Міністерство освіти й науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра цифрових технологій в енергетиці

[Методи синтезу віртуальної реальності](https://campus.kpi.ua/student/index.php?mode=studysheet&action=view&id=125230)

Розрахунково графічна робота

Варіант - 2

**Виконав:**

Студент 1-го курсу

групи ТР-21мп НН ІАТЕ

Барчук Роман

**Перевірив:**

Демчишин А.А.

Київ-2023

**Завдання**

1. Використати код з практичного заняття №2.

2. Імплементувати обертання джерела звуку навколо геометричного центру площини за допомогою фізичного (матеріального) інтерфейсу (на цей раз поверхня залишається нерухомою, а джерело звуку рухається). Відтворіти улюблену пісню у форматі mp3/ogg з контролем користувача над просторовим положенням джерела звуку.

3. Візуалізувати положення джерела звуку за допомогою сфери.

4. Додати звуковий фільтр (використовуючи інтерфейс BiquadFilterNode) для відповідного варіанту (№2). Додати чекбокс(checkbox), який вмикає або вимикає фільтр. Встановити параметри фільтра на свій смак.

**Теоретичні відомості**

Web Audio API є потужним інструментом для роботи з аудіо веб-додатками. Цей API, який був введена в стандарт HTML5, надає розробникам можливість створювати, маніпулювати та відтворювати звукові ефекти в реальному часі у веб-середовищі.

Web Audio API включає в себе обробку аудіооперацій всередині аудіоконтексту. Основні аудіооперації виконуються за допомогою аудіовузлів, які пов’язані між собою, щоб сформувати граф маршрутизації аудіо. Кілька джерел — із різними типами компонування каналів — підтримуються навіть в одному контексті. Ця модульна конструкція забезпечує гнучкість для створення складних аудіо функцій з динамічними ефектами.

Основні концепції Web Audio API:

1. Аудіо-контекст (AudioContext): Це головний об'єкт API, який ініціалізує та керує аудіо-системою. Аудіо-контекст визначає аудіо-середовище та містить всі необхідні ресурси для створення, маніпулювання та відтворення звуку.

2. Аудіо-вузли (Audio Nodes): Аудіо-вузли представляють оброблювальні блоки, які обробляють аудіо-сигнали. Вони можуть бути з'єднані між собою для створення аудіо-графів. Деякі типи аудіо-вузлів включають генератори звуку, ефекти, аналізатори та вихідні вузли.

3. Роутинг аудіо (Audio Routing): З'єднання аудіо-вузлів утворює аудіо-граф, що визначає потік аудіо-сигналу від джерела до виходу. Це дозволяє створювати складні маршрутизації сигналу та контролювати його потік.

4. Відтворення звуку: Web Audio API дозволяє завантажувати аудіо-файли та відтворювати їх за допомогою аудіо-вузлів. Ви можете контролювати гучність, панораму та інші параметри відтворення.

5. Ефекти та обробка звуку: API надає набір ефектів та фільтрів, які можна застосовувати до аудіо-сигналу. Це дозволяє створювати різноманітні звукові ефекти та обробки, такі як реверберація, еквалайзери, затримки та інші.

Інтерфейс відкриває широкі можливості для створення інтерактивних та звуко-насичених веб-додатків.

Web Audio API надає можливість застосовувати звукові фільтри до аудіо-сигналу. Один з інтерфейсів, які використовуються для цього, - це BiquadFilterNode. Цей інтерфейс дозволяє налаштовувати параметри фільтра, такі як тип фільтра, частоту зрізу, підсилення та інші.

Основні кроки для роботи з фільтрами у Web Audio API:

1. Створіть аудіо-контекст (AudioContext) та завантажте аудіо-файл.

2. Створіть BiquadFilterNode, використовуючи метод createBiquadFilter() аудіо-контексту.

3. Налаштуйте параметри фільтра, такі як тип фільтра (наприклад, lowpass, highpass, bandpass), частота зрізу, підсилення та інші.

