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 4

Виконав студент групи ТР-32мп  
Гасанов Ельдар Ігорович

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 (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 довгий час було досить примітивним і вимагало використання плагінів, таких як Flash і QuickTime, для відтворення. Введення аудіоелемента в HTML5 стало значним кроком вперед, оскільки дозволяє базове потокове аудіо. Однак цього недостатньо для складніших аудіододатків. Для більш комплексних веб-ігор або інтерактивних програм потрібне інше рішення. Ця специфікація має на меті включити можливості сучасних ігрових звукових рушіїв, а також задачі мікшування, обробки та фільтрації, притаманні сучасним настільним програмам для створення звуку.

API було розроблено з урахуванням широкого спектру випадків використання [webaudio-usecases]. Ідеально він повинен підтримувати будь-який сценарій використання, який можна розумно реалізувати за допомогою оптимізованого механізму на C++, що керується через сценарії та працює у браузері. Тим не менш, сучасне настільне аудіопрограмне забезпечення може мати дуже розширені можливості, деякі з яких було б важко або неможливо створити за допомогою цієї системи. Apple Logic Audio є прикладом такої програми, яка підтримує зовнішні MIDI-контролери, довільні аудіоефекти та плагіни синтезаторів, високооптимізоване читання/запис аудіофайлів безпосередньо на диск, тісно інтегроване розтягнення часу тощо. Незважаючи на це, запропонована система буде цілком здатна підтримувати широкий спектр досить складних ігор та інтерактивних програм, включаючи музичні. Вона стане гарним доповненням до розширених графічних функцій, які пропонує WebGL. API розроблено так, щоб у майбутньому можна було додати розширені можливості.

**BiquadFilterNode**

BiquadFilterNode — це процесор AudioNode, який реалізує дуже поширені фільтри нижчого порядку.

Фільтри нижчого порядку є основними елементами для побудови базових регуляторів тембру (баси, середні та високі частоти), графічних еквалайзерів та більш складних фільтрів. Кілька фільтрів BiquadFilterNode можна об'єднати для створення складніших фільтруючих систем. Параметри фільтра, такі як частота, можуть змінюватися з часом для створення динамічних ефектів. Кожен BiquadFilterNode можна налаштувати на один із кількох типів фільтрів, як показано в IDL нижче. Тип фільтра за замовчуванням — низькочастотний (НЧ).

Фільтр низьких частот пропускає частоти, нижчі за граничну, і послаблює частоти, вищі за граничну. Це стандартний резонансний низькочастотний фільтр другого порядку з ослабленням 12 дБ на октаву.

Фільтр високих частот діє протилежно до низькочастотного фільтра. Він пропускає частоти, вищі за граничну, і послаблює частоти, нижчі за граничну. Це стандартний резонансний високочастотний фільтр другого порядку з ослабленням 12 дБ на октаву.

Смуговий фільтр пропускає певний діапазон частот і послаблює частоти, нижчі та вищі за цей діапазон. Це смуговий фільтр другого порядку.

Фільтр Lowshelf пропускає всі частоти, але підсилює (або послаблює) нижчі частоти. Це фільтр низького рівня другого порядку.

Фільтр Highshelf діє навпаки: він пропускає всі частоти, але підсилює (або послаблює) вищі частоти. Це фільтр високого рівня другого порядку.

Піковий фільтр пропускає всі частоти, але підсилює (або послаблює) певний діапазон частот.

Режекторний фільтр (або смуговий фільтр) пропускає всі частоти, крім певного діапазону частот, який він ослаблює.

1. **Аспекти імплементації**

Реалізовано кругове обертання джерела звуку навколо геометричного центру поверхні протягом певного часу. Джерело звуку відображено у формі сферичної геометрії.

