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”

Варіант 21

Виконав студент групи ТР-31мп  
Плаксицький Всеволод

Київ 2024

**Завдання**

* Імплементувати просторове аудіо за допомогою WebAudio HTML5 API, використовуючи код з практичного завдання 2.
* Імплементувати обертання джерела звуку відносно геометричного центру поверхні з використанням дотичного інтерфейсу (смартфону, що використовує сенсори для обробки інформації, щодо положення в просторі). На відміну від попередніх робіт поверхня залишається нерухомою - рухається джерело звуку. Відтворити улюблену пісню в форматі mp3/ogg, з можливістю контролювати місцезнаходження джерела звуку в просторі користувачем.
* Візуалізувати джерело звуку за допомогою сфери.
* Додати звуковий фільтр за варіантом (використовуючи інтерфейс BiquadFilterNode). Додати інтерфейс користувача, який би вмикав/вимикав фільтр. Встановити параметри фільтру відповідно до вподобань.
* Підготувати звіт в цифровому вигляді, який би містив необхідні частини, що сповна описують поставлені задачі та виконану роботу.

**Теоретичні відомості**

Аудіо в Інтернеті до цього моменту було досить примітивним, і його доводилося відтворювати через плагіни, такі як Flash і QuickTime. Введення елемента аудіо в HTML5 стало важливим кроком, оскільки дозволяє відтворювати базове потокове аудіо без використання сторонніх плагінів. Однак цей елемент недостатньо потужний для роботи зі складнішими аудіододатками. Для реалізації складних веб-ігор або інтерактивних програм потрібне інше рішення.

Мета WebAudio API полягає в тому, щоб включити можливості сучасних звукових движків, а також завдання мікшування, обробки та фільтрації аудіо, що використовуються в сучасних настільних програмах для створення звуку. Інтерфейси API були розроблені з урахуванням широкого спектру випадків використання. В ідеалі, система повинна підтримувати будь-який варіант використання, який можна реалізувати за допомогою оптимізованого механізму на C++, керованого через сценарій і запущеного в браузері.

Незважаючи на це, сучасне настільне аудіопрограмне забезпечення може мати дуже розширені можливості, деякі з яких важко або неможливо реалізувати за допомогою цієї системи. Наприклад, Apple Logic Audio підтримує зовнішні MIDI-контролери, довільні плагіни аудіоефектів і синтезаторів, оптимізоване читання/запис аудіофайлів на диск, тісно інтегроване розтягування часу тощо. Тим не менш, запропонована система буде цілком здатна підтримувати широкий спектр досить складних ігор та інтерактивних програм, включаючи музичні.

Це буде гарним доповненням до розширених графічних функцій, які пропонує WebGL. API розроблено таким чином, щоб у майбутньому можна було додати ще більше розширених можливостей.

BiquadFilterNode — це процесор AudioNode, який реалізує поширені фільтри нижчого порядку. Ці фільтри є будівельними блоками для базових регуляторів тембру (баси, середні та високі частоти), графічних еквалайзерів і більш складних фільтрів. Кілька фільтрів BiquadFilterNode можна комбінувати для створення більш складних фільтраційних схем. Параметри фільтра, такі як частота, можна змінювати з часом для динамічної обробки звуку. Кожен BiquadFilterNode можна налаштувати як один із кількох типів фільтрів, наведених у специфікації IDL. Тип фільтра за замовчуванням – «НЧ».

Фільтр низьких частот пропускає частоти нижче граничної частоти та послаблює частоти вище неї. Це стандартний резонансний фільтр низьких частот другого порядку зі спадом 12 дБ/октаву.

Фільтр високих частот є протилежністю фільтру низьких частот. Він пропускає частоти вище граничної частоти та послаблює частоти нижче неї. Це стандартний резонансний фільтр високих частот другого порядку зі спадом 12 дБ/октаву.

Смуговий фільтр пропускає діапазон частот і послаблює частоти нижче та вище цього діапазону. Це смуговий фільтр другого порядку.

Фільтр Lowshelf пропускає всі частоти, але додає підсилення (або ослаблення) нижніх частот. Це фільтр низького рівня другого порядку.

Фільтр Highshelf пропускає всі частоти, але додає підсилення до вищих частот. Це фільтр високої полиці другого порядку, протилежний до фільтру Lowshelf.

Піковий фільтр пропускає всі частоти, але додає підсилення (або ослаблення) до певного діапазону частот.

Режекторний фільтр, також відомий як смуговий або смуговий фільтр, є протилежністю смуговому фільтру. Він пропускає всі частоти, крім певного діапазону частот.

