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»
Варіант 13

Виконав: Осадчий Д.О.
Студент групи ТР-23мп

Київ 2023

1. Завдання

Тема роботи: Звук у просторі. Імлементувати звук у просторі за допомогою WebAudio HTML5 API

Вимоги:

- Перевикористати код з практичної роботи №2.
- Імплементувати обертання джерела звуку навколо геометричного центру поверхні за допомогою матеріального інтерфейсу. Програвати улюблену пісню у форматі mp3/ogg, змінюючи розташування джерела звуку відповідно до введення користувача.
- Візуалізувати джерело звуку у вигляді сфери.
- Додати звуковий фільтр за варіантом. Додати «галочку», яка вмикає ти вимикає фільтр. Задати параметри фільтру за смаком.

2. Теоретичні відомості

Web Audio API — це потужний інструмент, який дозволяє розробникам маніпулювати та синтезувати звук у веб-додатках. Він надає набір інтерфейсів і об’єктів, які дозволяють створювати, модифікувати та маршрутизувати аудіосигнали в реальному часі. Одним із ключових аспектів API веб-аудіо є його здатність обробляти аудіо та керувати ним за допомогою модульного підходу, який дозволяє створювати складні конвеєри обробки аудіо.

Серед багатьох об’єктів, доступних у API веб-аудіо, три широко використовувані – це AudioContext, MediaElementSourceNode, PannerNode і BiquadFilterNode. Давайте заглибимося в кожен із цих об’єктів і дослідимо їхні ролі та функції.

AudioContext:

AudioContext представляє граф обробки аудіо та діє як центральний центр для створення та підключення аудіо вузлів. Він служить точкою входу для доступу та керування аудіофункціями, які надає Web Audio API. Створюючи екземпляр AudioContext, розробники отримують доступ до широкого спектру методів і властивостей для керування відтворенням аудіо, маршрутизацією та ефектами.

У фрагменті коду “context = new AudioContext();” ініціалізується об’єкт AudioContext, який служить основою для конвеєра обробки звуку.

MediaElementSourceNode:

MediaElementSourceNode використовується для отримання аудіоданих із медіа-елементів HTML, таких як <audio> або <video>. Він являє собою джерело аудіо, яке можна підключити до інших аудіо вузлів для подальшої обробки або маршрутизації. Використовуючи MediaElementSourceNode, розробники можуть включати існуючі медіа-елементи в екосистему Web Audio API і застосовувати різні звукові ефекти або маніпуляції.

У коді “source = context.createMediaElementSource(audio);” створює MediaElementSourceNode, де змінна audio посилається на елемент HTML

<audio>. Це дозволяє обробляти аудіодані з указаного медіа-елемента за допомогою Web Audio API.

PannerNode:

PannerNode відповідає за просторове позиціонування та панорамування звуку. Він імітує тривимірне аудіо, регулюючи положення, орієнтацію та швидкість аудіоджерела у віртуальному 3D-просторі. Цей об’єкт дозволяє розробникам створювати ефект занурення в аудіо, коли звук виходить із певних напрямків, створюючи відчуття глибини та руху.

У коді “panner = context.createPanner();” створює PannerNode, який можна підключити до звукового графіка. PannerNode можна використовувати для керування положенням і рухом аудіоджерела, забезпечуючи динамічне розподілення звуку в просторі.

BiquadFilterNode:

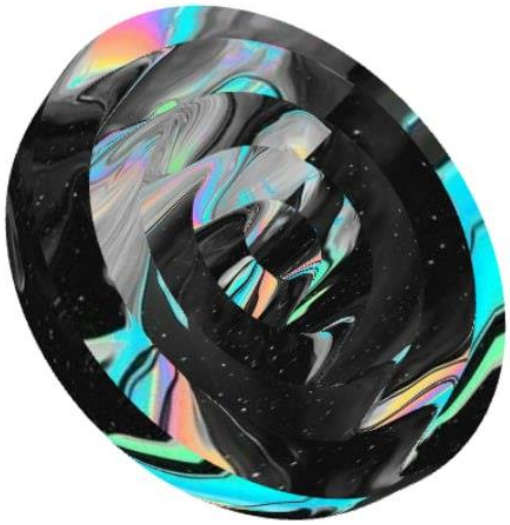
BiquadFilterNode реалізує різні типи цифрових фільтрів, наприклад фільтри низьких частот, високочастотні, смугові та пікові фільтри. Це дозволяє розробникам формувати частотну характеристику аудіосигналу, змінюючи його тембр і застосовуючи такі ефекти, як вирівнювання або резонанс. BiquadFilterNode пропонує параметри для керування частотою зрізу, посиленням і якістю фільтра.

