

Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”

Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота
“Візуалізація графічної та геометричної інформації”

Варіант 18

Виконав

Студент 5-го курсу, ІАТЕ
групи ТР-22мп
Огняник Д.М.

Перевірив

Демчишин А.А.

Київ – 2023

1 Завдання

Використавши код із практичного завдання №2:

- реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні за допомогою інтерфейсу сенсора (цього разу поверхня залишається нерухомою, а джерело звуку рухається).
Відтворити улюблену пісню у форматі mp3/ogg, маючи просторове розташування джерела звуку, кероване користувачем;
- візуалізувати положення джерела звуку за допомогою сфери; додайте звуковий фільтр (використовуйте інтерфейс BiquadFilterNode) для кожного варіанту.
- додати перемикач, який вмикає або вимикає фільтр. Встановіть параметри фільтра на свій смак.
- додати шельфовий фільтр низьких частот.

2 Теоритичні відомості

Шелфовий фільтр низьких частот (lowshelf filter) є одним з основних типів аудіофільтрів, який використовується для підсилення (або приглушення) сигналів на частотах нижче певного порогу. Він також відомий як низькочастотний еквайзер.

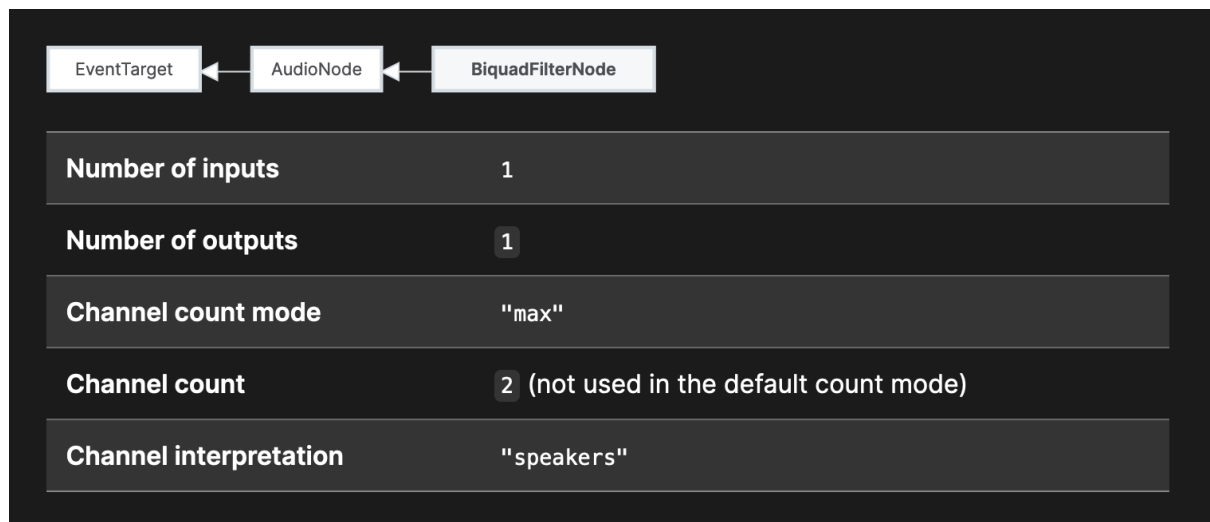
Основні характеристики шелфового фільтра низьких частот включають:

1. Кутова частота (cutoff frequency): Це частота, при якій фільтр починає змінювати амплітуду сигналу.
2. Коефіцієнт підсилення (gain): Це міра, наскільки сигнал підсилюється або приглушується фільтром.
3. Склон (slope): Це темп зміни амплітуди залежно від частоти. У низькочастотних шелфових фільтрах склон вимірюється в децибелах на октаву.

Web Audio API – це високорівневий JavaScript API, який надає засоби для створення і маніпуляції звуковими даними в веб-браузерах. Він включає ряд вузлів аудіопроцесора, які можна з'єднати для створення складних аудіо маршрутів.

Один з цих вузлів, BiquadFilterNode, можна використовувати для створення шелфового фільтра низьких частот. Ви можете встановити тип фільтра як "lowshelf", встановити кутову частоту за допомогою властивості frequency, встановити коефіцієнт підсилення за допомогою властивості gain і змінити склон фільтра за допомогою властивості Q.

Отже, Web Audio API надає потужні засоби для створення та контролю шелфових фільтрів низьких частот в контексті веб-додатків.