1. **Аспекти імплементації**

В результаті виконання 3 та 4 лабораторних робіт був розроблений матеріальний інтерфейс: додаток, який дозволяє користувачу обертати фігуру відповідно до показів сенсорів смартфона, враховуючи його відносне положення в просторі.

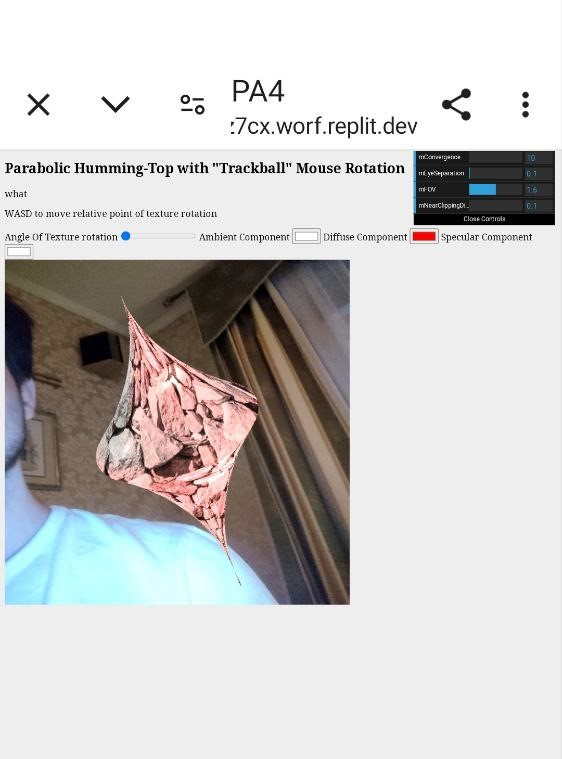


Рисунок 1 - Скріншот інтерфейсу 4 лабораторної роботи

В подальшому цей функціонал дозволить керувати положенням джерела звуку просторового аудіо всередині програми. Першим кроком для впровадження просторового аудіо є створення HTML-елемента <audio>, який містить інформацію про джерело аудіодоріжки. У моєму випадку це audio.mp3. Цей елемент управління дозволить зупиняти та продовжувати відтворення аудіодоріжки.

Після цього за допомогою JavaScript, а саме WebAudio API, був створений об’єкт аудіоконтексту (AudioContext), до якого було під’єднано три основні об’єкти:

* Джерело звуку (MediaElementSource)
* Об’єкт обробки просторового аудіо (Panner)
* Звуковий фільтр (BiquadFilter)

Наступним кроком було встановлено параметри обраного фільтра. Взагалі, BiquadFilter має 3 параметри для налаштування: frequency (частота), Q (ширина смуги), gain (підсилення).

Згідно з варіантом 21 було обрано режекторний фільтр. Його особливість в тому, що для режекторного фільтра frequency - центральна частота, де нанесено виїмку, Q - контролює ширину смуги частот, які послаблюються. Велике значення означає вузьку ширину, gain - Не використовується в цьому типі фільтра.

Джерело звуку, обробка якого здійснюється об’єктом класу Panner, відображається у WebGL контексті у вигляді сфери, яка може переміщуватися. Відповідно змінюються параметри об’єкта Panner, що створює ефект переміщення джерела звуку в стерео режимі відповідно до місцезнаходження сфери в системі координат.

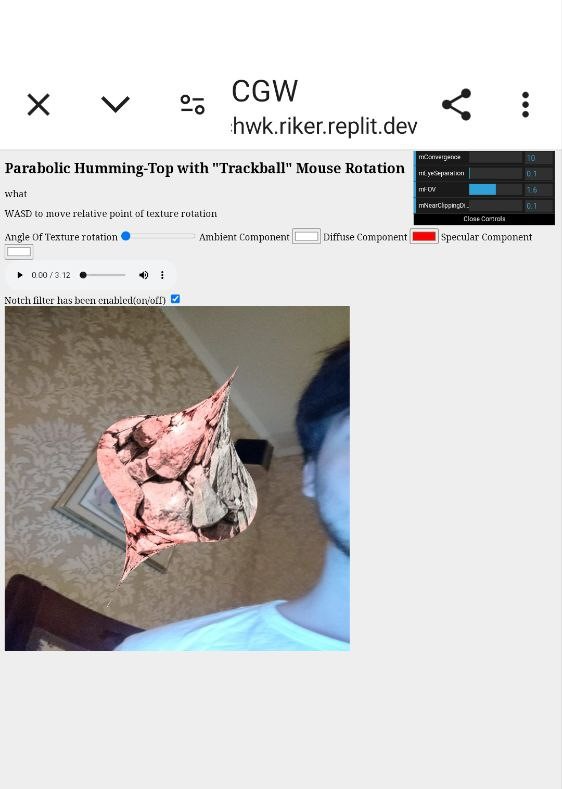


Рисунок 2 - скріншот інтерфейсу РГР

1. **Інструкція користувача**

Для налаштування стерео зображення використовуються 4 повзунці з такими параметрами:

* Convergence (збіжність)
* Eye separation (відстань між очима)
* Field of view (поле зору)
* Near clipping distance (відстань ближньої площини, що відсікає зображену на екрані геометрію)

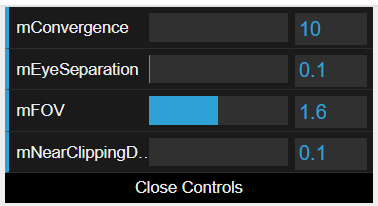


Рисунок 3 - Скрішот повзунців, для налаштування стерео зображення

Згідно до встановлених параметрів стерео зображення фігури виглядає по-різному:

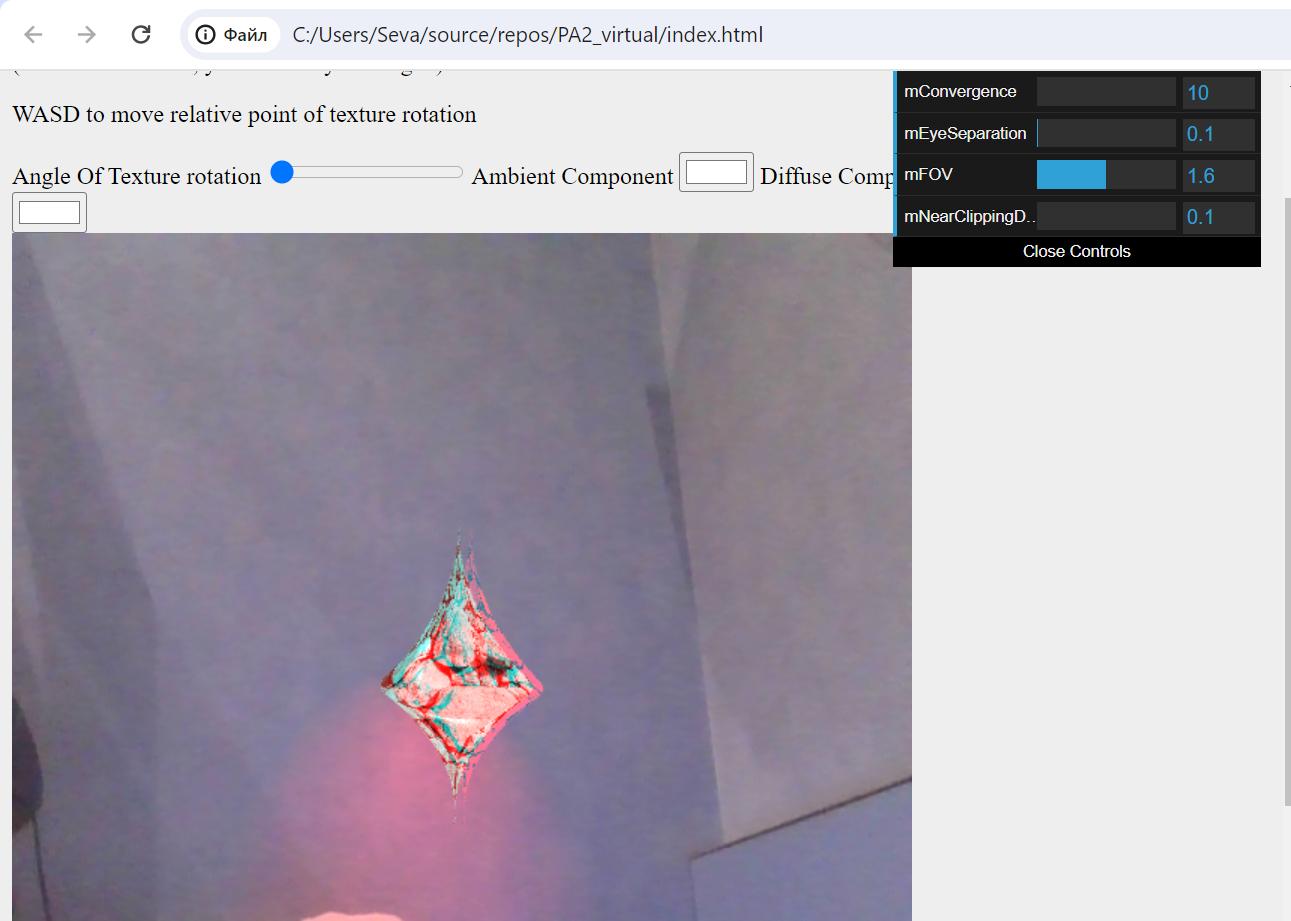