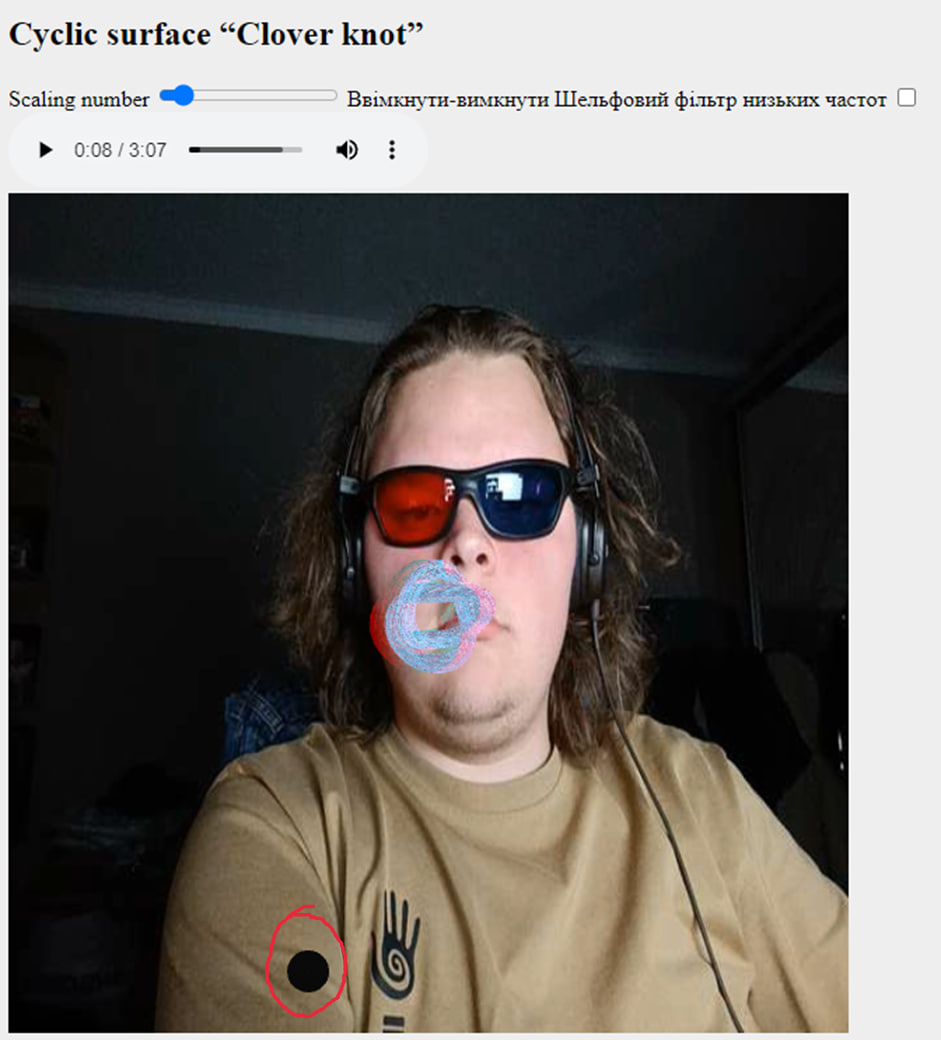


Рисунок 3.1

В подальшому цей функціонал дозволить керувати положенням джерела звуку просторового аудіо всередині програми.

Першим кроком для імплементації просторового аудіо є створення HTML-елементу <audio>, який містить інформацію про джерело аудіо-доріжки в моєму випадку це «sadsvit.mp3» в форматі mp3. Використана пісня «Силуети». Цей елемент керування дозволить зупиняти та продовжувати відтворення аудіо-доріжки. Після цього за допомогою JavaScript, а саме WebAudio API було створено обʼєкт аудіоконтексту (AudioContext), для якого було створено та підʼєднано 3 основних обʼєкти:

* джерело звуку (MediaElementSource)
* обʼєкт обробки просторового аудіо (Panner)
* звуковий фільтр (BiquadFilter)

Згідно з варіантом було обрано Шелфовий фільтр низьких частот.

