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 13

Виконав студент 5-го курсу
групи ТР-32мп
Лиштван Владислав Валерійович

Київ 2024

Завдання

1. Імплементувати просторове аудіо за допомогою WebAudio HTML5 API, використовуючи код з практичного завдання 2.

2. Реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні по колу протягом певного часу (поверхня залишається нерухомою, а джерело звуку рухається). Відтворювати улюблену пісню у форматі mp3/ogg, маючи просторове розташування джерела звуку, кероване користувачем.

3. Візуалізувати джерело звуку за допомогою сфери.

4. 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.

Теоретичні відомості

Web Audio API - це високорівневе JavaScript API, яке призначене для обробки та синтезу звуку у веб-додатках. Його головною метою є надання можливостей, що характерні для сучасних ігрових аудіо-двигунів, а також виконання деяких завдань зі змішування, обробки та фільтрації звуку.

AudioContext використовується для управління та відтворенням звуків. Для створення звуку за допомогою Web Audio API потрібно створити одне або кілька джерел звуку та підключити їх до об'єкту, який відповідає за звук (AudioContext). Підключення може бути прямим або через будь-яку кількість проміжних вузлів обробки аудіо, які виступають як модулі обробки аудіо-сигналу. Один AudioContext може обробляти кілька вхідних звуків та складні аудіо-структури, тому для одного додатку досить одного екземпляру.

Audio Sources у Web Audio API використовуються для представлення аудіо джерел. Вони є початковими компонентами аудіо-графа і надають 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, зокрема панорамування (рух зліва направо) і нахил (рух вгору і вниз). Panner отримує вхідний аудіо-сигнал і застосовує різні параметри для визначення положення звуку в тривимірному просторі:

- Position: визначає місцезнаходження звуку в тривимірному просторі за допомогою координат (x, y, z);

- Orientation: визначає орієнтацію звукового джерела, тобто його напрямок;

- Doppler effect: моделює ефект Доплера, який виникає при рухомому джерелі звуку або слухачі.

BiquadFilterNode - це один з вузлів у Web Audio API, який використовується для застосування фільтрації до аудіо-сигналу. Цей вузол

ґрунтується на математичному алгоритмі, відомому як "біквадратне рівняння" (biquadratic equation), що дозволяє використовувати різні типи фільтрації звуку.

BiquadFilterNode дозволяє створювати різноманітні фільтри, такі як низько- та високочастотні фільтри, шельфові фільтри та інші. Крім того, він має параметри, такі як коефіцієнти фільтра, які визначають його тип і поведінку, а також рівень підсилення, частоту та Q-фактор, що регулюють деталі фільтрації.

Фільтри нижчого порядку використовуються для базових налаштувань тембру, графічних еквалайзерів і складніших фільтрів. Декілька BiquadFilterNode можуть бути об'єднані для створення складніших фільтрів. Параметри фільтра, такі як частота, можуть змінюватися з часом для динамічного налаштування. Кожен BiquadFilterNode може мати один із типів фільтрів, як показано в IDL. За замовчуванням тип фільтра - «НЧ».

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

Фільтр високих частот є протилежністю фільтру низьких частот. Він пропускає частоти вище граничної та послаблює частоти нижче. Він реалізує стандартний резонансний фільтр високих частот другого порядку із спадом 12 дБ/октаву.

Смуговий фільтр пропускає діапазон частот і послаблює частоти нижче та вище цього діапазону. 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 пропускає всі частоти, крім певного діапазону частот.

Деталі впровадження

Реалізувано обертання джерела звуку навколо геометричного центру ділянки поверхні по колу протягом певного часу (поверхня залишається нерухомою, а джерело звуку рухається). Джерело звуку візуалізовано у вигляді сферичної геометрії.

Для реалізації просторового звуку створимо HTML-елемент <audio>, який містить інформацію про джерело аудіофайлу, в даному випадку "filename" у форматі MP3/OGG. Цей елемент керування дозволяє зупиняти та відновлювати відтворення аудіофайлу.

Після цього, за допомогою JavaScript та WebAudio API, було створено об'єкт AudioContext, для якого було створено та під'єднано три основні об'єкти:

- джерело звуку (MediaElementSource);
- об'єкт обробки просторового звуку (Panner);
- звуковий фільтр (BiquadFilter).

Ці об'єкти забезпечують функціонал для обробки та маніпуляції звуковим потоком, включаючи можливість позиціонування джерела звуку у просторі.

