючи природне поширення звуку в реальному світі. На відміну від звичайного стерео, де звук рухається лише по горизонтальній площині між двома динаміками, просторове аудіо враховує тривимірне положення джерела звуку відносно слухача.

Ключовими елементами просторового аудіо є:

Джерело звуку (Sound Source): Об'єкт, що випромінює звук. Його положення в 3D-просторі, напрямок та характеристики (наприклад, гучність, колір) визначають, як звук буде сприйматися.

Слухач (Listener): Об'єкт, що сприймає звук. Його положення та орієнтація в 3D-просторі є критично важливими для правильного розрахунку звукового ефекту. У більшості Web Audio API реалізацій слухач представлений об'єктом AudioListener.

Модель поширення звуку (Sound Propagation Model): Алгоритми, що імітують, як звук поширюється від джерела до слухача, враховуючи відстань, перешкоди, відбиття та інші акустичні явища. Web Audio API надає базові моделі загасання (attenuation) та панорамування.

Web Audio API – це потужний JavaScript API для обробки та синтезу звуку в браузері. Він дозволяє створювати складні звукові ефекти, включаючи просторове аудіо, за допомогою концепції "аудіографа" (Audio Graph), де різні вузли (AudioNodes) з'єднуються між собою для обробки звукових даних.

Для реалізації просторового аудіо Web Audio API надає ключові вузли:

AudioContext: Основний об'єкт, що представляє контекст обробки звуку. Всі аудіовузли створюються в рамках цього контексту.

PannerNode: Цей вузол є центральним для просторового аудіо. Він відповідає за просторове розміщення звуку. PannerNode отримує на вхід монофонічний або стереофонічний потік і застосовує просторові ефекти (панорамування, загасання за відстанню) для створення об'ємного звучання.

Позиція (positionX, positionY, positionZ): Визначає 3D-координати джерела звуку відносно слухача.

Орієнтація (orientationX, orientationY, orientationZ): Визначає напрямок, в якому "дивиться" джерело звуку. Це важливо для спрямованих джерел.

Модель загасання (panningModel): Визначає, як звук загасає з відстанню. Поширені моделі:

HRTF (Head-Related Transfer Function): Імітує, як вуха людини сприймають звук з різних напрямків, враховуючи форму голови та вушних раковин. Забезпечує найбільш реалістичне просторове сприйняття.

equalpower: Проста модель, що забезпечує плавне перемикання між каналами.

Шельфовий фільтр низьких частот (lowshelf) – це тип фільтра, який підсилює або послаблює всі частоти нижче заданої частоти зрізу (frequency), при цьому частоти вище цієї точки залишаються відносно незмінними. Він створює "полицю" (shelf) на нижніх частотах частотної характеристики.

type: 'lowshelf'

frequency: Частота, нижче якої починає діяти підсилення/послаблення.

gain: Величина підсилення або послаблення низьких частот у децибелах. Позитивне значення додає "басів", негативне – зменшує.

**Деталі Реалізації Функціоналу**

Ініціалізація (init, initGL, initAudio)

* *init():*

1)Отримує контекст WebGL з <canvas id="webglcanvas">.

2)Викликає initGL() для налаштування всіх графічних ресурсів.

3)Ініціалізує TrackballRotator для взаємодії з камерою.

4)Оновлює параметри стереокамери з елементів UI та встановлює початковий стан для введення адреси сенсора.

* *initGL():*

1)Компілює та лінкує шейдери (які, ймовірно, знаходяться у vertexShaderSource та fragmentShaderSource у окремих файлах) в ShaderProgram.

2)Ініціалізує буферні дані для 3D-моделей: surface (використовуючи CreateNeoviusSurfaceData) та soundSourceModel (сфера, використовуючи CreateSphereData).

3)Налаштовує StereoCamera з заданими параметрами для об'ємного перегляду.

* *initAudio():*

1)Створює новий AudioContext.

2)Ініціалізує PannerNode (panner) з моделлю панорамування HRTF (Head-Related Transfer Function) для максимально реалістичної локалізації звуку, інверсною моделлю відстані та налаштуваннями конуса для спрямованості звуку.

3)Створює GainNode (sourceGainNode) для контролю гучності джерела.

4)Встановлює початкову позицію (0,0,0) та орієнтацію (дивлячись у напрямку негативної осі Z) слухача (audioContext.listener).

5)Ініціалізує BiquadFilterNode (biquadFilter) з типом highpass (високочастотний фільтр). Важливо: Згідно з вимогами завдання, тут мав би бути "Шелфовий фільтр низьких частот" (lowshelf).

