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 21

Виконав: Сергєєв Данило Вікторович

Студент групи ТР-21мп

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.

1. Теоретичні відомості

Web Audio API є потужним інструментом для маніпулювання та синтезу звуку у веб-додатках. Вона надає набір інтерфейсів і об'єктів для створення, модифікації та маршрутизації аудіосигналів у реальному часі. Основною особливістю Web Audio API є його модульний підхід, що дозволяє створювати складні обробники аудіо.

У API веб-аудіо доступні різноманітні об'єкти, із яких три широко використовуються: AudioContext, MediaElementSourceNode, PannerNode і BiquadFilterNode. Давайте детальніше розглянемо кожен з них і їх функції.

AudioContext є центральним елементом у графі обробки аудіо і служить точкою входу для створення та підключення аудіовузлів. Він надає доступ до методів і властивостей для керування відтворенням аудіо, маршрутизацією та ефектами. Ініціалізація об'єкта AudioContext, наприклад, через "context = new AudioContext();", створює основу для обробки звуку.

MediaElementSourceNode використовується для отримання аудіоданих з елементів HTML, таких як <audio> або <video>. Цей вузол є джерелом аудіо, яке можна підключити до інших аудіовузлів для обробки або маршрутизації. За допомогою MediaElementSourceNode розробники можуть включати існуючі медіа-елементи в Web Audio API і застосовувати до них різноманітні звукові ефекти або маніпуляції. Приклад створення MediaElementSourceNode: "source = context.createMediaElementSource(audio);", де змінна audio вказує на елемент HTML <audio>, що дозволяє обробляти аудіодані з цього елемента.

PannerNode відповідає за просторове позиціонування та панорамування звуку. Він імітує тривимірне аудіо, керуючи положенням, орієнтацією та швидкістю аудіоджерела у віртуальному 3D-просторі. Цей об'єкт дозволяє розробникам створювати ефект занурення в аудіо, коли звук рухається з певних напрямків, створюючи враження глибини та руху. Щоб створити PannerNode, можна використати код "panner = context.createPanner();". Після підключення до аудіовузла цей об'єкт може використовуватись для контролю положення та руху аудіоджерела, забезпечуючи динамічне розподілення звуку у просторі.

BiquadFilterNode реалізує різні типи цифрових фільтрів, такі як низькочастотні, високочастотні, смугові та пікові фільтри. Це дозволяє розробникам формувати частотну характеристику аудіосигналу, змінюючи його тембр і застосовуючи ефекти, такі як вирівнювання або резонанс. BiquadFilterNode надає параметри для керування частотою зрізу, посиленням і якістю фільтра. Приклад створення BiquadFilterNode: "biquadFilter = context.createBiquadFilter();". Після підключення до аудіовузла цей вузол можна використовувати для застосування ефектів фільтрації до аудіосигналу, покращуючи або змінюючи його спектральні характеристики.

Загалом, Web Audio API надає розробникам потужний набір об'єктів для маніпулювання та обробки звуку в веб-додатках. AudioContext виступає як головний інтерфейс, а MediaElementSourceNode, PannerNode і BiquadFilterNode надають спеціалізовані функції для отримання 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 аудіорішення в Інтернеті.

1. Особливості виконання завдання

Під час другої лабораторної роботи було реалізовано функціонал обертання об'єкта "Surface of Conjugation of Coaxial Cylinder and Cone" у стерео-режимі за допомогою сенсора смартфона. Поворот телефона відповідав обертанню фігури. Демонстрацію роботи програми можна побачити на Рис. 3.1.

A hand holding a cell phone

Description automatically generated with medium confidence

Рис. 3.1 Демонстрація застосування програми, отриманої в ході другої лабораторної

За допомогою Web Audio API, а саме документації представленої на сторінці https://webaudio.github.io/web-audio-api/ було імплементовано основну частину завдання розрахунково-графічної роботи.

В ході виконання лабораторної роботи необхідно було спочатку створити об’єкт аудіоконтексту, що дозволяє отримати доступ до Web Audio API.

Для виконання роботи було також обрано аудіо-файл формату mp3 і представлено його на веб-сторінці через HTML-елемент <audio>.

Наступним кроком було створити джерело аудіо передавши аудіо-елемент в конструктор.