Рисунок 4 - Скріншот фігури з початковими налаштуваннями

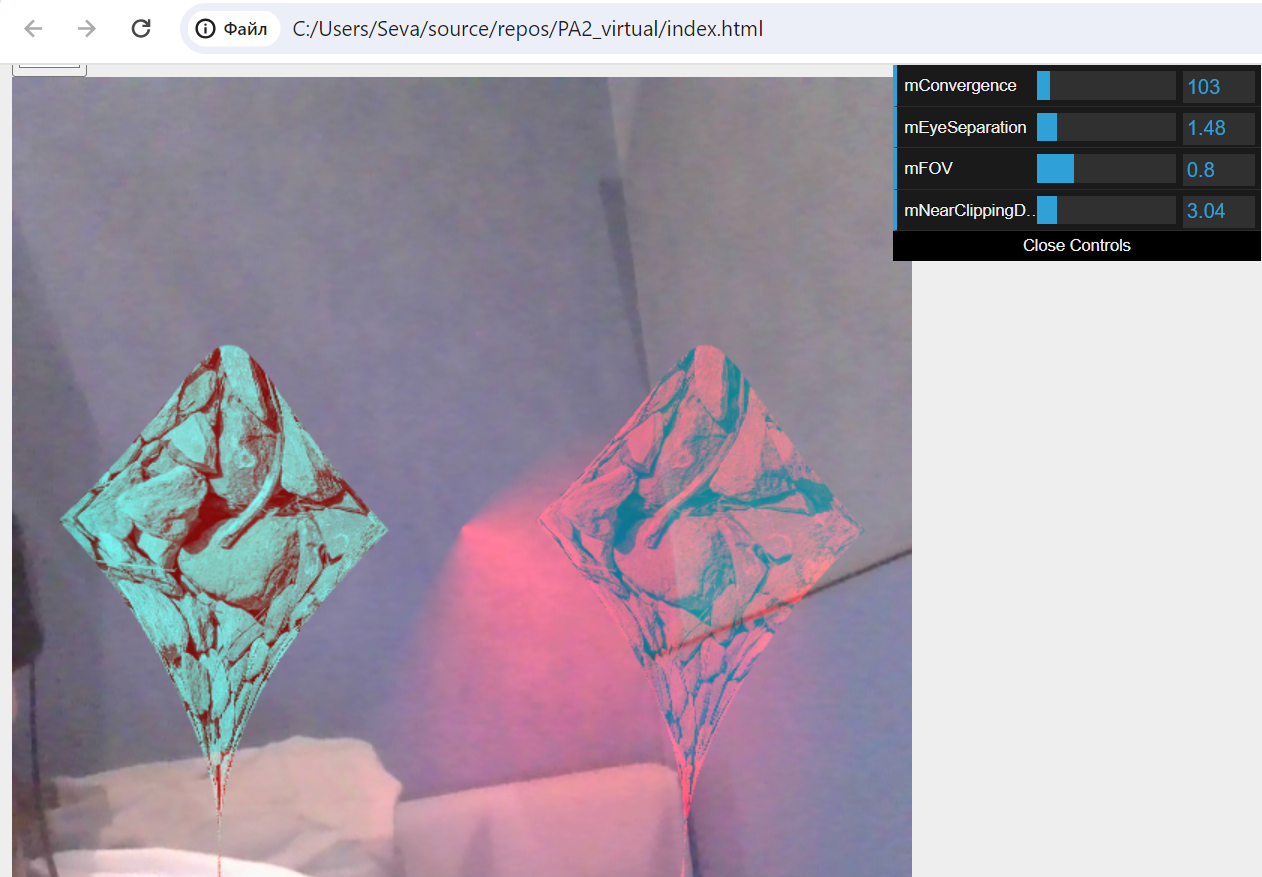


Рисунок 5 - Скріншот фігури зі зміненими налаштуваннями

HTML-елемент audio дозволяє керувати аудіодоріжкою: зупиняти та продовжувати відтворення, перемотувати до потрібного моменту, керувати гучністю. Чекбокс дозволяє вмикати та вимикати звуковий фільтр. Залежно від стану фільтра, можна помітити зміну звучання.

