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 27

Виконав студент групи ТР-31мп  
Уваровський Олександр Дмитрович

Київ 2024

1. **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 – піковий фільтр (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.

1. **Теоретичні відомості**

**WebAudio API**

Аудіо в Інтернеті до цього моменту було досить примітивним і донедавна доставлялося через плагіни, такі як Flash і QuickTime. Введення елемента аудіо в HTML5 є важливим, оскільки дозволяє відтворювати базове потокове аудіо. Однак, його можливостей недостатньо для роботи зі складнішими аудіододатками. Для складних веб-ігор або інтерактивних програм потрібне інше рішення. Мета цієї специфікації полягає в тому, щоб включити можливості сучасних ігрових звукових движків, а також завдання мікшування, обробки та фільтрації, які є в сучасних настільних програмах створення звуку.

Інтерфейси API були розроблені з урахуванням широкого спектру випадків використання. В ідеалі, він повинен підтримувати будь-який варіант використання, який можна розумно реалізувати за допомогою оптимізованого механізму C++, керованого через сценарій і запущеного в браузері. Проте сучасне аудіопрограмне забезпечення для настільних комп’ютерів 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, високооптимізоване читання/запис аудіофайлів прямо на диск, тісно інтегроване розтягування часу тощо. Однак запропонована система буде здатна підтримувати широкий спектр досить складних ігор та інтерактивних програм, включаючи музичні. Вона може бути гарним доповненням до розширених графічних функцій, які пропонує WebGL. API розроблено таким чином, щоб пізніше можна було додати розширені можливості.

**BiquadFilterNode**

BiquadFilterNode — це процесор AudioNode, який реалізує дуже поширені фільтри нижчого порядку. Ці фільтри є будівельними блоками базових регуляторів тембру (баси, середні та високі частоти), графічних еквалайзерів та більш розширених фільтрів. Кілька фільтрів BiquadFilterNode можна комбінувати для створення більш складних фільтрів. Параметри фільтра, такі як частота, можна змінювати з часом для розгортки фільтра тощо. Кожен BiquadFilterNode можна налаштувати як один із кількох загальних типів фільтрів, як показано в IDL нижче. Тип фільтра за замовчуванням — «НЧ».

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

Фільтр високих частот є протилежністю фільтру низьких частот. Частоти вище граничної частоти пропускаються, але частоти нижче граничної послаблюються. Він реалізує 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 частот. Він реалізує смуговий фільтр другого порядку.

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

Фільтр Highshelf є протилежністю фільтру Lowshelf і пропускає всі частоти, але додає підсилення до вищих частот. Він реалізує фільтр високої полиці другого порядку.

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

Режекторний фільтр (також відомий як смуговий або смуговий фільтр) є протилежністю смуговому фільтру. Він дозволяє пропускати всі частоти, крім набору частот.

1. **Аспекти імплементації**

В результаті виконання 3 та 4 лабораторних робіт було створено матеріальний інтерфейс: смартфон, який дозволяє користувачу обертати фігуру відповідно до показів сенсорів смартфона, залежно від його положення в просторі.

