МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра цифрових технологій в енергетиці

Звіт з розрахунково-графічної роботи
з дисципліни «Методи синтезу віртуальної реальності»

Виконав:
студент 1 курсу магістратури
групи ТР-21мп, ТЕФ
Коцюба В. О.
Перевірив:
Демчишин А. А.

Київ — 2023

Завдання

Варіант: 8 – фільтр низьких частот.

Мета: засвоїти навички роботи з просторовим звуком за допомогою WebAudio HTML5 API.

Завдання:

1. Повторно використати код із практичного завдання №2.

2. Реалізувати обертання джерела звуку навколо геометричного центру поверхні за допомогою матеріального інтерфейсу (цього разу поверхня залишається нерухомою, а джерело звуку рухається). Відтворіть улюблену пісню у форматі mp3/ogg, маючи просторове розташування джерела звуку, кероване користувачем.

3. Візуалізувати положення джерела звуку за допомогою сфери.

4. Додати звуковий фільтр (використовувати інтерфейс BiquadFilterNode) згідно свого варіанту. Додати елемент прапорця, який увімкне або вимкне фільтр. Встановити параметри фільтра на свій смак.

Теоретичні відомості

Web Audio API — це JavaScript API високого рівня, призначене для обробки та синтезу звуку у веб-додатках. Його метою є надання можливостей, які зазвичай присутні в сучасних ігрових аудіо двигунах, а також виконання завдань зі змішування, обробки та фільтрації звуку.

AudioContext використовується для управління та відтворення звуків. Для створення звуку за допомогою *Web Audio API* потрібно створити одне або кілька джерел звуку та підключити їх до аудіо контексту, який надає екземпляр *AudioContext*. Це підключення може пройти через будь-яку кількість проміжних аудіо вузлів, які виконують обробку аудіо-сигналу.

Один екземпляр *AudioContext* може обробляти кілька вхідних звуків та складні аудіо-графи, тому для одного додатка достатньо одного екземпляра.

Аудіо джерела (*Audio Sources*) у *Web Audio API* використовуються для представлення звукових джерел. Вони є початковими елементами аудіо-графа і постачають аудіо-дані, які потім обробляються та відтворюються. Після проходження аудіо-сигналу через різні етапи обробки, його можна підключити до аудіо-призначення (наприклад, *AudioContext.destination*), де він відтворюється на динаміках або записується у файл.

Panner (або PannerNode) — це вузол в *Web Audio API*, який використовується для просторового позиціонування звукових джерел у тривимірному просторі. Він дозволяє контролювати положення звуку в просторі, зміщуючи його зліва направо (панорамування) та вгору-вниз (нахил).

Panner отримує вхідний аудіо-сигнал і використовує різні параметри для визначення положення звуку у тривимірному просторі:

1. Позиція (*Position*) — визначає положення звуку у тривимірному просторі за допомогою координат (x, y, z).

2. Орієнтація (*Orientation*) — визначає напрямок звукового джерела.

3. Ефект Доплера (*Doppler effect*) — моделює ефект Доплера, який виникає при рухомому джерелі звуку або слухачі.

BiquadFilterNode є ще одним вузлом в *Web Audio API* і використовується для застосування фільтрації до аудіо-сигналу. Він базується на математичному алгоритмі, відомому як «біквадратне рівняння» (biquadratic equation), що дозволяє застосовувати різні типи фільтрації звуку.

BiquadFilterNode дозволяє створювати різноманітні фільтри, такі як нижньо- та верхньочастотні фільтри, шельфові фільтри та інші. Він має параметри, такі як коефіцієнти фільтра, які визначають тип та поведінку фільтра, а також рівень підсилення, частоту та Q-фактор, які контролюють деталі фільтрації.

Фільтр низьких частот (*Low-pass Filter*) використовується для проходження сигналів з низькими частотами та зменшення або приглушення високочастотних складових сигналу. У *Web Audio API* цей фільтр можна реалізувати за допомогою вузла *BiquadFilterNode*.