*Стереоскопічний рендеринг:*

Для лівого ока: Розраховується лівий фрустум та матриця зсуву для лівого ока. Поверхня та джерело звуку рендеряться з використанням gl.colorMask(true, false, false, true) (тільки червоний канал).

Для правого ока: Очищається лише буфер глибини. Розраховується правий фрустум та матриця зсуву для правого ока. Поверхня та джерело звуку рендеряться з використанням gl.colorMask(false, true, true, true) (зелений та синій канали).

В кінці gl.colorMask(true, true, true, true) відновлює всі канали.

Обробка Аудіо (connectAudioChain, toggleAudioPlayback, toggleFilter, updateFilterParameters)

*connectAudioChain():* Це ключова функція для керування потоком аудіо. Вона спочатку відключає всі існуючі з'єднання, а потім встановлює їх заново:

Якщо прапорець enableFilterCheckbox увімкнено, потік: audioBufferSource -> sourceGainNode -> biquadFilter -> panner -> audioContext.destination.

Якщо вимкнено, потік: audioBufferSource -> sourceGainNode -> panner -> audioContext.destination (фільтр обходиться).

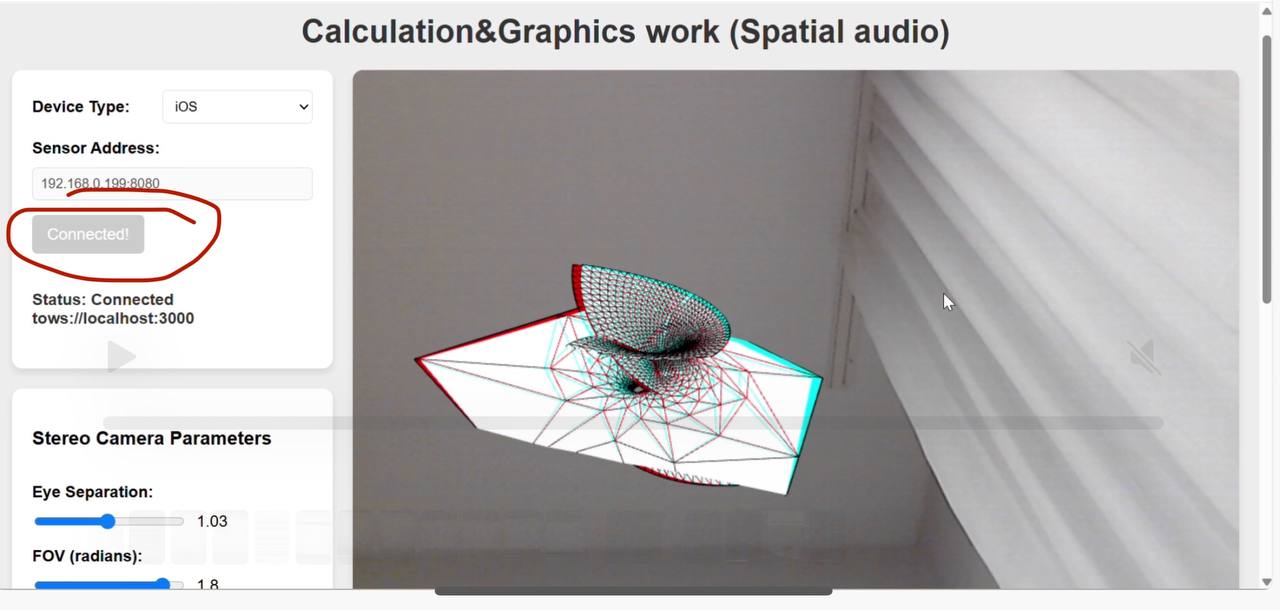
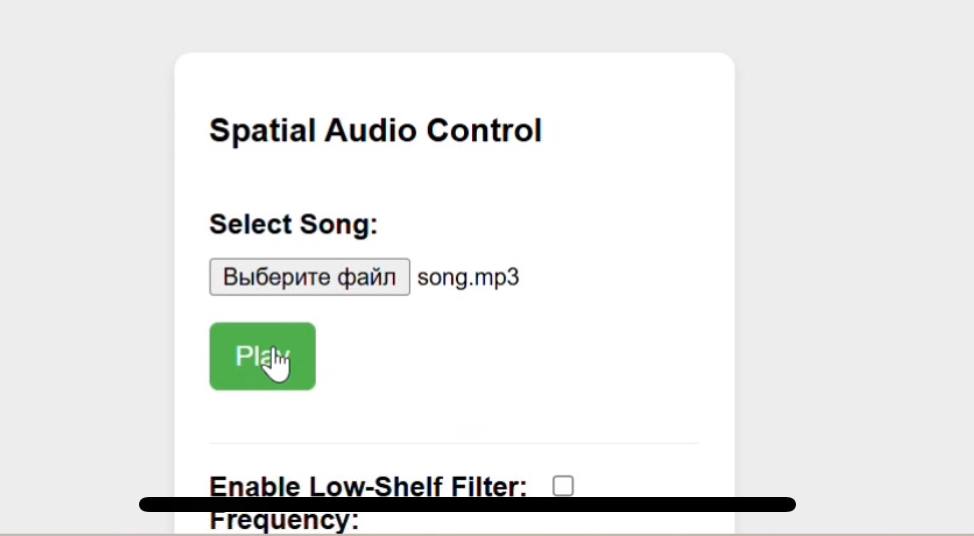
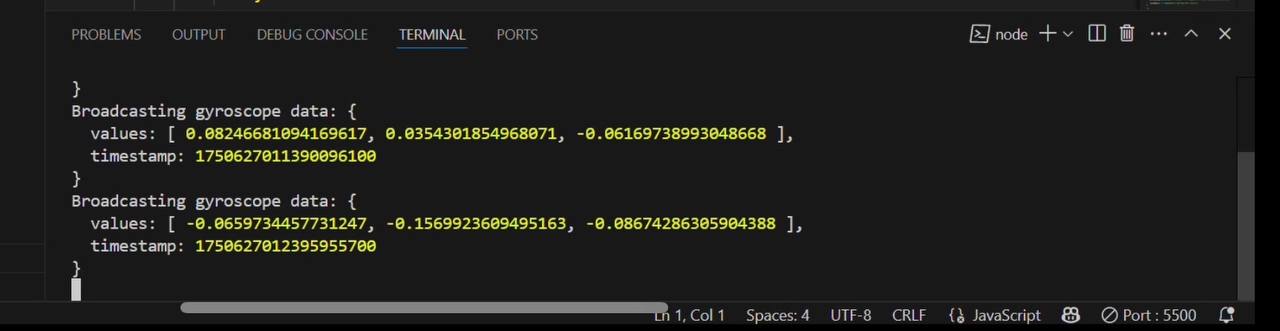
*toggleAudioPlayback():* Керує відтворенням та паузою.