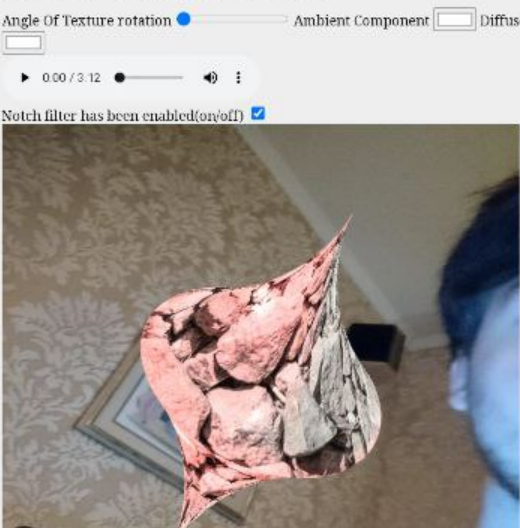


Рисунок 6 - Скріншот з UI-елементом плеєра

1. **Код програми**

let timestamp,

gyroscope,

gyroscopied = { x: 0, y: 0, z: 0, started: false, matrix: m4.identity() },

alpha = 0,

beta = 0,

gamma = 0;

const E = 0.001

const MS2S = 1.0 / 1000.0;

// some constants

function startGyroscope() {

timestamp = Date.now();

gyroscope = new Gyroscope();

gyroscope.addEventListener('reading', () => {

timestamp = Date.now();

gyroscopied.x = gyroscope.x

gyroscopied.y = gyroscope.y

gyroscopied.z = gyroscope.z

gyroscopeToRotationMatrix()

})

gyroscope.start();

gyroscope.started = true

}

function getRotationMatrixFromVector(rotationVector) {

const q1 = rotationVector[0];

const q2 = rotationVector[1];

const q3 = rotationVector[2];

let q0;

if (rotationVector.length >= 4) {

q0 = rotationVector[3];

} else {

q0 = 1 - q1 \* q1 - q2 \* q2 - q3 \* q3;

q0 = q0 > 0 ? Math.sqrt(q0) : 0;

}

const sq\_q1 = 2 \* q1 \* q1;

const sq\_q2 = 2 \* q2 \* q2;

const sq\_q3 = 2 \* q3 \* q3;

const q1\_q2 = 2 \* q1 \* q2;

const q3\_q0 = 2 \* q3 \* q0;

const q1\_q3 = 2 \* q1 \* q3;

const q2\_q0 = 2 \* q2 \* q0;

const q2\_q3 = 2 \* q2 \* q3;

const q1\_q0 = 2 \* q1 \* q0;

let R = [];

R.push(1 - sq\_q2 - sq\_q3);

R.push(q1\_q2 - q3\_q0);

R.push(q1\_q3 + q2\_q0);

R.push(0.0);

R.push(q1\_q2 + q3\_q0);

R.push(1 - sq\_q1 - sq\_q3);

R.push(q2\_q3 - q1\_q0);

R.push(0.0);

R.push(q1\_q3 - q2\_q0);

R.push(q2\_q3 + q1\_q0);

R.push(1 - sq\_q1 - sq\_q2);

R.push(0.0);

R.push(0.0);

R.push(0.0);

R.push(0.0);

R.push(1.0);

return R;

}

function gyroscopeToRotationMatrix() {

if (gyroscope.started) {

let dT = (Date.now() - timestamp) \* MS2S;

let omegaMagnitude = Math.sqrt(

gyroscopied.x \* gyroscopied.x +

gyroscopied.y \* gyroscopied.y +

gyroscopied.z \* gyroscopied.z);

if (omegaMagnitude > E) {

alpha += gyroscopied.x \* dT;

beta += gyroscopied.y \* dT;

gamma += gyroscopied.z \* dT;

alpha = Math.min(Math.max(alpha, -Math.PI \* 0.25), Math.PI \* 0.25)

beta = Math.min(Math.max(beta, -Math.PI \* 0.25), Math.PI \* 0.25)

gamma = Math.min(Math.max(gamma, -Math.PI \* 0.25), Math.PI \* 0.25)

let deltaRotationVector = [];

deltaRotationVector.push(alpha);

deltaRotationVector.push(beta);

deltaRotationVector.push(gamma);

gyroscopied.matrix = getRotationMatrixFromVector(deltaRotationVector)

timestamp = Date.now();

}

}

}