4. Підключіть аудіо-вузли (наприклад, AudioBufferSourceNode) до фільтра, а потім до вихідного вузла для відтворення фільтрованого звуку. (рисунок 1)

5. Запустіть аудіо-відтворення та налаштуйте контроли для зміни параметрів фільтра (наприклад кнопки, повзунки), щоб користувач міг налаштувати звук за своїм смаком.

6. Додайте чекбокс (checkbox), який дозволяє користувачеві увімкнути або вимкнути фільтр за потреби.



Рисунок 1 – абстрактна схема підключення аудіо-вузлів

Інтерфейс Web Audio API також дозволяє нам контролювати, як аудіо розподіляється в просторі. Використовуючи систему, побудовану на моделі джерело-слухач, вона дозволяє керувати просторовою моделлю та регулювати звук залежно від відстані, зміна якої спричинена рухом джерела звуку.

**Виконання роботи**

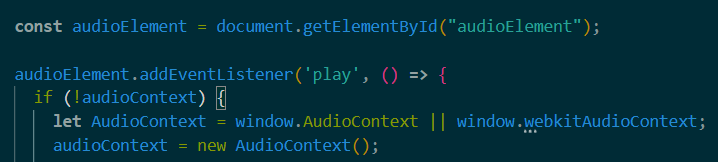
1. **Система контролю версій**Перед початком роботи було створено гілку в GIT з назвою CGW, для зручного зберігання та контролю версій програми.
2. **Audio Context**Спочатку було створено Аудіо Контекст. Через обмеження авто-відтворення аудіо в сучасних браузерах, контекст не може бути створений без попередньої взаємодії користувача з програмою. Тому було створено eventListener (рисунок 2), який ініціалізує контекст лише після натискання кнопки програвання, яке є частиною елементу <audio> та активується, якщо елементу присвоїти параметр controls (рисунок 3). 

Рисунок 2 **–** ініціалізація аудіо контексту

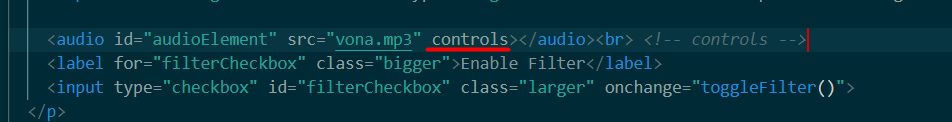


Рисунок 3 – створення інтерфейсу користувача

Було присвоєно вихідні параметри джерелу звуку, як от позиція. Створені вузли – джерело та просторовий. Після цього вузли було з’єднано і створено своєрідний ланцюжок виклику Джерело – Просторовий вузол – Назначення (звукові пристрої фізичного інтерфейсу). Опісля запускаємо програвання звуку.

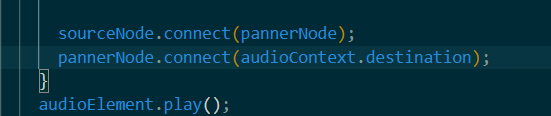


Рисунок 4 – підключення вузлів

1. **Джерело звуку**Сфера – джерело звуку, була створена використовуючи функцію createSurface з попередніх робіт. В буфер передаються вертекси, нормалі та текстури з функції createSphere. Сфера переміщується по колу, на деякій відстані від фігури. Контролюється за допомогою магнітометру. При нахилі матеріального інтерфейсу обчислюється нова матриця обертання, яка одразу ж множиться на матрицю переміщення. Сфера рухається, а в функції moveModelCGWRotationMatrix джерелу звуку задаються нові координати шляхом. Вони обчислюються в цій функції та оновлюються у вузлі за допомогою функції setPosition.
2. **Фільтр**Для створення фільтру високих частот було ініціалізовано вузол фільтрування, зазначено його тип – високочастотний, да частоту обрізання й резонанс.