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

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

Особливість Шелфового фільтру низьких частот в тому, що

frequency - верхня межа частот, на яких застосовується підсилення.

Q - не використовується в цьому типі фільтра.

gain - підсилення, у дБ, яке буде застосовано. Якщо значення від'ємне, частоти послаблюються.

Джерело звуку, яке обробляється об'єктом класу Panner, зображено у WebGL у вигляді сфери, що може переміщуватися. Це дозволяє змінювати параметри об'єкту Panner, створюючи ефект переміщення джерела звуку при прослуховуванні стерео аудіо, відповідно до положення сфери в системі координат.

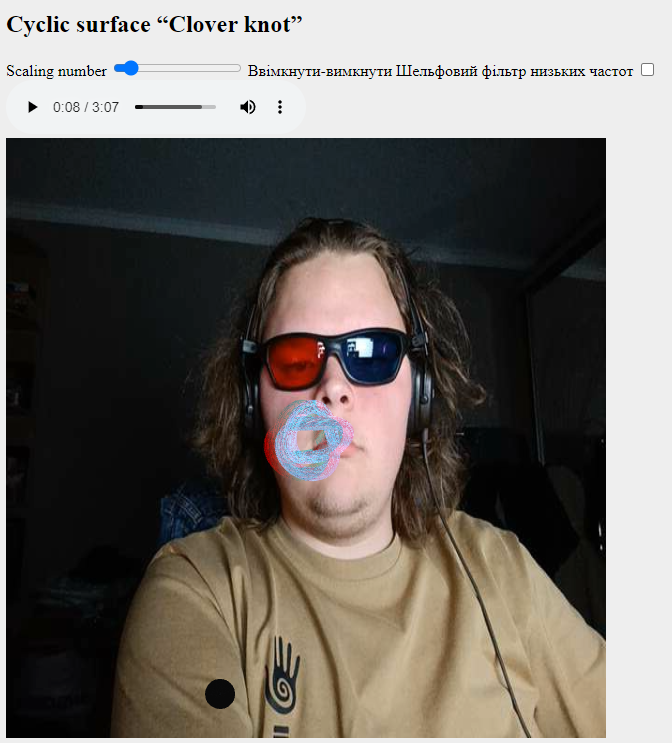


Рисунок 3.2

1. **Інструкція користувача**

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

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

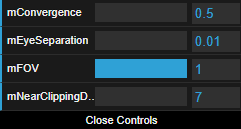


Рисунок 4.1

Згідно до встановлених параметрів стерео зображення фігури виглядає дещо інакше

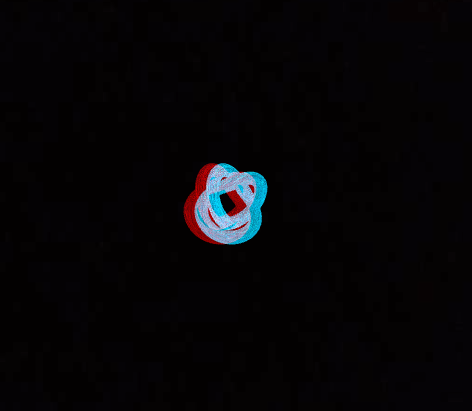


Рисунок 4.2

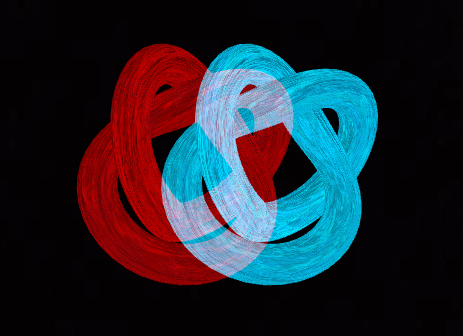


Рисунок 4.3

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



Рисунок 4.4

«Чекбокс» дозволяє керувати станом звукового фільтру: вмикати та вимикати його. В залежності від стану фільтру можна помітити зміну звучання. Шельфовий фільтр низьких частот підсилює всі частоти до певної заданої межі.