У коді “biquadFilter = context.createBiquadFilter();” створює BiquadFilterNode. Після підключення до аудіографа цей вузол можна використовувати для застосування ефектів фільтрації до аудіосигналу, покращуючи або змінюючи його спектральні характеристики.

Підсумовуючи, Web Audio API надає потужний набір об’єктів, які дозволяють розробникам маніпулювати та обробляти звук у веб-додатках. AudioContext діє як основний інтерфейс, тоді як такі об’єкти, як MediaElementSourceNode, PannerNode і BiquadFilterNode, пропонують спеціальні функції для вилучення аудіоданих, розміщення звуку у віртуальному просторі та застосування ефектів цифрової фільтрації. Використовуючи ці об’єкти та можливості Web Audio API, розробники можуть створювати захоплюючі та інтерактивні аудіо в Інтернеті.

3. Особливості виконання завдання

Під час другої лабораторної роботи була реалізована можливість обертати "Поверхня обертання загального синусоїда" у стереоскопічному форматі за допомогою програмного сенсора смартфона: при обертанні телефону відповідно оберталася й фігура.



Поверхня обертання загального синусоїда

Була виконана робота з використанням Web Audio API, використовуючи документацію, яка доступна на веб-сторінці https://webaudio.github.io/web-audio-api/. Перш за все, був створений об'єкт аудіоконтексту, який дозволяє використовувати Web Audio API. У цій роботі був використаний аудіофайл у форматі mp3, який був відображений на веб-сторінці за допомогою HTML-елемента <audio>. Далі було створено джерело звуку, передаючи аудіоелемент у конструктор. Також був створений об'єкт panner в контексті для подальшого керування звуком, зокрема його позицією, яка буде

змінюватися при обертанні телефону (джерело звуку буде розташовано на відстані 2 від центру, у відповідному напрямку обертання телефону в просторі). Важливою частиною роботи було застосування фільтра до вихідного звуку. За варіантом було реалізовано шельфовий фільтр низьких частот з параметрами, вказаними в розділі 5. Потім було з'єднано відповідні об'єкти між собою. Був доданий eventListener, який відповідає за зупинку та продовження програвання аудіофайлу. Також було створено поле для увімкнення та вимкнення фільтра, і був доданий інший eventListener для перемикання фільтра, залежно від стану цього поля. Оновлення позиції звуку через переміщення об'єкту panner було реалізовано в основній функції з назвою draw.

4. Приклад Роботи

Користувач може керувати рухом умовної сфери, яка відображає умовне місцезнаходження джерела звуку. Відповідні рухи можна побачити на рисунках 4.1.