При роботі фільтра низьких частот, вузол *BiquadFilterNode* має коефіцієнти, які визначають тип фільтра та його параметри такі, як:

1. *type: «lowpass»* — це тип фільтра, що вказує на функцію затухання сигналу.

2. *frequency*: це параметр, який визначає частоту роздільної смуги, тобто точку, де сигнал починає зменшуватись у відношенні до своєї початкової амплітуди. Нижні значення частоти дозволяють проходити низькочастотні компоненти сигналу, тоді як високочастотні компоненти затухають або приглушуються.

3. *Q-factor*: це параметр, який визначає ширину роздільної смуги фільтра. Він контролює степінь звуження частотного діапазону, де фільтр діє. Вищі значення *Q-factor* створюють вужчу роздільну смугу, тоді як нижчі значення створюють ширшу роздільну смугу.

4. *gain*: це параметр, який дозволяє змінювати рівень підсилення або приглушення сигналу після фільтрації. Додатні значення *gain* збільшують амплітуду сигналу, тоді як від'ємні значення приглушують його.

Після встановлення параметрів фільтра низьких частот, аудіо-сигнал, який проходить через вузол *BiquadFilterNode*, буде піддаватись фільтрації, що призведе до приглушення або пригнічення високочастотних складових сигналу, а низькочастотні компоненти будуть проходити майже без змін.

Опис деталей реалізації

Для опису процесу реалізації сценарію, який включає створення обертаючої сфери навколо фігури та завантаження звукової доріжки, необхідно розглянути такі основні етапи реалізації програми:

1. Починаємо зі створення змінних для маніпуляцій з джерелом звуку та приєднання аудіо контексту браузера до джерела звуку.

```
export let ctx;
export let source;
export let panner;
export let filter;
```

2. Підключаємо джерело звуку до Web Audio API, використовуючи змінну контексту.

```
export function handleAudioSlider() {
  const audio = document.getElementById("audio");
  audio.addEventListener("play", (e) => {
    if (!ctx) {
      ctx = new (window.AudioContext || window.webkitAudioContext)();
...
```

3. Отримуємо джерело звуку, прослуховуючи події аудіо елементу, щоб мати доступ до існуючого аудіо контексту або створити новий.

```
async function init() {
    ...
    handleAudioSlider();
    ...
}
```

4. Додаємо фільтр до звуку.

```
      source = ctx.createMediaElementSource(audio);
      panner = ctx.createPanner();
      filter = ctx.createBiquadFilter();

      source.connect(panner);
      panner.connect(filter);
      filter.connect(ctx.destination);
```

5. Налаштовуємо фільтр, встановлюючи тип та частоту фільтрації.

```
      filter.type = "lowpass";
      filter.frequency.value = 1500;
      ctx.resume();
```

6. Генеруємо масив точок у формі сфери для відображення місцезнаходження звукового джерела у просторі.

```
export function CreateSphereData(multiplier, iSegments, jSegments) {
  const vertexList = [];
  const textureList = [];
```

```
  for (let i = 0; i <= iSegments; i++) {
    const theta = (i * Math.PI) / iSegments;
    const sinTheta = Math.sin(theta);
    const cosTheta = Math.cos(theta);

    for (let j = 0; j <= jSegments; j++) {
      const phi = (j * 2 * Math.PI) / jSegments;
      const sinPhi = Math.sin(phi);
      const cosPhi = Math.cos(phi);

      const x = multiplier * cosPhi * sinTheta;
      const y = multiplier * cosTheta;
      const z = multiplier * sinPhi * sinTheta;

      vertexList.push(x, y, z);

      const u = 1 - j / jSegments;
      const v = 1 - i / iSegments;
      textureList.push(u, v);
    }
  }

  return { vertexList, textureList };
}
```

7. Додаємо метод для малювання сфери.

```
  this.DrawSphere = function () {
    this.Draw();
    gl.drawArrays(gl.LINE_STRIP, 0, this.count);
  };
```

8. Прив’язуємо звук до позиції сфери, розраховуючи нове значення просторової позиції звуку на кожному кадрі рендерингу за допомогою функції panner.setPosition().

```
  panner?.setPosition(...sphereCoords);
```

