МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
КАФЕДРА ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

МЕТОДИ СИНТЕЗУ ВІРТУАЛЬНОЇ РЕАЛЬНОСТІ

Розрахунково-графічна робота
Варіант №4

Виконав:
студент 1-го курсу НН ІАТЕ
групи ТР-31мп
Вербіцький Євген Степанович

Перевірив: Демчишин А. А.

Київ – 2024

ЗАВДАННЯ

1. Повторно використати код з практичного завдання №2;
2. Реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні за допомогою матеріального інтерфейсу (поверхня при цьому залишається нерухомою, а джерело звуку рухається). Відтворити улюблену пісню у форматі mp3/ogg, при цьому просторове положення джерела звуку контролюється користувачем;
3. Візуалізувати положення джерела звуку за допомогою сфери;
4. Додати звуковий фільтр (використати інтерфейс BiquadFilterNode): шелфовий фільтр низьких частот. Додати елемент з прапорцем, який вмикає або вимикає фільтр. Налаштувати параметри фільтра на свій смак.

ТЕОРЕТИЧНІ ОСНОВИ

WebGL (Web Graphics Library) – це технологія, яка дозволяє створювати 3D-графіку в браузері без необхідності встановлення додаткових плагінів чи розширень. Вона базується на OpenGL ES (Embedded Systems), із рядом доповнень для взаємодії з елементами веб-сторінки. За допомогою WebGL розробники можуть створювати візуально багаті та інтерактивні веб-додатки, ігри та візуалізації, використовуючи високоякісну 3D-графіку.

WebAudio HTML5 API – це інтерфейс програмування додатків, який дозволяє розробникам створювати, обробляти і відтворювати аудіо у веб-браузерах. Розглянемо ключові аспекти WebAudio API.

AudioContext є основним об'єктом у Web Audio API, призначеним для роботи зі звуком у веб-додатках. Він дозволяє ініціалізувати аудіо середовище, створювати та зв'язувати аудіо вузли, а також керувати аудіо ланцюгами для формування складних звукових ефектів. Серед можливостей AudioContext — створення джерел звуку (осцилятори, аудіо буферні джерела), обробка звуку через різні ефекти (фільтри, компресори, реверберatори), контроль гучності за допомогою вузлів Gain, аналіз звуку за допомогою аналізаторів, планування та синхронізація аудіо подій, завантаження та декодування аудіо файлів, а також динамічні зміни параметрів аудіо обробки в режимі реального часу. AudioContext забезпечує всі необхідні інструменти для створення інтерактивного і захоплюючого звукового досвіду на веб-сторінках.

Audio Sources у Web Audio API використовуються для представлення аудіо джерел. Вони є початковими елементами аудіо-графа і постачають аудіо-дані, які потім обробляються та відтворюються. Після того, як аудіо-сигнал пройшов крізь різні етапи обробки, він може бути підключений до аудіо-призначення (наприклад, AudioContext.destination), де відтворюється на динаміки або записується у файл.

WebAudio API дозволяє точно синхронізувати аудіо події, забезпечуючи плавне відтворення та обробку аудіо. Окрім цього, наявна підтримка тривимірного звуку, що дозволяє розташовувати джерела звуку в тривимірному

просторі для створення більш реалістичних аудіо ефектів, для чого використовується Panner.

PannerNode в Web Audio API використовується для позиціонування звуку в тривимірному просторі, створюючи ефекти просторового звуку. Він змінює стереопозицію аудіо сигналу, створюючи ілюзію звуку, що виходить з певної точки. PannerNode дозволяє налаштовувати положення та напрямок джерела звуку відносно слухача, використовуючи різні моделі позиціонування для зміни гучності залежно від відстані (лінійна, зворотна, експоненційна). Також можна налаштувати конус орієнтації, що визначає, як звук затухає залежно від напряму. Це дозволяє створювати реалістичні звукові сцени в веб-додатках.

Шелфовий фільтр низьких частот (Low Shelf Filter) — це тип еквалайзера, який регулює амплітуду частот нижче певного порогового значення, званого частотою зрізу. Частота зрізу є точкою, нижче якої всі частоти будуть посилюватися або ослаблюватися. Підсилення або ослаблення визначається параметром gain: позитивне значення означає підсилення (буст), а негативне — ослаблення (кат). Шелфовий фільтр створює «полицю» на графіку амплітудно-частотної характеристики (АЧХ), де всі частоти нижче частоти зрізу зазнають рівномірного підйому або спаду в амплітуді. Основна функція цього фільтра полягає в коригуванні низькочастотної складової аудіосигналу, і він часто використовується для підсилення басів, щоб додати тепла або потужності звуку, або для ослаблення басів, щоб зменшити гул або видалити небажані низькочастотні шуми.