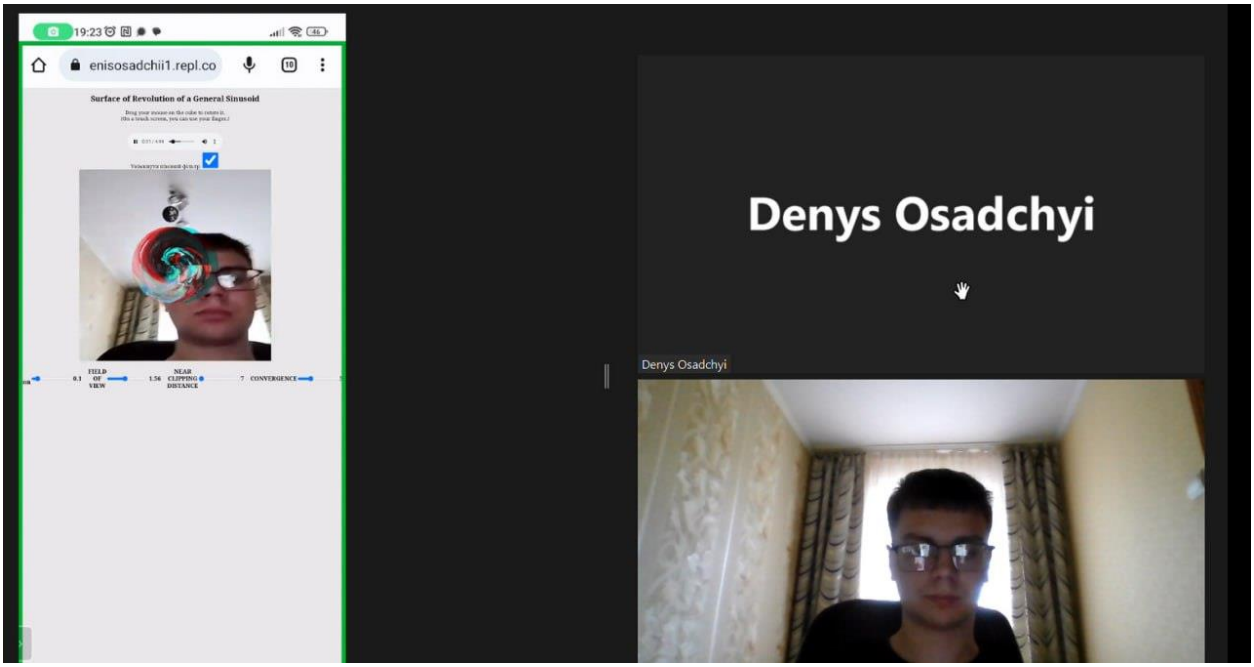


Рисунок 4.1 Матеріальний інферфейс повернуто догори

Під час обертання телефону сфера рухається навколо фігури, створюючи ефект переміщення джерела звуку, який найкраще відчувається у навушниках та аудіосистемах зі стерео-ефектом. Крім того, на сторінці присутні елементи інтерфейсу для зміни параметрів стереозображення, таких як відстань між очима (eye separation), поле зору (field of view), найближча відстань (near clipping distance) та збіжність (convergence). Дані повзунки можна побачити на рисунку 4.2.

Рисунок 4.2 Слайдери для зміни параметрів стерео зображення

«чекбокс» для увімкнення та вимкнення фільтру, див. рисунок 4.3.

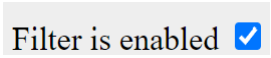


Рисунок 4.3 Чекбокс перемикання стану фільтру

На сторінці присутні елементи управління аудіофайлом, такі як перемотка, пауза, продовження, регулювання гучності. Дивіться рисунок 4.4, щоб побачити ці елементи .

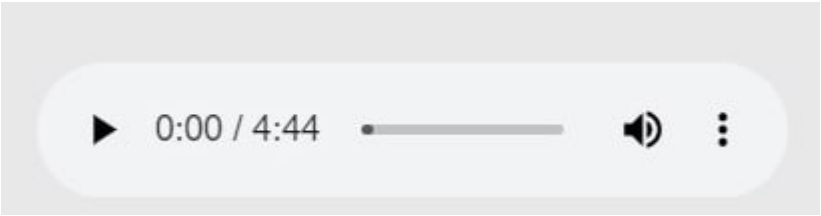


Рисунок 4.4 Елемент управління аудіофайлом (Audio Control Element)

5. Деякі зразки коду

Код функцій для налаштування та ініціалізації аудіо

```
let audio = null;
let audioContext;
let source;
let panner;
let filter;

function initializeAudio() {
  audio = document.getElementById('audio');

  audio.addEventListener('play', handlePlay);

  audio.addEventListener('pause', handlePause);
}

function handlePlay() {
  console.log('play');
  if (!audioContext) {
    audioContext = new AudioContext();
    source = audioContext.createMediaElementSource(audio);
    panner = audioContext.createPanner();
    filter = audioContext.createBiquadFilter();

    // Connect audio nodes
    source.connect(panner);
    panner.connect(filter);
    filter.connect(audioContext.destination);

    // Set filter parameters
    filter.type = 'peaking';
    filter.Q.value = 2;
```

```
    filter.frequency.value = 500;
    filter.gain.value = 20;

    audioContext.resume();
  }
}

function handlePause() {
  console.log('pause');
  audioContext.resume();
}

function toggleFilter() {
  let filterCheckbox = document.getElementById('filterCheckbox');
  if (filterCheckbox.checked) {
    // Connect filter when checkbox is checked
    panner.disconnect();
    panner.connect(filter);
    filter.connect(audioContext.destination);
  } else {
    // Disconnect filter when checkbox is unchecked
    panner.disconnect();
    panner.connect(audioContext.destination);
  }
}

function startAudio() {
  initializeAudio();

  let filterCheckbox = document.getElementById('filterCheckbox');
  filterCheckbox.addEventListener('change', toggleFilter);

  audio.play();
}
```