A person taking a selfie

Description automatically generated

Рисунок 3.1 - скріншот 3 лабораторної

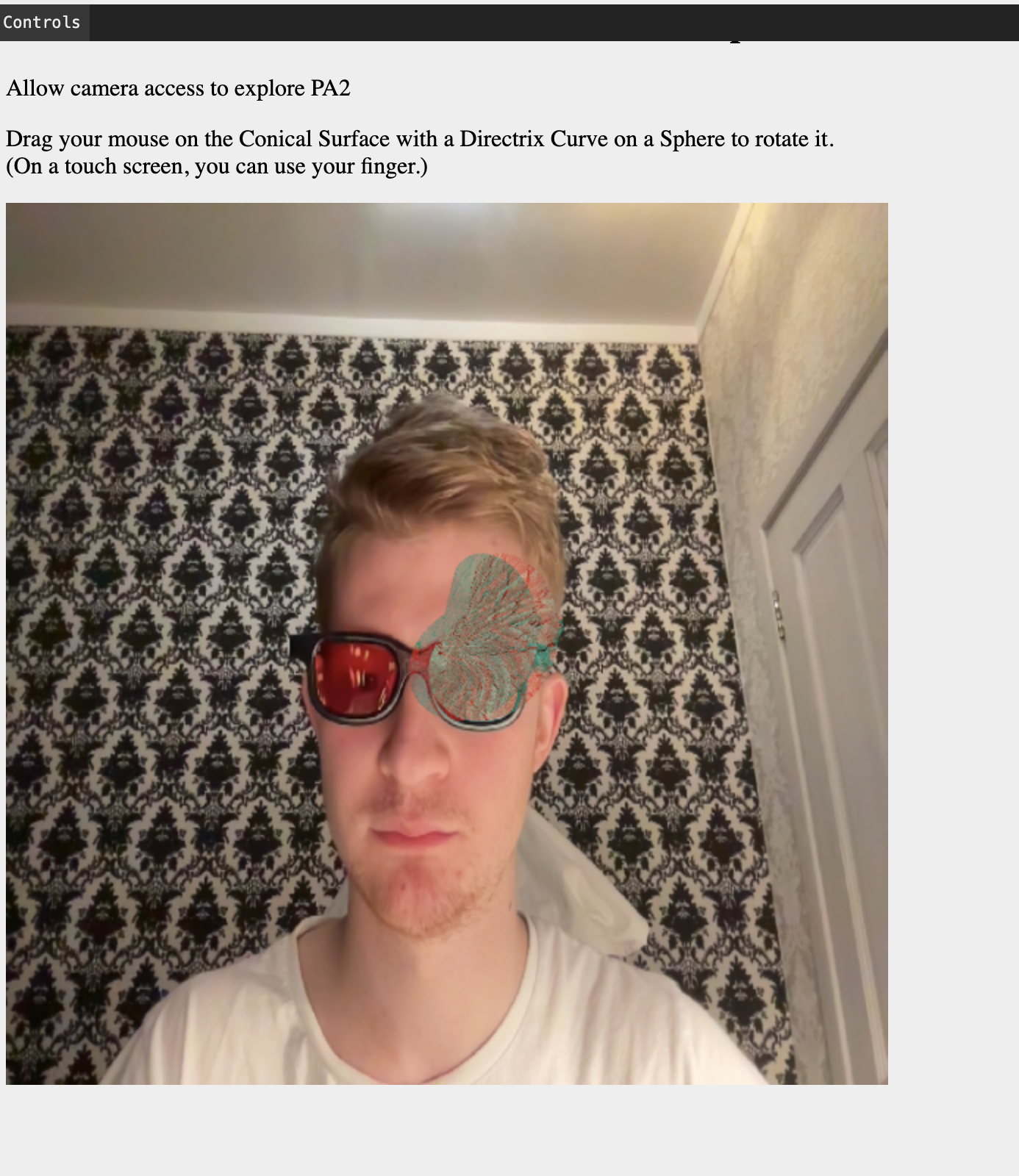


Рисунок 3.1 - скріншот 2 лабораторної

У майбутньому 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 аудіо-доріжки, в моєму випадку це "02\_\_Master\_Of\_Puppets\_64kb.mp3" у форматі mp3. Цей елемент керування дозволить зупиняти та продовжувати відтворення аудіо-доріжки. Після цього за допомогою JavaScript, а саме WebAudio API, було створено об'єкт аудіоконтексту (AudioContext), для якого було створено та під'єднано три основних об'єкти:

- джерело звуку (MediaElementSource)

- об'єкт обробки просторового аудіо (Panner)

- звуковий фільтр (BiquadFilter)

Згідно з варіантом було обрано піковий фільтр.