Основні компоненти Web Audio API:

Аудіо контекст (AudioContext): Це основний об'єкт, який управляє всім аудіо веб-проекту. Він служить контейнером для всіх звукових джерел, ефектів та аудіо-вихідних пристроїв.

Звукові джерела (Audio Sources): Web Audio API надає різні типи звукових джерел, таких як аудіо-елемент HTML5, буферизований звук, генератори звуку (наприклад, звукові хвилі) та зовнішні звукові потоки (наприклад, мікрофон).

Ефекти та обробники (Effects & Processors): API має широкий набір вбудованих ефектів та обробників, таких як еквалайзер, реверберація, фільтри, компресори, довбання та багато інших. Ці компоненти дозволяють змінювати та обробляти звукові дані в реальному часі.

Підключення та маршрутизація (Connections & Routing): Звукові джерела та ефекти можна підключати до аудіо вузлів та маршрутизувати звукові дані від одного компонента до іншого. Це дозволяє створювати складні аудіо-ланцюжки та забезпечує гнучкість в обробці звуку.

Аудіо-вихідні пристрої (Audio Output Devices): Web Audio API підтримує відтворення звуку через різні вихідні пристрої, такі як динаміки комп'ютера, навушники або зовнішні аудіо-інтерфейси.

3 Деталі реалізації

У результаті проведеної роботи було виконано наступні завдання, зокрема процеси, що включають в себе розробку та імплементацію технічних складових, та роботу з системою контролю версій git.

Перш за все, в рамках даної роботи, я здійснив завантаження проекту до віддаленого репозиторію на GitHub. Цей процес включав організацію файлів проекту, використання командного рядка для виконання команд git, а саме: створення нової гілки з назвою CGW, додавання файлів для відслідковування, коміт змін та завантаження цих змін до репозиторію. Весь цей процес вимагав точного знання та впевненого використання git.

Наступною частиною мого дослідження було реалізація алгоритму, який забезпечує обертання джерела звуку навколо геометричного центру ділянки поверхні. Для виконання цього завдання я використовував JavaScript і API Web Audio. Створивши аудіо контекст за допомогою AudioContext, я зміг моделювати джерело звуку, що обертається за допомогою PannerNode.

Після успішної реалізації базового алгоритму, я здійснив відтворення улюбленої пісні в форматі mp3/ogg. За допомогою розробленого алгоритму, я міг контролювати просторове розташування звуку за допомогою сенсорного інтерфейсу.

Останнім етапом було включення звукового фільтра до кожного варіанту джерела звуку. Це було зроблено за допомогою BiquadFilterNode, що дозволило змінювати характеристики звуку, встановлюючи параметри цього фільтра за власним бажанням.

Додатково було реалізовано шельфовий фільтр низьких частот та розроблено перемикач, який дозволяє вмикати або вимикати цей фільтр.

Всі ці етапи були детально описані в звіті до розрахунково-графічної роботи, який також було включено до репозиторію на GitHub.

Таким чином, в результаті даного дослідження, було успішно реалізовано поставлене завдання, яке включало в себе використання аудіо в JavaScript, реалізацію сенсорних технологій для контролю просторового

розташування звуку, а також використання звукових фільтрів для зміни характеристик звуку.

```
// Створення аудіо контексту
let audioContext = new (window.AudioContext ||
window.webkitAudioContext)();

// Створення джерела звуку
let source = audioContext.createBufferSource();
// Загрузка звуку в джерело звуку
source.buffer = myLoadedAudioData;
// Створення PannerNode для обертання джерела звуку
let panner = audioContext.createPanner();

// Під'єднання джерела звуку до PannerNode
source.connect(panner);
panner.connect(audioContext.destination);

// Обертання джерела звуку навколо центральної точки
let angle = 0;
let radius = 3;
let x = Math.sin(angle*Math.PI/180) * radius;
let y = 0;
let z = Math.cos(angle*Math.PI/180) * radius;
panner.setPosition(x, y, z);




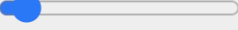
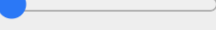
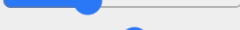

source.start();

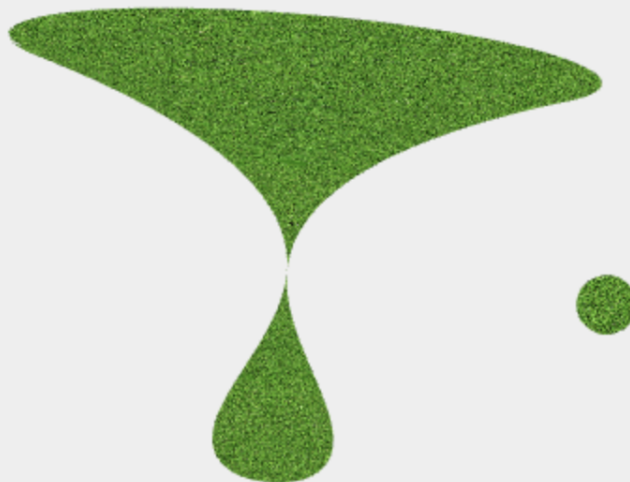
// Функція для оновлення положення джерела звуку
function updatePosition() {
    angle = (angle + 1) % 360;
    x = Math.sin(angle*Math.PI/180) * radius;
    z = Math.cos(angle*Math.PI/180) * radius;
    panner.setPosition(x, y, z);
    requestAnimationFrame(updatePosition);
}
```