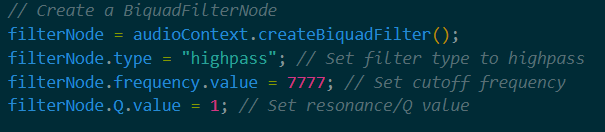


Рисунок 5 – налаштування фільтру

Для Роботи фільтру було створено відповідний чекбокс да подію. При перемиканні чекбоксу на «увімкнено» просторовий вузол відключається, підключається до фільтраційного вузла, а той в свою чергу до назначення (звукового пристрою, destination). При переключенні на стан «вимкнено» відбувається зворотній процес.

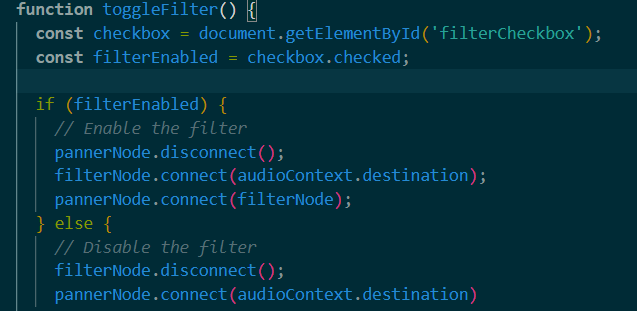


Рисунок 6 – імплементація події зміни стану чекбоксу

**Результат роботи**

На рисунку зображено фігуру та сферу, елементи інтерфейсу - результат виконання розрахунково графічної. Сфера є джерелом звуку та контролюється нахилом матеріального інтерфейсу вперед та назад. Сфера ближче до користувача (тобто знизу) – звук лунає гучніше з основного динаміку.

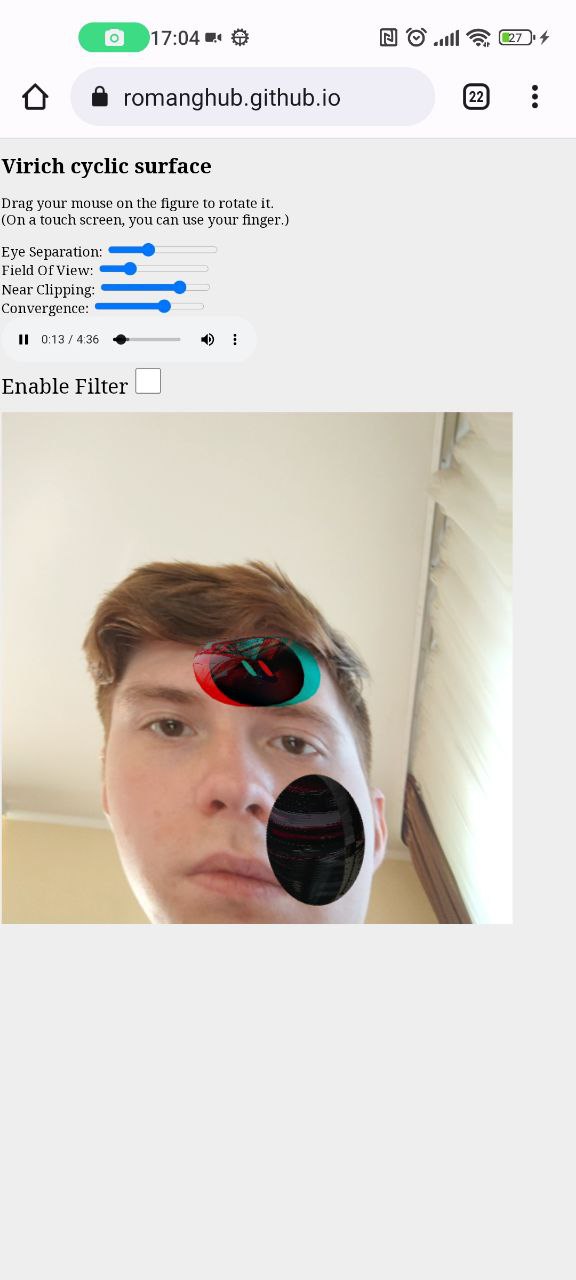


Рисунок 7 - Сфера знизу

На рисунку 8 сфера знаходиться далі від користувача (тобто зверху) – звук лунає гучніше з розмовного динаміку.