РЕАЛІЗАЦІЯ

В файлі index.html додаємо атрибути, які будуть відповідати за роботу з аудіофайлами:

```
<fieldset>
  <legend>Audio</legend>
  <label for="isFilterOn">Enable filter:</label>
  <input type="checkbox" id="isFilterOn" name="isFilterOn">
  <fieldset>
    <legend>Filter gain</legend>
    <input type="range" id="gain" min="-10" max="10" step="0.5" value="0" onchange="getFilter()">
  </fieldset>
  <audio id="audio" controls loop crossorigin="anonymous">
    <source src="audiofile.mp3" type="audio/mpeg" />
    Your browser does not support the audio element.
  </audio>
</fieldset>
```

Чекбокс <input type="checkbox"> з id "isFilterOn" дозволяє увімкнути або вимкнути фільтр звуку. Поле <input type="range"> з id "gain" дозволяє користувачу вибирати значення затухання фільтра у діапазоні від -10 до 10. Аудіофайл вказаний за допомогою елементу <audio>. Цей елемент містить контроли аудіо, які дозволяють користувачу керувати відтворенням (play/pause), звуком та прокруткою файлу.

Далі наведемо налаштування аудіо. Функція, що відповідає за створення аудіо контексту та підключення аудіо файлу до нього виглядає наступним чином:

```
function createAudio() {
  audio = document.getElementById("audio");
  audio.addEventListener("pause", () => {
    audioContext.resume();
  });
  audio.addEventListener("play", () => {
    if (!audioContext) {
      audioContext = new (window.AudioContext || window.webkitAudioContext)();
      audioSource = audioContext.createMediaElementSource(audio);
      getPanner();
      getFilter();
      audioSource.connect(audioPanner);
      audioFilter.connect(audioContext.destination);
```

```
            audioContext.resume();
            audio.play();
        }
    });
}
```

В функції getPanner() створюємо та налаштовуємо PannerNode:

```
function getPanner() {
    audioPanner = audioContext.createPanner();
    audioPanner.panningModel = "HRTF";
    audioPanner.distanceModel = "linear";
}
```

В функції getFilter() створюємо шелфовий фільтр низьких частот, налаштовуємо його. Також, налаштовуємо чекбокс для включення та виключення фільтру:

```
function getFilter() {
    if (!audioFilter) {
        audioFilter = audioContext.createBiquadFilter();
        audioFilter.type = 'lowshelf';
        audioFilter.frequency.value = 1000;
    }
    audioFilter.gain.value = document.getElementById("gain").value;
    isFilterOn = document.getElementById("isFilterOn");
    isFilterOn.addEventListener("change", function () {
        if (isFilterOn.checked) {
            audioPanner.disconnect();
            audioPanner.connect(audioFilter);
            audioFilter.connect(audioContext.destination);
        } else {
            audioPanner.disconnect();
            audioPanner.connect(audioContext.destination);
        }
    });
}
```

Для візуалізації джерела звуку створюємо сферу. Точки для неї розраховуються наступним чином:

```
function createSphereData() {
    let radius = 0.1;
    let vertexList = [];
    let step = 10;
    let max = 360;
    for (let u = 0; u <= max; u += step) {
```

```
    for (let v = 0; v <= max; v += step) {
      let x1 = sphereCenter.x + (radius * Math.cos(deg2rad(u)) * Math.sin(deg2rad(v)));
      let y1 = sphereCenter.y + (radius * Math.sin(deg2rad(u)) * Math.sin(deg2rad(v)));
      let z1 = sphereCenter.z + (radius * Math.cos(deg2rad(v)));
      let x2 = sphereCenter.x + (radius * Math.cos(deg2rad(u + step)) * Math.sin(deg2rad(v + step)));
      let y2 = sphereCenter.y + (radius * Math.sin(deg2rad(u + step)) * Math.sin(deg2rad(v + step)));
      let z2 = sphereCenter.z + (radius * Math.cos(deg2rad(v + step)))
      vertexList.push(x1, y1, z1);
      vertexList.push(x2, y2, z2);
    }
  }
  return vertexList;
}
```

Рендириться сфера в функції draw(). Положення сфери та джерела звуку змінюється за допомогою акселерометра. Так переміщуємо сферу, знаходячи для неї нові точки після зміни центру:

```
function readAccelerometer() {
  sensor = new Accelerometer({ frequency: 60 });
  sensor.addEventListener("reading", () => {
    sphereCenter.x = sensor.x
    sphereCenter.y = sensor.y
    sphereCenter.z = sensor.z
    sphere.BufferDataSphere(createSphereData());
  });
  sensor.start();
}
```