Код основної функції draw

```
function draw() {
  onPerspectiveChange();
  gl.clearColor(1, 1, 1, 1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  /* Set the values of the projection transformation */
  let projection = m4.orthographic(-4, 4, -4, 4,
stereoCamera.mNearClippingDistance, 4 * 4);

  /* Get the view matrix from the SimpleRotator object.*/
  defineAccelMat()
  // let modelView = m4.multiply(rotateBall.getViewMatrix(),
matAccel);
  let modelView = rotateBall.getViewMatrix();
  let modelView_bg = [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1]

  let rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0.);
  let translateToPointZero = m4.translation(0, 0, -10);

  let matAccum0 = m4.multiply(rotateToPointZero, modelView);
  let matAccum0_bg = m4.multiply(rotateToPointZero, modelView_bg);
  let matAccum1 = m4.multiply(translateToPointZero, matAccum0);
  let matAccum1_bg = m4.multiply(translateToPointZero,
matAccum0_bg);

  /* Multiply the projection matrix times the modelview matrix to
give the
     combined transformation matrix, and send that to the shader
program. */
  let modelViewProjection = m4.multiply(projection, matAccum1);
  let modelViewProjection_bg = m4.multiply(projection,
matAccum1_bg);
```

```
  gl.uniformMatrix4fv(
    shaderProgram.iModelViewProjectionMatrix,
    false,
    modelViewProjection_bg
  );

  gl.uniform1i(shaderProgram.iTMU, 0);
  gl.enable(gl.TEXTURE_2D);
  gl.bindTexture(gl.TEXTURE_2D, videoTexture);
  gl.texImage2D(
    gl.TEXTURE_2D,
    0,
    gl.RGBA,
    gl.RGBA,
    gl.UNSIGNED_BYTE,
    video
  );
  gl.uniform1f(shaderProgram.iB, 1);
  gl.uniform3fv(shaderProgram.iTranslateSphere, [-5, -5, -0]);
  bg_surface.Draw();
  gl.clear(gl.DEPTH_BUFFER_BIT);
  stereoCamera.ApplyLeftFrustum();
  gl.uniformMatrix4fv(
    shaderProgram.iModelViewProjectionMatrix,
    false,
    // modelViewProjection
    m4.multiply(stereoCamera.mModelViewMatrix,
m4.multiply(m4.multiply(stereoCamera.mProjectionMatrix,
translateToPointZero), modelView))
  );

  gl.uniform2fv(shaderProgram.iUserPoint, [userPoint.x,
userPoint.y]);
  gl.uniform1f(shaderProgram.iMagnit, magnit);
  gl.uniform1f(shaderProgram.iB, -1);
```

```
  gl.uniform3fv(shaderProgram.iTranslateSphere, [-0, -0, -0]);
  gl.bindTexture(gl.TEXTURE_2D, imageTexture);
  gl.colorMask(true, false, false, false);
  surface.Draw();
  gl.clear(gl.DEPTH_BUFFER_BIT);
  stereoCamera.ApplyRightFrustum();
  gl.uniformMatrix4fv(
    shaderProgram.iModelViewProjectionMatrix,
    false,
    // modelViewProjection
    m4.multiply(stereoCamera.mModelViewMatrix,
m4.multiply(m4.multiply(stereoCamera.mProjectionMatrix,
translateToPointZero), modelView))
  );
  gl.colorMask(false, true, true, false);
  surface.Draw();
  gl.clear(gl.DEPTH_BUFFER_BIT);
  gl.colorMask(true, true, true, true);
  // let translate = Sinus(
  //   map(userPoint.x, 0, 1, 0, 5),
  //   map(userPoint.y, 0, 1, 0, Math.PI * 2)
  // );
  let translate = [0.5 * sensor.x, 0.5 * sensor.y, (-0.5) *
sensor.z]
  if (panner) {
    panner.setPosition(translate[0],
      translate[1],
      translate[2])
  }
  gl.uniform3fv(shaderProgram.iTranslateSphere, [
    translate[0],
    translate[1],
    translate[2],
  ]);
  gl.uniform1f(shaderProgram.iB, 1);
```

```
  sphere.DrawSphere();


}
```