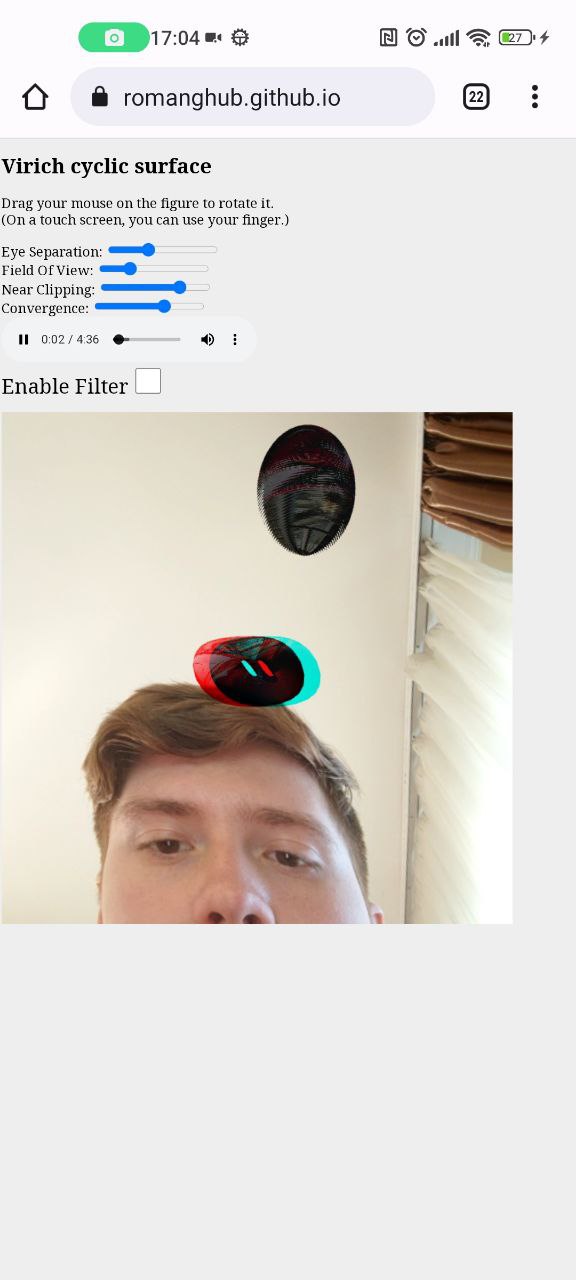


Рисунок 8 – Сфера зверху

Під час роботи програми все ще можна змінювати параметри основної фігури. Натискання на чекбокс вмикає й вимикає фільтр високих частот. Це зображено на рисунку 9.

