МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря Сікорського»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра ЦТЕ

ЗВІТ
про виконання графічно-розрахункової роботи
з дисципліни: «Методи синтезу віртуальної реальності»
На тему: «Просторове аудіо»
Варіант №3

Виконав:
студент групи ТР-23мп
Бондарчук О.О.

Перевірив:
Демчишин А.А.

Київ 2023

Постановка задачі

Варіант: 3 – реалізувати смуговий фільтр.

Мета: засвоїти навички роботи з просторовим аудіо та **WebAudio HTML5 API**.

Вимоги:

- Повторно використати код із практичного завдання №2.
- Реалізувати обертання джерела звуку навколо геометричного центра поверхні за допомогою інтерфейсу. Цього разу поверхня залишається нерухомою, а джерело звуку переміщується. Відтворити улюблену пісню у форматі mp3/ogg з підтримкою просторового положення звуку, яким можна керувати.
- Візуалізувати положення джерела звуку сферою.
- Додати звуковий фільтр за допомогою інтерфейсу **BiquadFilterNode**. Додати чекбокс елемент, який вмикає та вимикає цей фільтр. Задати параметри для фільтру на свій смак.

Звіт повинен містити:

- титульну сторінку;
- розділ з постановкою задачі;
- розділ з теоретичними відомостями;
- розділ з описом деталей реалізації;
- розділ з інструкціями користувача зі скріншотами;
- зразок вихідного коду.

Теоретичні відомості

Web Audio API є могутнім інструментом для обробки та синтезу звуку веб-додатками. Дозволяючи доступ до низькорівневих аудіоопераций, він надає розширені можливості зміни, маніпулювання та створення аудіоекспериментів у браузері. Давайте розглянемо деякі ключові елементи Web Audio API:

1. **AudioContext**: Це головний об'єкт, який представляє аудіоконтекст. Він відповідає за створення, керування та координацію різних елементів аудіо. Щоб почати роботу з Web Audio API, спочатку потрібно створити екземпляр "AudioContext". Наприклад:

```
const audioContext = new AudioContext();
```

2. **Audio Sources**: Ви можете створювати аудіо джерела, такі як аудіофайли або захоплення звуку з мікрофону, за допомогою різних об'єктів, таких як "AudioBufferSourceNode" або "MediaStreamAudioSourceNode".

   Перший використовується для відтворення аудіоданих з попередньо завантаженого буфера. Ось приклад створення "AudioBufferSourceNode":

```
const audioBufferSource = audioContext.createBufferSource();
audioBufferSource.buffer = myAudioBuffer; // Завантажений аудіобуфер
audioBufferSource.connect(audioContext.destination);
audioBufferSource.start();
```

3. **Panner**: Цей об'єкт відповідає за панування звуку в просторі. Ви можете переміщувати звукові джерела вліво або вправо, регулювати їх відстань та висоту. Це корисно для створення просторового звучання. Ось як створити "PannerNode":

```
const panner = audioContext.createPanner();
panner.setPosition(x, y, z); // Встановлення позиції джерела звуку
panner.connect(audioContext.destination);
```

4. **BiquadFilterNode**: Цей об'єкт дозволяє застосовувати фільтрацію на аудіосигналі. Він має різноманітні типи фільтрів, такі як низькочастотний фільтр, високочастотний фільтр, піковий фільтр тощо.

Використовувати “BiquadFilterNode” можна так:

```
const filter = audioContext.createBiquadFilter();
filter.type = 'lowpass'; // Встановлення типу фільтра
filter.frequency.value = 1000; // Встановлення частоти фільтра
filter.connect(audioContext.destination);
```

5. **Band-pass filter**: Це один з типів фільтрів, доступних за допомогою “BiquadFilterNode”. Він пропускає сигнали, які знаходяться в межах певного діапазону частот. Щоб налаштувати “BiquadFilterNode” як band-pass фільтр, ви можете встановити тип фільтра на “bandpass” та встановити “frequency” та “Q” параметри для визначення діапазону частот, які пропускаються фільтром. Ось приклад:

```
const bandPassFilter = audioContext.createBiquadFilter();
bandPassFilter.type = 'bandpass'; // Встановлення типу фільтра
bandPassFilter.frequency.value = 1000; // Встановлення частоти фільтра
bandPassFilter.Q.value = 10; // Визначення діапазону пропускання
bandPassFilter.connect(audioContext.destination);
```

Окрім вже згаданих параметрів, Web Audio API надає можливість налаштувати різні інші параметри для контролю звуку. Ось кілька прикладів:

1. **Gain**: Об’єкт “GainNode” дозволяє керувати гучністю аудіосигналу. За допомогою параметра “gain” можна змінювати гучність в діапазоні від 0 до 1, де 0 означає беззвучність, а 1 - максимальну гучність. Наприклад:

```
const gainNode = audioContext.createGain();
gainNode.gain.value = 0.5; // Зміна гучності на половину
```