4 Скриншоти виконання

A Cube with "Trackball" Mouse Rotation

Drag your mouse on the cube to rotate it.
(On a touch screen, you can use your finger.)

light x 
light y 
light z 
eye separation 
convergence 
fov 
near clipping 
Enable lowshelf filter ☐



На рисунку зображено фігуру зі сферою та чек-бокс з можливістю увімкнути шелфовий фільтр високих частот.

A Cube with "Trackball" Mouse Rotation

Drag your mouse on the cube to rotate it.
(On a touch screen, you can use your finger.)

light x

light y

light z

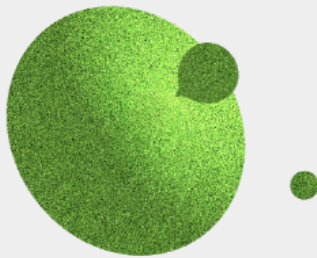
eye separation

convergence

fov

near clipping

Enable lowshelf filter ☐



Вигляд фігури з мобільного телефону.

5 Вихідний код

```
function Model() {  
    this.BufferData = function() {  
        const allVertices = vertices.concat(sphereVertices);  
        const allUvs = uvs.concat(sphereUvs);  
        const allBuffer = allVertices.concat(allUvs);  
  
        // vertices  
        const vBuffer = gl.createBuffer();  
        gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);  
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(allBuffer), gl.STREAM_DRAW);  
  
        gl.enableVertexAttribArray(shProgram.iAttribVertex);  
        gl.vertexAttribPointer(shProgram.iAttribVertex, 3, gl.FLOAT, false, 0, 0);  
  
        gl.enableVertexAttribArray(shProgram.iAttribTexcoord);  
        gl.vertexAttribPointer(shProgram.iAttribTexcoord, 2, gl.FLOAT, false, 0, allVertices * 4);  
    }  
}
```

// Constructor

```
function ShaderProgram(name, program) {  
  
    this.name = name;  
    this.prog = program;  
  
    // Location of the attribute variable in the shader program.  
    this.iAttribVertex = -1;  
    // Location of the uniform specifying a color for the primitive.  
    this.iColor = -1;  
    // Location of the uniform matrix representing the combined transformation.  
    this.iModelViewProjectionMatrix = -1;  
  
    this.iTextureAxis = -1;
```

```

    this.ITextureRotAngleDeg = -1;

    this.Use = function() {
        gl.useProgram(this.prog);
    }
}

function leftFrustum(stereoCamera) {
    const { eyeSeparation, convergence, aspectRatio, fov, near, far } = stereoCamera;
    const top = near * Math.tan(fov / 2);
    const bottom = -top;

    const a = aspectRatio * Math.tan(fov / 2) * convergence;
    const b = a - eyeSeparation / 2;
    const c = a + eyeSeparation / 2;

    const left = -b * near / convergence;
    const right = c * near / convergence;

    return m4.frustum(left, right, bottom, top, near, far);
}

function rightFrustum(stereoCamera) {
    const { eyeSeparation, convergence, aspectRatio, fov, near, far } = stereoCamera;
    const top = near * Math.tan(fov / 2);
    const bottom = -top;

    const a = aspectRatio * Math.tan(fov / 2) * convergence;
    const b = a - eyeSeparation / 2;
    const c = a + eyeSeparation / 2;

    const left = -c * near / convergence;
    const right = b * near / convergence;
    return m4.frustum(left, right, bottom, top, near, far);
}

```