За варіантом обрано піковий фільтр. BiquadFilter має три основні параметри: frequency (частота), Q (ширина смуги), gain (підсилення). Встановлено необхідні параметри для обраного фільтру, а саме frequency = 1000, Q = 4 та gain = 12. Особливості параметрів BiquadFilter для пікового фільтру полягають в наступному:

- частота (frequency) - це центральна частота, на якій застосовується підсилення або ослаблення. Вона визначає, навколо якої частоти буде зосереджено вплив фільтру;
- Q - контролює ширину смуги частот навколо центральної частоти, на яку буде поширюватися підсилення або ослаблення. Велике значення Q означає вужчу смугу частот, а отже, більш локалізований вплив фільтру;
- посилення (gain) - визначає рівень підсилення або ослаблення в децибелах (дБ), що буде застосовано до виділеного діапазону частот.

Якщо значення позитивне, ці частоти будуть підсилені, а якщо від'ємне - послаблені.

Джерело звуку, обробка якого в просторі здійснюється об'єктом класу Panner, зображується у WebGL контексті у вигляді сфери, щоб його можна було переміщати. Зміна параметрів об'єкту Panner відповідно впливає на сприйняття звуку під час прослуховування стереозвуку, створюючи ефект переміщення джерела звуку відповідно до місцезнаходження сфери в системі координат.

Інструкція користувача

Розроблений застосунок зображено на рисунку 1. Зверху знаходиться опис, потім набір елементів для зміни різних параметрів фігури та звуку.

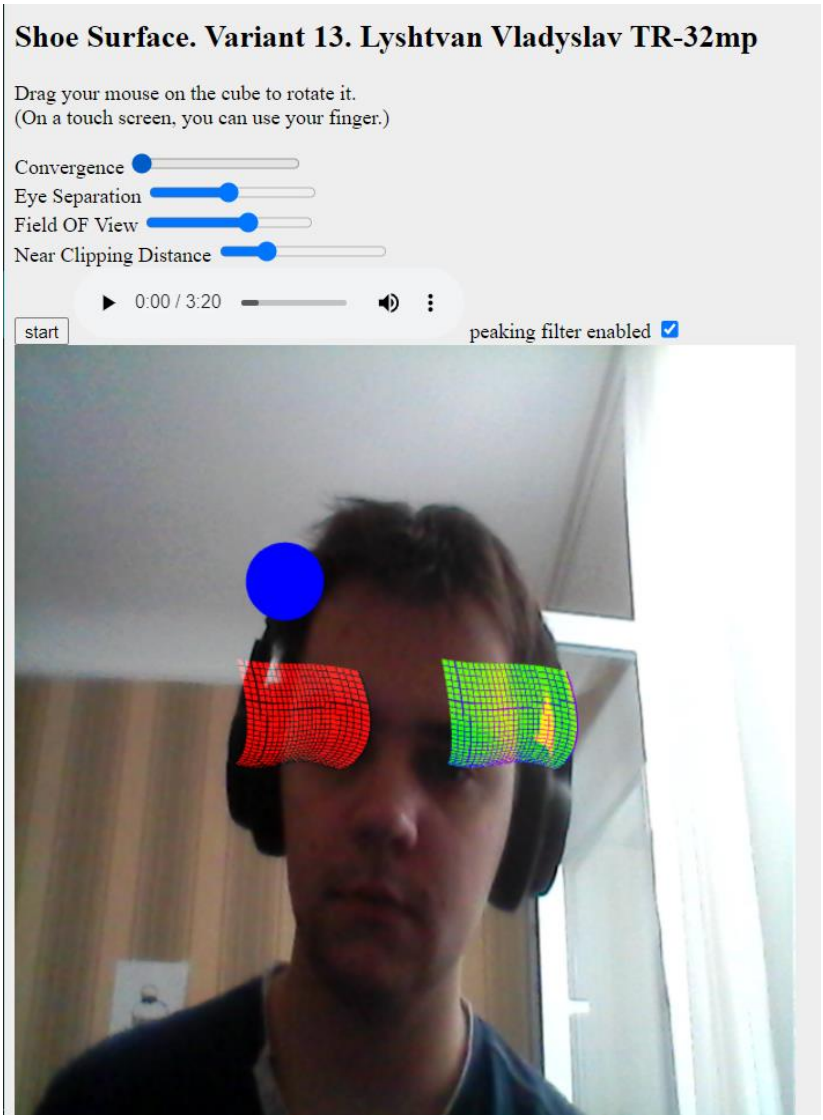

Рисунок 1. Інтерфейс розробленого застосунку

Перший блок елементів відповідає за зміну параметрів основної фігури, таких як eye separation, convergence distance, field of view та near clipping distance.

Другий блок відображає елемент audio та відповідає за ввімкнення, перемотку, вимкнення аудіо.