9. Додаємо кнопки управління, такі як прапорці для увімкнення та вимкнення фільтру та аудіо елемент для управління звуковою доріжкою, що дозволяють включати/виключати музику, перемотувати її та регулювати гучність.

```
      <div class="audio-wrapper">
        <audio id="audio" src="./assets/rose_golden.mp3" loop controls></audio>

        <label for="filter">Lowpass filter</label>
        <input id="filter" type="checkbox" checked />
      </div>
```

Інструкція користувача

Розроблений додаток має наступний вигляд (рисунок 1). Зверху розташований опис роботи, потім набір елементів для зміни різних параметрів фігури, а також налаштування звуку.

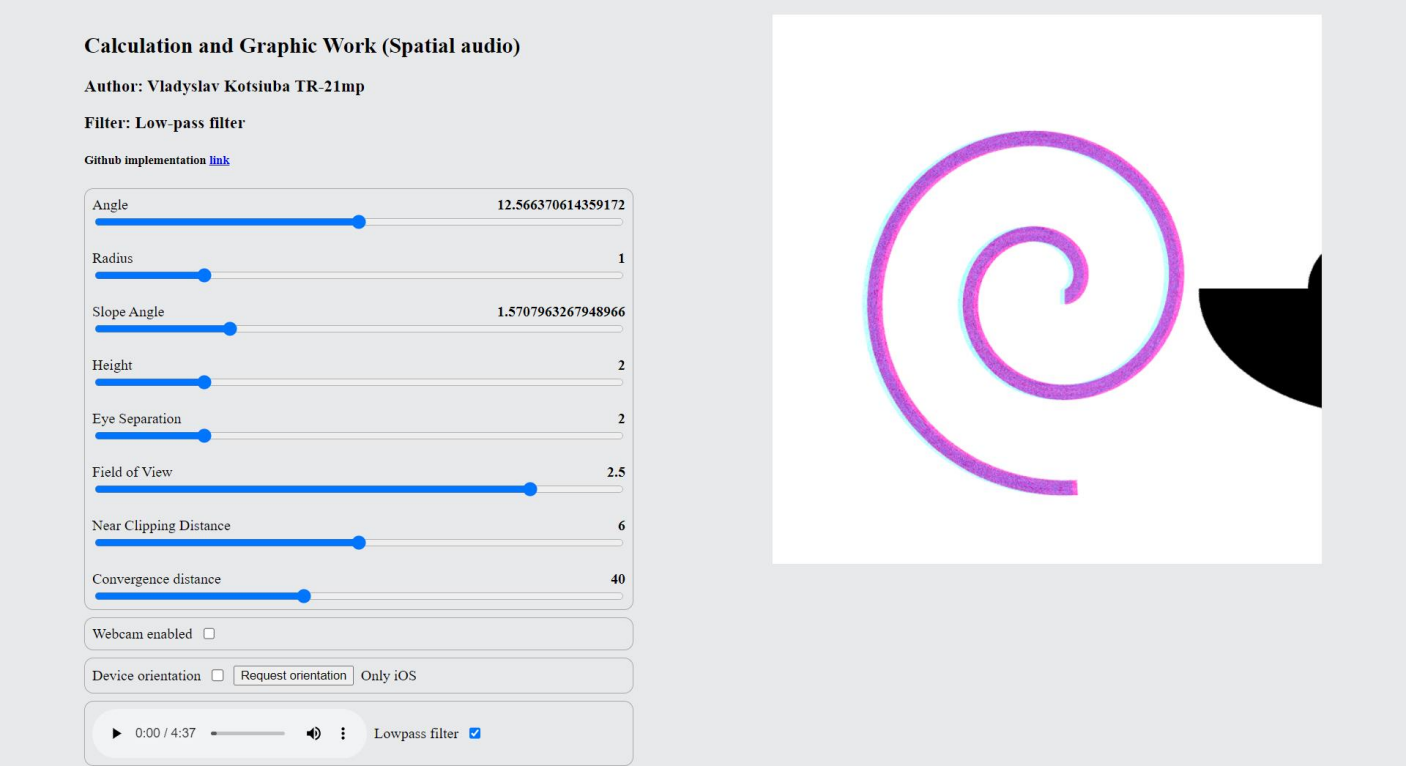

Рисунок 1 — Зображення сторінки програми

Перший блок елементів відповідає за зміну параметрів фігури, таких як angle, radius, slope angle, height, а також її 3D налаштувань, наприклад, eye separation, field of view, near clipping distance та convergence distance.