Джерело звуку переміщуємо за допомогою Panner:

```
  if (audioPanner) {
    audioPanner.setPosition(
      sphereCenter.x * 0.7,
      sphereCenter.y * 0.7,
      sphereCenter.z
    );
  }
```

ІНСТРУКЦІЯ КОРИСТУВАЧА

Після запуску програми бачимо головний інтерфейс (рис. 1). На ньому представлене відео з веб-камери (фронтальної камери на телефоні), зображена на поверхня та сфера, що візуалізує джерело звуку аудіо файлу. З правого боку панель, де ми можемо в реальному часу змінювати різні параметри.

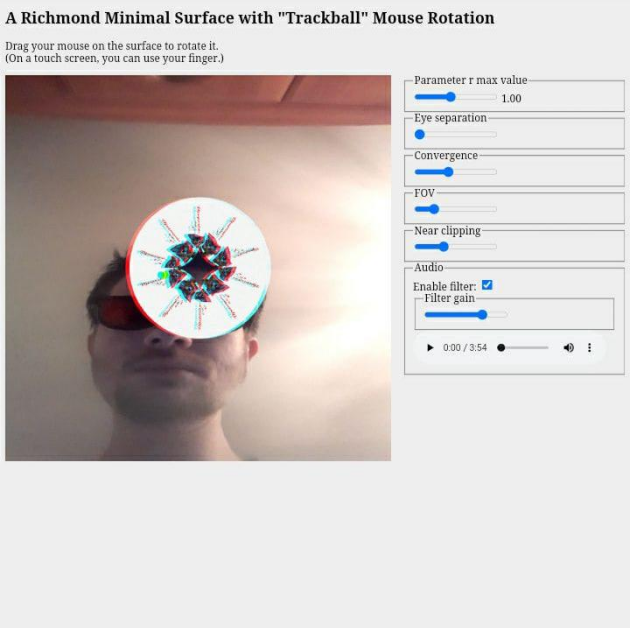

Рисунок 1. – Інтерфейс програми

В блоці Audio можемо вмикати та вимикати аудіо файл, перемотувати його, змінювати гучність, а також вмикати та вимикати фільтр та змінювати його значення gain.

При повороті телефону зміщується джерело звуку, що відображається зеленою сферою (рис. 2).

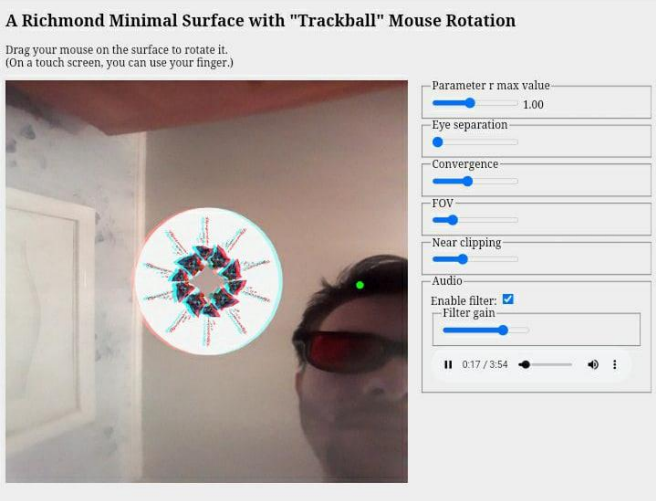

Рисунок 2. – Зміщення джерела звуку при повороті телефону

Використовуючи чекбокс можемо вимкнути фільтр (рис. 3).

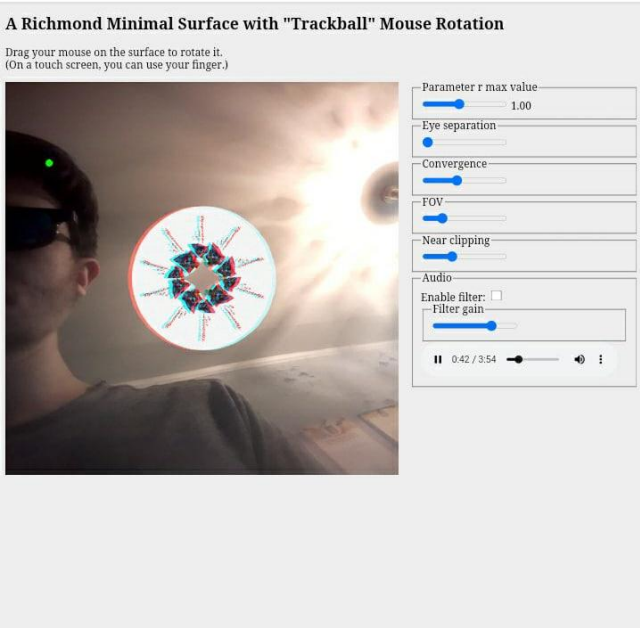

Рисунок 3. – Вимкнення фільтру

Окрім цього, програма дозволяє змінювати максимальне значення параметру r, яке впливає на вигляд поверхні, а також надає можливість змінювати масштабування, а також керувати параметрами для анагліфічного стереозображення.