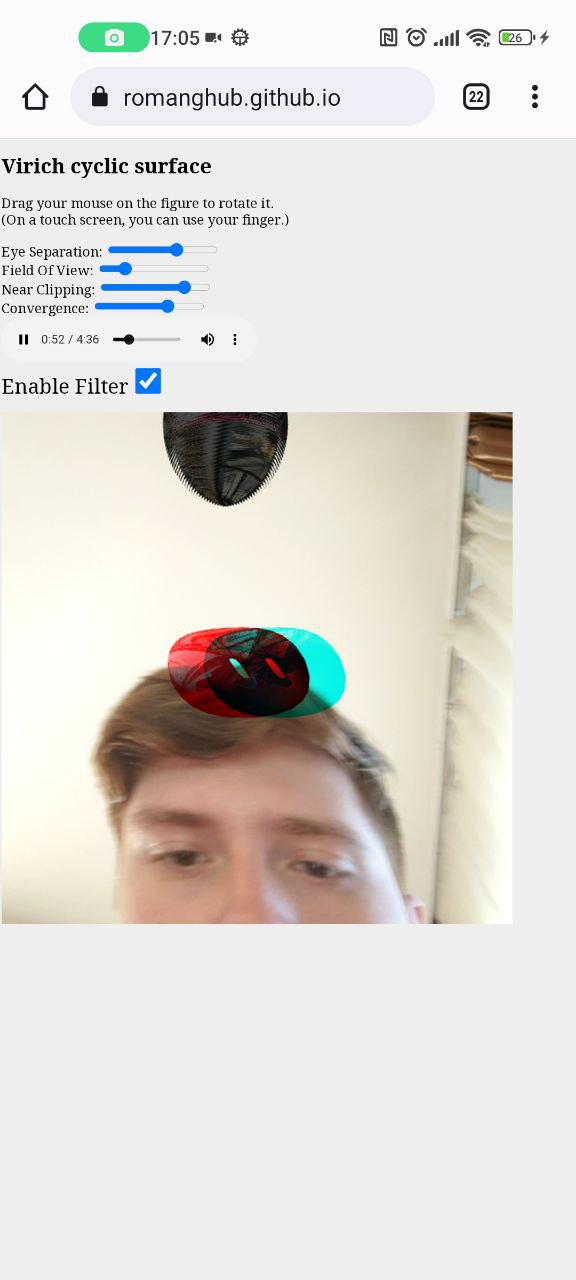


Рисунок 9 – Поверхня зі зміненими параметрами, чекбокс активований

**Код**

let audioContext;

let sourceNode;

let pannerNode;

let filterNode;

let audioElement;

let playButton;

let listener;

let sphere;

let sphereTexture;

function initGL(){

……

….

const audioElement = document.getElementById("audioElement");

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

    if (!audioContext) {

      let AudioContext = window.AudioContext || window.webkitAudioContext;

      audioContext = new AudioContext();

      listener = audioContext.listener;

      listener.positionX.value = 0;

      listener.positionY.value = 0;

      listener.positionZ.value = -5;

      listener.forwardX.value = 0;

      listener.forwardY.value = 0;

      listener.forwardZ.value = -1;

      sourceNode = audioContext.createMediaElementSource(audioElement);

      pannerNode = audioContext.createPanner();

      // Connect audio nodes and set up spatial audio properties

      // Create a BiquadFilterNode

      filterNode = audioContext.createBiquadFilter();

      filterNode.type = "highpass"; // Set filter type to highpass

      filterNode.frequency.value = 7777; // Set cutoff frequency

      filterNode.Q.value = 1; // Set resonance/Q value

      sourceNode.connect(pannerNode);

      pannerNode.connect(audioContext.destination);

    }

    audioElement.play();

  });

}

function toggleFilter() {

  const checkbox = document.getElementById('filterCheckbox');

  const filterEnabled = checkbox.checked;

  if (filterEnabled) {

    // Enable the filter

    pannerNode.disconnect();

    filterNode.connect(audioContext.destination);

    pannerNode.connect(filterNode);

  } else {

    // Disable the filter

    filterNode.disconnect();

    pannerNode.connect(audioContext.destination)

  }

}

  sphere = new Model('Sphere');

  sphere.BufferData(createSphere(1, 70, 70));

  LoadSphereTexture();

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

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

  gl.uniform1f(shProgram.iL, 10)

  gl.uniform1i(shProgram.iTexture, 0);

  gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, m4.multiply(projection, moveModelCGWRotationMatrix(calculateSurfaceRotation())));

  sphere.Draw();

  gl.clear(gl.DEPTH\_BUFFER\_BIT);

function LoadSphereTexture() {

  sphereTexture = gl.createTexture();

  let image = new Image();

  image.src = 'https://images.pexels.com/photos/1545743/pexels-photo-1545743.jpeg?cs=srgb&dl=pexels-yurii-hlei-1545743.jpg&fm=jpg';

  image.crossOrigin = 'anonymous';

  image.onload = () => {

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

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.LINEAR);

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MAG\_FILTER, gl.LINEAR);

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_T, gl.CLAMP\_TO\_EDGE);

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_S, gl.CLAMP\_TO\_EDGE);

    gl.pixelStorei(gl.UNPACK\_FLIP\_Y\_WEBGL, 1);

    gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, image);

    draw();

  };

}