Другий блок відповідає за вмикання камери та відображення зображення з неї як заднього фону для графічного полотна.

Третій блок відповідає за контроль над статусом роботи просторової орієнтації.

Четвертий блок відповідає за звукову складову. Наявний аудіо плеєр для контролю над звуковою доріжкою та перемикач накладання фільтру. Елемент аудіо складається з кнопки паузи або старту відтворення, відображення часу відтворення, кнопки для регулювання звуку та додаткових опцій (завантажити аудіо файл або змінити швидкість програвання). За замовчуванням, фільтр є увімкненим.

З правої сторони сторінки знаходиться елемент canvas, на якому знаходяться дві фігури: основна фігура та сфера джерела звуку (рисунок 2).

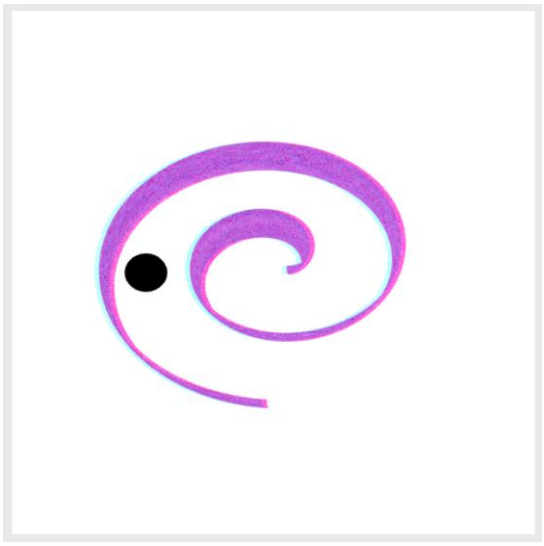

Рисунок 2 — Графічне полотно з два фігурами: основна фігура та сфера джерела звуку

Для мобільних пристроїв макет сторінки змінюється на відповідний згідно розмірів екрану пристрою (рисунок 3).

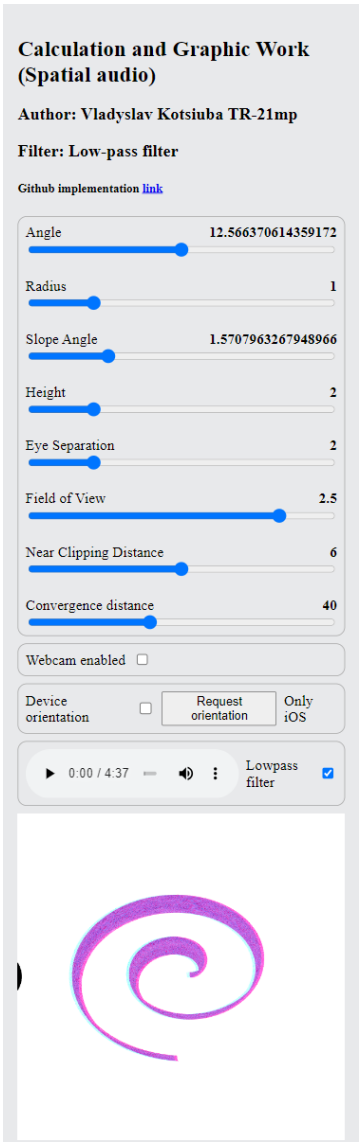

Рисунок 3 — Приклад інтерфейсу сторінки для мобільних пристроїв

Лістинг програмного коду

Програмний код наведений не повністю, наведені лише нові додані рядки вихідного коду.