Також необхідно було створити об’єкт panner в контексті, для подальшої маніпуляції звуком, зокрема позицією, що буде змінюватися по обертанню телефоном(джерело звуку буде знаходитися на умовній відстані 2 від центру в сторону відповідну повороту телефону в просторі).

Важливою частиною завдання було застосувати фільтр до вихідного звуку. За варіантом було імпелментовано Режекторний фільтр з параметрами вказанами в розділі 5.

Далі потрібно було поєднати об’єкту, передавші відповідні об’єкти іншим.

Було додано eventListener, що відповдає за зупинку та продовження програвання аудіо-файлу.

Крім цього необхідно було створити поле для увімкнення та вимкнення фільтру, а також додати інший eventListener для перемикання фільтра по перемиканню вище вказаного поля.

Оновлення позиції звуку через переміщення об’єкту panner було імплементовані в основній функції під назвою draw.

1. Вказівки користувачу

Користувач може керувати переміщенням умовної сфери, яка відображає умовне місцезнаходження джерела звуку. Переміщення сфери під час обертання телефону можна спостерігати на Рис. 4.1 і Рис. 4.2.

A screenshot of a computer

Description automatically generated with medium confidence A close up of a finger

Description automatically generated with medium confidence

Рисунок 4.1 та Рисунок 4.2

На Рис. 4.1 зображено сферу, що повернута догори на матеріальному інтерфейсі, тоді як на Рис. 4.2 вона повернута вбік. При обертанні телефону сфера рухається навколо фігури, створюючи ефект переміщення джерела звуку, особливо відчутний у навушниках та аудіо-стереосистемах.

Крім того, на сторінці присутні елементи інтерфейсу для зміни параметрів стерео-зображення, таких як значення eye separation, field of view, near clipping distance та convergence. Слайдери для цих параметрів можна побачити на Рис. 4.3.

A picture containing text, screenshot, font, line

Description automatically generated

Рисунок 4.3

Також на сторінці присутні елементи управління аудіофайлом, які дозволяють перемотувати, призупинити, відновити та регулювати гучність. Приклад елементу управління аудіофайлом показано на Рис. 4.4.

A picture containing text, font, screenshot, white

Description automatically generated

Рисунок 4.4

Додатково було створено елемент "чекбокс" для увімкнення та вимкнення фільтру, який можна побачити на Рис. 4.4.

1. Деякі зразки коду

Код функцій для налаштування та ініціалізації аудіо

function audioSet() {

  audio = document.getElementById('audio');

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

    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);

      // highshelf filter parameters

      filter.type = 'notch';

      filter.Q.value = 0.1;

      filter.frequency.value = 900;

      filter.gain.value = 9;

      audioContext.resume();

    }

  })

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

    console.log('pause');

    audioContext.resume();

  })

}

function audioFun() {

  audioSet();

  let filterCheckbox = document.getElementById('filterCheckbox');

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

    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);

    }

  });

  audio.play();

}

Код основної функції draw

function draw() {

  let D = document;

  let spans = D.getElementsByClassName("slider-value");

  gl.clearColor(0, 0, 0, 1);

  gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

  /\* Set the values of the projection transformation \*/

  let projection = m4.perspective(Math.PI / 32, 1, 8, 12);

  // let projection = m4.orthographic(-10, 10, -10, 10, -10, 10);

  let conv, // convergence

    eyes, // eye separation

    ratio, // aspect ratio

    fov; // field of view

  conv = 2000.0;

  conv = D.getElementById("conv").value;

  spans[3].innerHTML = conv;

  eyes = 70.0;

  eyes = D.getElementById("eyes").value;

  spans[0].innerHTML = eyes;

  ratio = 1.0;

  fov = Math.PI / 4;

  fov = D.getElementById("fov").value;

  spans[1].innerHTML = fov;

  let top, bottom, left, right, near, far;

  near = 5.0;

  near = D.getElementById("near").value - 0.0;

  spans[2].innerHTML = near;

  far = 2000.0;

  top = near \* Math.tan(fov / 2.0);

  bottom = -top;

  let a = ratio \* Math.tan(fov / 2.0) \* conv;

  let b = a - eyes / 2;

  let c = a + eyes / 2;

  left = -b \* near / conv;

  right = c \* near / conv;