Рисунок 4.5

1. **Код програми**

let context, source, panner, biquadFilter;

// Audio context initialization

function addEventListeners() {

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

if (!context) {

context = new AudioContext();

source = context.createMediaElementSource(audio);

panner = context.createPanner();

biquadFilter = context.createBiquadFilter();

source.connect(panner);

biquadFilter.type = 'lowshelf';

biquadFilter.gain.value = 11;

biquadFilter.frequency.value = 1111;

context.resume();

}

})

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

console.log('pause');

context.resume();

})

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

if (chck.checked) {

panner.disconnect();

panner.connect(biquadFilter);

biquadFilter.connect(context.destination);

} else {

panner.disconnect();

panner.connect(context.destination);

}

});

}

function doTouchStart(evt) {

if (evt.touches.length != 1) {

doTouchCancel();

return;

}

evt.preventDefault();

var r = canvas.getBoundingClientRect();

prevx = evt.touches[0].clientX - r.left;

prevy = evt.touches[0].clientY - r.top;

canvas.addEventListener("touchmove", doTouchMove, false);

canvas.addEventListener("touchend", doTouchEnd, false);

canvas.addEventListener("touchcancel", doTouchCancel, false);

touchStarted = true;

centerX = canvas.width/2;

centerY = canvas.height/2;

var radius = Math.min(centerX,centerY);

radius2 = radius\*radius;

}

function doTouchMove(evt) {

if (evt.touches.length != 1 || !touchStarted) {

doTouchCancel();

return;

}

evt.preventDefault();

var r = canvas.getBoundingClientRect();

var x = evt.touches[0].clientX - r.left;

var y = evt.touches[0].clientY - r.top;

var ray1 = toRay(prevx,prevy);

var ray2 = toRay(x,y);

applyTransvection(ray1,ray2);

prevx = x;

prevy = y;

if (callback) {

callback();

}

}

function doTouchEnd(evt) {

doTouchCancel();

}

function initGL() {

let prog = createProgram(gl, vertexShaderSource, fragmentShaderSource);

shProgram = new ShaderProgram('Basic', prog);

shProgram.Use();

shProgram.iAttribVertex = gl.getAttribLocation(prog, "vertex");

shProgram.iAttribNormal = gl.getAttribLocation(prog, "normal");

shProgram.iAttribTexture = gl.getAttribLocation(prog, "texture");

shProgram.iModelViewProjectionMatrix = gl.getUniformLocation(prog, "ModelViewProjectionMatrix");

shProgram.iNormalMatrix = gl.getUniformLocation(prog, "NormalMatrix");

shProgram.iColor = gl.getUniformLocation(prog, "color");

shProgram.iTime = gl.getUniformLocation(prog, "t");

shProgram.iTranslateLight = gl.getUniformLocation(prog, "translateLight");

shProgram.iTMU = gl.getUniformLocation(prog, 'tmu');

shProgram.iPointOfScaling = gl.getUniformLocation(prog, 'pointOfScaling');

shProgram.iPointPosition = gl.getUniformLocation(prog, 'pointPos');

shProgram.iScalingNumber = gl.getUniformLocation(prog, 'scalingNumber');

surface = new Model('Surface');

lightSurf = new Model('Surface');

let bufferDataCK = cloverKnotVerts(100, 100, 2, 0.5)

surface.BufferData(bufferDataCK[0]);

surface.NormalBufferData(bufferDataCK[1]);

surface.TextureBufferData(bufferDataCK[2]);

let bufferDataLS = CreateSphereData();

lightSurf.BufferData(bufferDataLS[0])

lightSurf.NormalBufferData(bufferDataLS[1])

lightSurf.TextureBufferData(bufferDataLS[1])

gl.enable(gl.DEPTH\_TEST);

}