При зупинці (stop()) AudioBufferSourceNode його не можна повторно запустити, тому створюється новий екземпляр з тим самим buffer і підключається заново.

Якщо audioContext призупинений (наприклад, після першого кліку користувача), він відновлюється за допомогою audioContext.resume().

*toggleFilter():* Викликає connectAudioChain() для оновлення зв'язків аудіографа та updateFilterParameters().

*updateFilterParameters():* Оновлює biquadFilter.frequency на основі значення з повзунка filterFrequencyInput, але застосовує зміни до фільтрації *тільки*, якщо фільтр увімкнено.



// Змінні Web Audio API

let audioContext; // Аудіо контекст для обробки звуку.

let panner; // PannerNode для просторового позиціонування звуку.

let biquadFilter; // BiquadFilterNode для застосування аудіофільтрів.

let audioBufferSource; // AudioBufferSourceNode для відтворення аудіоданих.

let sourceGainNode; // GainNode для контролю гучності джерела звуку.

let audioReady = false; // Прапорець, що вказує, чи готовий аудіофайл до відтворення.

let audioPlaying = false; // Прапорець, що вказує, чи відтворюється аудіо.

let lastTimestamp = null; // Часова мітка останнього отриманого пакета даних сенсора.

let connectButton = document.getElementById('connectButton'); // Кнопка для підключення до сенсора.

let statusMessage = document.getElementById('statusMessage'); // Елемент для відображення статусу підключення.

let playPauseButton = document.getElementById('playPauseButton'); // Кнопка для відтворення/паузи аудіо.

let audioFileInput = document.getElementById('audioFile'); // Ввід для вибору аудіофайлу.

let enableFilterCheckbox = document.getElementById('enableFilter'); // Чекбокс для увімкнення/вимкнення фільтра.

let filterFrequencyInput = document.getElementById('filterFrequency'); // Повзунок для частоти фільтра.

let filterGainInput = document.getElementById('filterGain'); // Повзунок для підсилення фільтра.

let filterQInput = document.getElementById('filterQ'); // Повзунок для Q-фактора фільтра.