Наступним кроком було встановлено параметри обраного фільтра. Взагалі, BiquadFilter має 3 параметри для налаштування:

* frequency (частота)
* Q (ширина смуги)
* gain (підсилення)

Особливість пікового фільтра полягає в тому, що frequency визначає центральну частоту, на якій застосовується посилення. Параметр Q контролює ширину смуги частот, які підсилюються: велике значення Q означає вузьку ширину смуги. Параметр gain визначає рівень підсилення у дБ, яке буде застосовано. Якщо значення gain від'ємне, частоти послаблюються.

Джерело звуку, оброблене в просторі за допомогою об'єкта класу Panner, зображено в контексті WebGL у вигляді сфери, яка може переміщуватися. Це відповідно змінює параметри об'єкта Panner, створюючи ефект переміщення джерела звуку під час прослуховування стерео аудіо, залежно від місцезнаходження сфери в системі координат.



Рисунок 3.3 - скріншот РГР

1. **Інструкція користувача**

Для налаштування стерео зображення використовуються 4 слайдери, в який можна налаштувати наступні параметри:

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

A screenshot of a phone

Description automatically generated

Рисунок 4.1 - скрішот слайдерів

Згідно до встановлених параметрів стерео зображення фігури виглядає дещо інакше

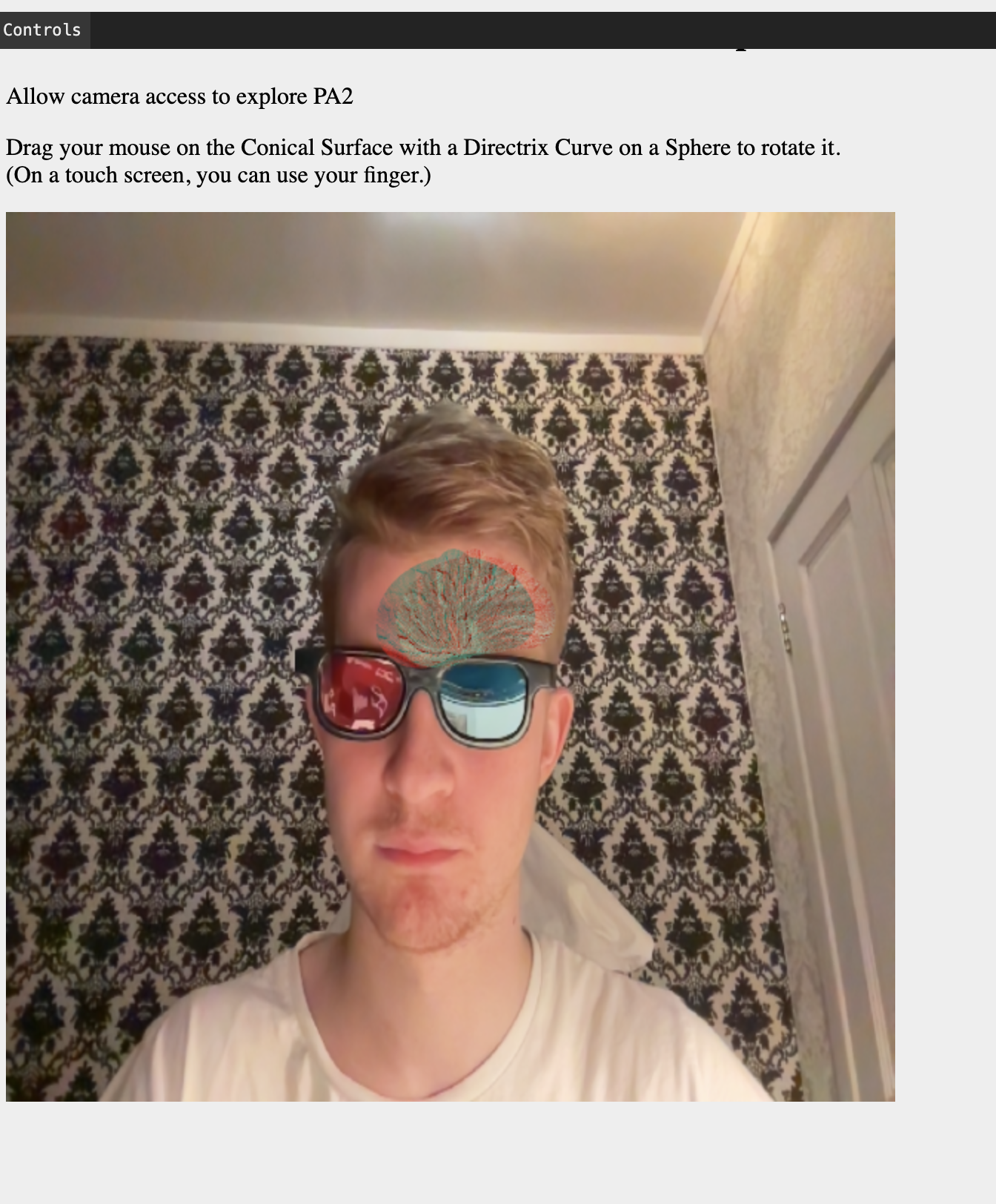


Рисунок 4.2 - скріншот фігури зі зміненими налаштуваннями

Вище згаданий HTML-елемент audio дозволяє керувати аудіо-доіжкою, а саме зупиняти та продовжувати відтворення, перемотувати на потрібний час, керувати гучністю.

A screenshot of a phone

Description automatically generated

Рисунок 4.4 - скріншот з UI-елементом плеєра

“Чекбокс” дозволяє керувати станом звукового фільтру: вмикати та вимикати його. В залежності від стану фільтру можна помітити зміну звучання.

1. **Код програми**
2. let context;
3. let audioPlayer;
4. let source;
5. let lowpassFilter;