Третій блок відповідає за ввімкнення/вимкнення пікового фільтру та старт функції initAudio. У цьому блоці можна ввімкнути або вимкнути фільтр (за замовчуванням він увімкнений) та почати роботу аудіо з фільтром. Усі зміни одразу відображаються на екрані та впливають на звук.

Останнім блоком на сторінці є елемент canvas, у якості фону на якому зображене відео з веб-камери пристрою, поверх якого знаходяться дві фігури: основна фігура та сфера, що є джерелом звуку. На рисунках 2 та 3 зображено зміну параметрів основної фігури та зміну положення сфери, яка обертається по колу протягом певного часу.

Рисунок 2. Вигляд основної фігури та сфери зі зміненими параметрами

Рисунок 3. Вигляд основної фігури та сфери зі зміненими параметрами

Лістинг коду

```
let audioContext;
let audio = null;
let src;
let peakingFilter;
let audioPos;
function setupAudioListeners() {
  audio = document.getElementById('audioElement');
  audio.addEventListener('play', () => {
    if (!audioContext) {
      //console.log("here")
      audioContext = new AudioContext();
      src = audioContext.createMediaElementSource(audio);
      audioPos = audioContext.createPanner();
      peakingFilter = audioContext.createBiquadFilter();
      src.connect(audioPos);
      audioPos.connect(peakingFilter);
      peakingFilter.connect(audioContext.destination);
      peakingFilter.type = 'peaking';
      peakingFilter.frequency.value = 1000;
      peakingFilter.Q.value = 4;
      peakingFilter.gain.value = 12;
      audioContext.resume();
    }
  });
  audio.addEventListener('pause', () => {
    console.log('pause');
    audioContext.resume();
  });
}
function initAudio() {
  setupAudioListeners();
  let filterListener = document.getElementById('filterEnabled');
  filterListener.addEventListener('change', function () {
    if (filterListener.checked) {
      audioPos.disconnect();
      audioPos.connect(peakingFilter);
      peakingFilter.connect(audioContext.destination);
    } else {
      audioPos.disconnect();
      audioPos.connect(audioContext.destination);
    }
  });
  audio.play();
}
function draw(animate=false) {
  gl.clearColor(0, 0, 0, 1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  let projection = m4.perspective(Math.PI / 8, 1, 8, 12);
  let modelView = spaceball.getViewMatrix();
```

```
  let rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0.7);
  let translateToPointZero = m4.translation(0, 0, -5);
  let matAccum0 = m4.multiply(rotateToPointZero, modelView);
  let matAccum1 = m4.multiply(translateToPointZero, matAccum0);
  gl.uniform1f(shProgram.iT, true);
  gl.bindTexture(gl.TEXTURE_2D, vTexture);
  gl.texImage2D(
        gl.TEXTURE_2D,
        0,
        gl.RGBA,
        gl.RGBA,
        gl.UNSIGNED_BYTE,
        video
  );
  gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, m4.identity());
  vModel.DrawTextured();
  gl.clear(gl.DEPTH_BUFFER_BIT)
  gl.uniform1f(shProgram.iT, false);
  let modelViewProjection = m4.multiply(projection, matAccum1);
  const now = Date.now() * 0.0001;
  gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false,
m4.translation(0.5 * Math.sin(now), 0.5 * Math.cos(now), 0));
  if (audioPos) {
    audioPos.setPosition(0.5 * Math.sin(now), 0.5 * Math.cos(now), 0)
  }
  sphere.Draw();
  gl.clear(gl.DEPTH_BUFFER_BIT);
  cam.ApplyLeftFrustum();
  modelViewProjection = m4.multiply(cam.projection, m4.multiply(cam.modelView,
matAccum1));
  gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false,
modelViewProjection);
  gl.colorMask(true, false, false, false);
  gl.uniform4fv(shProgram.iColor, [1, 1, 0, 1]);
  surface.Draw();
  gl.uniform4fv(shProgram.iColor, [0, 0, 1, 1]);
  surface.DrawLines();
  gl.clear(gl.DEPTH_BUFFER_BIT);
  cam.ApplyRightFrustum();
  modelViewProjection = m4.multiply(cam.projection, m4.multiply(cam.modelView,
matAccum1));
  gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false,
modelViewProjection);
  gl.colorMask(false, true, true, false);
  gl.uniform4fv(shProgram.iColor, [1, 1, 0, 1]);
  surface.Draw();
  gl.uniform4fv(shProgram.iColor, [0, 0, 1, 1]);
  surface.DrawLines();
  gl.colorMask(true, true, true, true);
  if (animate) {
    window.requestAnimationFrame(()=>draw(true));
  }
}
```