// Формування адреси WebSocket залежно від типу пристрою.

if (deviceType === "android") {

address = "ws://" + sensorAddress + "/sensor/connect?type=android.sensor.gyroscope";

console.log("Підключення до Android сенсора за адресою " + address);

} else {

// Для iOS припускається, що міст Node.js працює на localhost:3000.

address = "ws://localhost:3000";

console.log("Підключення до мосту сенсора iOS за адресою " + address);

}

connectSensorServer(address);

connectButton.disabled = true; // Вимкнути кнопку після натискання.

connectButton.textContent = "Connecting...";

}

/\*\*

\* Підключається до сенсорного сервера через WebSocket та обробляє дані гіроскопа.

\* @param {string} sensorIp IP-адреса або URL сенсорного сервера.

\*/

function connectSensorServer(sensorIp) {

// Якщо IP-адреса не надана, не підключатися.

if (!sensorIp) return;

let ws = new WebSocket(sensorIp);

// Відновлюємо AudioContext при взаємодії з користувачем (підключення до сенсора)

if (audioContext && audioContext.state === 'suspended') {

audioContext.resume().then(() => {

console.log("AudioContext успішно відновлено.");

draw(); // Перемалювати після готовності аудіоконтексту.

}).catch(e => console.error("Помилка відновлення AudioContext:", e));

} else {

draw(); // Перемалювати сцену.

}

};

// Оновлюємо орієнтацію PannerNode, якщо AudioContext працює і panner визначений.

if (panner && audioContext && audioContext.state === 'running' && audioContext.currentTime > 0) {

// Витягуємо вектор "вперед" з матриці обертання джерела звуку.

let forwardX = -soundSourceRotationMatrix[8]; // -Z з матриці [2][0]

let forwardY = -soundSourceRotationMatrix[9]; // -Z з матриці [2][1]

let forwardZ = -soundSourceRotationMatrix[10]; // -Z з матриці [2][2]

// Витягуємо вектор "вгору" для повної орієнтації.

let upX = soundSourceRotationMatrix[4]; // Y з матриці [1][0]

let upY = soundSourceRotationMatrix[5]; // Y з матриці [1][1]

let upZ = soundSourceRotationMatrix[6]; // Y з матриці [1][2]

// Трансформація для моделі джерела звуку (сфери).

const surfaceCenterX = 0;

const surfaceCenterY = 0;

const surfaceCenterZ = -15; // Z-координата центру поверхні (така ж, як і трансляція surfaceBaseTransform).

const sphereOrbitRadius = 3; // Відстань сфери від центру обертання.

const sphereVisualizationScale = 0.5; // Візуальний масштаб сфери.

let soundSourceOrbitTransform = m4.identity(); // Початкова матриця трансформації для сфери.

// Забезпечуємо повну гучність джерела звуку через sourceGainNode.

if (sourceGainNode && sourceGainNode.gain) {

sourceGainNode.gain.setValueAtTime(1.0, audioContext.currentTime);

}

} else {

console.warn("Властивості позиції Panner не повністю визначені, або currentTime дорівнює 0.");

}

}

// Встановлюємо початкову позицію слухача (камери) у світових координатах.

// Слухач знаходиться у світовому початку координат і дивиться вздовж негативної осі Z.

audioContext.listener.positionX.setValueAtTime(0, audioContext.currentTime);

audioContext.listener.positionY.setValueAtTime(0, audioContext.currentTime);

audioContext.listener.positionZ.setValueAtTime(0, audioContext.currentTime);

audioContext.listener.forwardX.setValueAtTime(0, audioContext.currentTime);

audioContext.listener.forwardY.setValueAtTime(0, audioContext.currentTime);

audioContext.listener.forwardZ.setValueAtTime(-1, audioContext.currentTime);

audioContext.listener.upX.setValueAtTime(0, audioContext.currentTime);

audioContext.listener.upY.setValueAtTime(1, audioContext.currentTime);

audioContext.listener.upZ.setValueAtTime(0, audioContext.currentTime);

// Створюємо BiquadFilterNode (фільтр Low-Shelf).

biquadFilter = audioContext.createBiquadFilter();

biquadFilter.type = 'lowshelf'; // Тип фільтра (збільшення/зменшення низьких частот).

updateFilterParameters(); // Встановлюємо початкові параметри фільтра з HTML-вводів.

biquadFilter.gain.value = 0; // Починаємо з вимкненим фільтром (без зміни гучності).