Файл програмного коду за шляхом: src/audio.js

```
export let ctx;
export let source;
export let panner;
export let filter;

export function handleAudioSlider() {
  const audio = document.getElementById("audio");
  audio.addEventListener("play", (e) => {
    if (!ctx) {
      ctx = new (window.AudioContext || window.webkitAudioContext)();

      source = ctx.createMediaElementSource(audio);
      panner = ctx.createPanner();
      filter = ctx.createBiquadFilter();

      source.connect(panner);
      panner.connect(filter);
      filter.connect(ctx.destination);

      filter.type = "lowpass";
      filter.frequency.value = 1500;
      ctx.resume();
    }
  });
}

export function handleFilterChange() {
  const filterCheckbox = document.getElementById("filter");
  filterCheckbox.addEventListener("change", async function (e) {
    const isChecked = e.target.checked;
    if (isChecked) {
      panner?.disconnect();
      panner?.connect?.(filter);
      filter?.connect?.(ctx.destination);
    } else {
      panner?.disconnect();
      panner?.connect?.(ctx.destination);
    }
  });
}
```

Файл програмного коду за шляхом: src/sphere.js

```
export function CreateSphereData(multiplier, iSegments, jSegments) {
  const vertexList = [];
  const textureList = [];
```

```
  for (let i = 0; i <= iSegments; i++) {
    const theta = (i * Math.PI) / iSegments;
    const sinTheta = Math.sin(theta);
    const cosTheta = Math.cos(theta);

    for (let j = 0; j <= jSegments; j++) {
      const phi = (j * 2 * Math.PI) / jSegments;
      const sinPhi = Math.sin(phi);
      const cosPhi = Math.cos(phi);

      const x = multiplier * cosPhi * sinTheta;
      const y = multiplier * cosTheta;
      const z = multiplier * sinPhi * sinTheta;

      vertexList.push(x, y, z);

      const u = 1 - j / jSegments;
      const v = 1 - i / iSegments;
      textureList.push(u, v);
    }
  }

  return { vertexList, textureList };
}

export function moveSphere(sphereCoords, angle, offsetX = 0, offsetZ = -5, radius = 4)
{
  sphereCoords[0] = radius * Math.cos(angle) + offsetX;
  sphereCoords[2] = radius * Math.sin(angle) + offsetZ;
  return sphereCoords;
}
```

Файл програмного коду за шляхом: main.js

```
let sphere; // A sphere model
let sphereStep = 0;
let sphereCoords = [0, 0, 0];

function degToRad(degrees) {
  return (degrees * Math.PI) / 180;
}

// Constructor of the Model
function Model(name) {
  this.name = name;
  this.iVertexBuffer = gl.createBuffer();
  this.iTextureBuffer = gl.createBuffer();
  this.count = 0;

  this.DrawSphere = function () {
    this.Draw();
    gl.drawArrays(gl.LINE_STRIP, 0, this.count);
```

```
  };
}

function draw() {
  gl.clearColor(1, 1, 1, 1);

  let modelView;
  if (deviceOrientation.checked && latestEvent.alpha && latestEvent.beta &&
latestEvent.gamma) {
    const alphaRad = (latestEvent.alpha * Math.PI) / 180;
    sphereCoords = moveSphere(sphereCoords, alphaRad + Math.PI / 2);
  } else {
    sphereStep += 0.015;
    sphereCoords = moveSphere(sphereCoords, sphereStep);
  }
  modelView = spaceball.getViewMatrix();

  panner?.setPosition(...sphereCoords);
  gl.bindTexture(gl.TEXTURE_2D, null);

  const projection = m4.perspective(degToRad(90), 1, 0.1, 100);
  const translationShere = m4.translation(...sphereCoords);
  const modelViewMatrix = m4.multiply(translationShere, modelView);

  gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projection);
  gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, modelViewMatrix);
  sphere.DrawSphere();
}

// Initialize the WebGL context
function initGL() {
  sphere = new Model("Sphere");
  const sphereData = CreateSphereData(0.5, 500, 500);
  sphere.BufferData(sphereData.vertexList, sphereData.textureList);
}
```