6. let panner;
7. function setupAudio() {
8. audioPlayer = document.getElementById('audioID');
9. audioPlayer.addEventListener('play', () => {
10. if (!context) {
11. context = new AudioContext();
12. source = context.createMediaElementSource(audioPlayer);
13. panner = context.createPanner();
14. lowpassFilter = context.createBiquadFilter();
15. source.connect(panner);
16. panner.connect(lowpassFilter);
17. lowpassFilter.connect(context.destination);
18. lowpassFilter.type = 'lowpass';
19. lowpassFilter.Q.value = 1;
20. lowpassFilter.gain.value = 11;
21. context.resume();
22. }
23. })
24. audioPlayer.addEventListener('pause', () => {
25. console.log('pause');
26. context.resume();
27. })
28. }
29. function initAudio() {
30. setupAudio();
31. const radioButton = document.getElementById('checker');
32. radioButton.addEventListener('change', function() {
33. if (radioButton.checked) {
34. panner.disconnect();
35. panner.connect(lowpassFilter);
36. lowpassFilter.connect(context.destination);
37. } else {
38. panner.disconnect();
39. panner.connect(context.destination);
40. }
41. });
42. audioPlayer.play();
43. }

function CreateSphereData() {

let vertexList = [];

let u = 0,

v = 0;

while (u < Math.PI \* 2) {

while (v < Math.PI) {

let v1 = getSphereVertex(u, v);

let v2 = getSphereVertex(u + 0.1, v);

let v3 = getSphereVertex(u, v + 0.1);

let v4 = getSphereVertex(u + 0.1, v + 0.1);

vertexList.push(v1.x, v1.y, v1.z);

vertexList.push(v2.x, v2.y, v2.z);

vertexList.push(v3.x, v3.y, v3.z);

vertexList.push(v3.x, v3.y, v3.z);

vertexList.push(v2.x, v2.y, v2.z);

vertexList.push(v4.x, v4.y, v4.z);

v += 0.1;

}

v = 0;

u += 0.1;

}

return vertexList

}

const radius = 0.2;

function getSphereVertex(long, lat) {

return {

x: radius \* Math.cos(long) \* Math.sin(lat),

y: radius \* Math.sin(long) \* Math.sin(lat),

z: radius \* Math.cos(lat)

}

}