ВИХІДНИЙ КОД

```
let audio = null;
let audioContext;
let audioSource;
let audioPanner;
let audioFilter;
let isFilterOn = false;
let sphere;
let sphereCenter = { x: 0, y: 0, z: 0 };
let sensor;
let reading = { x: 0, y: 0, z: 0 };

function deg2rad(angle) {
    return angle * Math.PI / 180;
}
// Constructor
function Model(name) {

      ...

    this.BufferDataSphere = function (vertices) {
        gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STREAM_DRAW);
        this.count = vertices.length / 3;
    }

      ...

    this.DrawSphere = function () {
        gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
        gl.vertexAttribPointer(shProgram.iAttribVertex, 3, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(shProgram.iAttribVertex);
        gl.drawArrays(gl.TRIANGLE_STRIP, 0, this.count);
    }
}
function draw() {
      ...

    if (audioPanner) {
        audioPanner.setPosition(
            sphereCenter.x * 0.7,
            sphereCenter.y * 0.7,
            sphereCenter.z
        );
    }
      ...

    gl.uniform1i(shProgram.isSphere, true);
    sphere.DrawSphere();
    gl.uniform1i(shProgram.isSphere, false);
```

```
      ...
}

function createSphereData() {
    let radius = 0.1;
    let vertexList = [];
    let step = 10;
    let max = 360;
    for (let u = 0; u <= max; u += step) {
        for (let v = 0; v <= max; v += step) {
            let x1 = sphereCenter.x + (radius * Math.cos(deg2rad(u)) *
Math.sin(deg2rad(v)));
            let y1 = sphereCenter.y + (radius * Math.sin(deg2rad(u)) *
Math.sin(deg2rad(v)));
            let z1 = sphereCenter.z + (radius * Math.cos(deg2rad(v)));
            let x2 = sphereCenter.x + (radius * Math.cos(deg2rad(u + step)) *
Math.sin(deg2rad(v + step)));
            let y2 = sphereCenter.y + (radius * Math.sin(deg2rad(u + step)) *
Math.sin(deg2rad(v + step)));
            let z2 = sphereCenter.z + (radius * Math.cos(deg2rad(v + step)))
            vertexList.push(x1, y1, z1);
            vertexList.push(x2, y2, z2);
        }
    }
    return vertexList;
}

function readAccelerometer() {
    sensor = new Accelerometer({ frequency: 60 });
    sensor.addEventListener("reading", () => {
        sphereCenter.x = sensor.x
        sphereCenter.y = sensor.y
        sphereCenter.z = sensor.z
        sphere.BufferDataSphere(createSphereData());
    });
    sensor.start();
}

function createAudio() {
    audio = document.getElementById("audio");
    audio.addEventListener("pause", () => {
        audioContext.resume();
    });
    audio.addEventListener("play", () => {
        if (!audioContext) {
            audioContext = new (window.AudioContext ||
window.webkitAudioContext)();
            audioSource = audioContext.createMediaElementSource(audio);
            getPanner();
            getFilter();
            audioSource.connect(audioPanner);
            audioFilter.connect(audioContext.destination);
            audioContext.resume();
```

```
            audio.play();
        }
    });
}

function getPanner() {
    audioPanner = audioContext.createPanner();
    audioPanner.panningModel = "HRTF";
    audioPanner.distanceModel = "linear";
}

function getFilter() {
    if (!audioFilter) {
        audioFilter = audioContext.createBiquadFilter();
        audioFilter.type = 'lowshelf';
        audioFilter.frequency.value = 1000;
    }
    audioFilter.gain.value = document.getElementById("gain").value;
    isFilterOn = document.getElementById("isFilterOn");
    isFilterOn.addEventListener("change", function () {
        if (isFilterOn.checked) {
            audioPanner.disconnect();
            audioPanner.connect(audioFilter);
            audioFilter.connect(audioContext.destination);
        } else {
            audioPanner.disconnect();
            audioPanner.connect(audioContext.destination);
        }
    });
}
```