  // console.log(left, right, bottom, top, near, far);

  let projectionLeft = m4.frustum(left, right, bottom, top, near, far);

  left = -c \* near / conv;

  right = b \* near / conv;

  let projectionRight = m4.frustum(left, right, bottom, top, near, far);

  /\* Get the view matrix from the SimpleRotator object.\*/

  let modelView = spaceball.getViewMatrix();

  let rotateToPointZero = m4.axisRotation([0.0, 0, -1.0], 0);

  let translateToPointZero = m4.translation(0, 0, -10);

  let translateToLeft = m4.translation(-0.03, 0, -10);

  let translateToRight = m4.translation(0.03, 0, -10);

  let matAccum0 = m4.multiply(rotateToPointZero, modelView);

  let matAccum1 = m4.multiply(translateToPointZero, matAccum0);

  let matAccum2 = m4.multiply(m4.axisRotation([0, 1, 0], 0.5 \* Math.PI \* aclVec.x \* 0.1), m4.axisRotation([1, 0, 0], -0.5 \* Math.PI \* aclVec.y \* 0.1))

  let matAccumLeft = m4.multiply(translateToLeft, matAccum0);

  let matAccumRight = m4.multiply(translateToRight, matAccum0);

  /\* 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);

  gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, modelViewProjection);

  let modelviewInv = new Float32Array(16);

  let normalmatrix = new Float32Array(16);

  mat4Invert(modelViewProjection, modelviewInv);

  mat4Transpose(modelviewInv, normalmatrix);

  gl.uniformMatrix4fv(shProgram.iNormalMatrix, false, normalmatrix);

  /\* Draw the six faces of a cube, with different colors. \*/

  gl.uniform4fv(shProgram.iColor, [0.2, 0.8, 0, 1]);

  gl.uniform3fv(shProgram.lightPosLoc, [20 \* Math.cos(Date.now() \* 0.005), 1, 20 \* Math.sin(Date.now() \* 0.005)]);

  gl.uniform1i(shProgram.iTMU, 0);

  gl.enable(gl.TEXTURE\_2D);

  gl.uniform1f(shProgram.iScale, -1.0)

  let matStill = m4.multiply(rotateToPointZero, [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]);

  let matAccumStill = m4.multiply(translateToPointZero, matStill);

  let translateWebCam = m4.translation(-0.5, -0.5, 0);

  let matAccumStill1 = m4.multiply(translateWebCam, matAccumStill);

  modelViewProjection = m4.multiply(projection, matAccumStill1);

  // gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, m4.multiply(m4.scaling(2, 2, 1), modelViewProjection));

  gl.bindTexture(gl.TEXTURE\_2D, textureCam);

  gl.texImage2D(

    gl.TEXTURE\_2D,

    0,

    gl.RGBA,

    gl.RGBA,

    gl.UNSIGNED\_BYTE,

    video

  );

  gl.uniform1f(shProgram.iScale, 1.0)

  gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, m4.multiply(m4.scaling(10, 10, 1), matAccumStill1));

  tris.DrawTris();

  gl.bindTexture(gl.TEXTURE\_2D, texture);

  gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccumRight);

  gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projectionRight);

  gl.colorMask(true, false, false, false);

  surface.Draw();

  gl.clear(gl.DEPTH\_BUFFER\_BIT);

  gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccumLeft);

  gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projectionLeft);

  gl.colorMask(false, true, true, false);

  gl.uniform4fv(shProgram.iColor, [0.5, 0.3, 1.0, 1]);

  surface.Draw();

  gl.colorMask(true, true, true, true);

  gl.uniform1f(shProgram.iScale, -1.0)

  // let a = 2 - 1;

  // let c = -2 \* Math.PI \* a / Math.tan(-0.5);

  // let b = 3 \* c / 4;

  let trS = {

    x: (-0.5) \* aclVec.x \* 0.1,

    y: (-0.5) \* aclVec.y \* 0.1,

    z: (-0.5) \* aclVec.z \* 0.1

  }

  gl.uniform3fv(shProgram.iUP, [trS.x \* 2.0, trS.y \* 2.0, trS.z \* 2.0]);

  if (panner) {

    panner.setPosition(trS.x, trS.y, trS.z)

  }

  sphere.DrawPoint();

  window.requestAnimationFrame(draw)

}