2. **Довжина затримки**: Об’єкт “DelayNode” дозволяє створювати затримку аудіосигналу. Ви можете встановити параметр “delayTime” для визначення тривалості затримки в секундах. Наприклад:

```
const delayNode = audioContext.createDelay();
delayNode.delayTime.value = 1; // Затримка на 1 секунду
```

3. **Реверберація**: Об’єкт “ConvolverNode” використовується для створення ефекту реверберації, який додає просторову глибину до звуку. Ви можете завантажити аудіофайл із звуком реверберації та встановити його як імпульсну відповідь для об’єкта “ConvolverNode”. Наприклад:

```
const convolverNode = audioContext.createConvolver();
const impulseResponse = await loadAudioFile('reverb.wav');
convolverNode.buffer = impulseResponse;
```

Опис деталей реалізації

Було створено сферу для візуалізації джерела аудіо.

```
function CreateSphereSurface(r = 0.15) {
    let vertexList = [];
    for (let lon = -Math.PI; lon < Math.PI; lon += 0.5) {
        for (let lat = -Math.PI * 0.5; lat < Math.PI * 0.5; lat += 0.5) {
            vertexList.push(
                ...sphereSurfaceData(r, lon, lat),
                ...sphereSurfaceData(r, lon + 0.5, lat),
                ...sphereSurfaceData(r, lon, lat + 0.5),
                ...sphereSurfaceData(r, lon + 0.5, lat + 0.5),
                ...sphereSurfaceData(r, lon + 0.5, lat),
                ...sphereSurfaceData(r, lon, lat + 0.5)
            );
        }
    }
    return vertexList;
}

function sphereSurfaceData(r, u, v) {
    let x = r * Math.sin(u) * Math.cos(v);
    let y = r * Math.sin(u) * Math.sin(v);
    let z = r * Math.cos(u);
    return [x, y, z];
}
```

Функція CreateSphereSurface генерує вершини для побудови сфери з заданим радіусом r. Вона використовує два цикли for для обходу довготи (lon) та широти (lat). Для кожної комбінації значень lon та lat, вона викликає функцію sphereSurfaceData, яка обчислює координати вершини на сфері за допомогою сферичних координат і додає ці координати до списку vertexList. Функція повертає отриманий список вершин. Функція sphereSurfaceData обчислює координати вершини на сфері з заданими радіусом r, широтою u та довготою v. Вона використовує математичні формули для перетворення сферичних координат в декартові координати і повертає масив з трьох значень [x, y, z], що представляють координати вершини. Функція DrawSphere відповідає за візуалізацію сфери. Перед відрисовкою сфери на неї накладається текстура, яка попередньо завантажується у функції LoadSphereTexture:

```
function LoadSphereTexture() {
    SphereTexture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, SphereTexture);

    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA,
    gl.UNSIGNED_BYTE,
        new Uint8Array([0, 0, 255, 255]));

    const SphereImage = new Image();
    SphereImage.crossOrigin = "anonymus";
    SphereImage.onload = () => {
```

```
            gl.bindTexture(gl.TEXTURE_2D, SphereTexture);
            gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
        SphereImage);
            gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S,
        gl.CLAMP_TO_EDGE);
            gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T,
        gl.CLAMP_TO_EDGE);
            gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
            draw();
        };
        SphereImage.src = "./texture/gold.avif";
    }
```

Після цього сфера рендериться у функції draw. Також було додано обробники подій для елемента аудіо, який попередньо був доданий в html:

```
audioPanner.panningModel = "HRTF";
audioPanner.distanceModel = "linear";
audioFilter.type = "bandpass";
audioFilter.frequency.value = centerFrequencyInput;
audioFilter.Q.value = Q_value;

audio.volume = Volume;
audio.playbackRate = PlaybackRate;

const reverbUrl = "audio/reverb/BIG_HALL_E001_M2S.wav";
const reverbRequest = new XMLHttpRequest();
reverbRequest.open("GET", reverbUrl, true);
reverbRequest.responseType = "arraybuffer";
```

При програванні аудіо відтворюється звуковий контекст audioContext, створюються аудіо-джерело audioSource, панорамування audioPanner, біквадратичний фільтр audioFilter та об’єкт reverbNode для ефекту реверберації. Звуковий контекст продовжується (resume), аудіо-джерело підключається до панорамування, панорамування підключається до фільтра, фільтр підключається до об’єкта реверберації, а об’єкт реверберації підключається до призначення звуку (audioContext.destination). Також є обробники подій для перемикання фільтрації та реверберації. При перемиканні фільтрації аудіо-джерело відключається від панорамування та підключається до фільтра або безпосередньо до призначення звуку. При перемиканні реверберації фільтр відключається або підключається до об’єкта реверберації, а об’єкт реверберації підключається або відключається від призначення звуку.

Інструкція користувача

Розроблений веб-додаток (https://bondar4uk.github.io/MSVR) має наступний вигляд (Рисунок 1). Зверху знаходиться опис, потім різний набір елементів для зміни параметрів фігури, заднього фону, сфери та звуку.

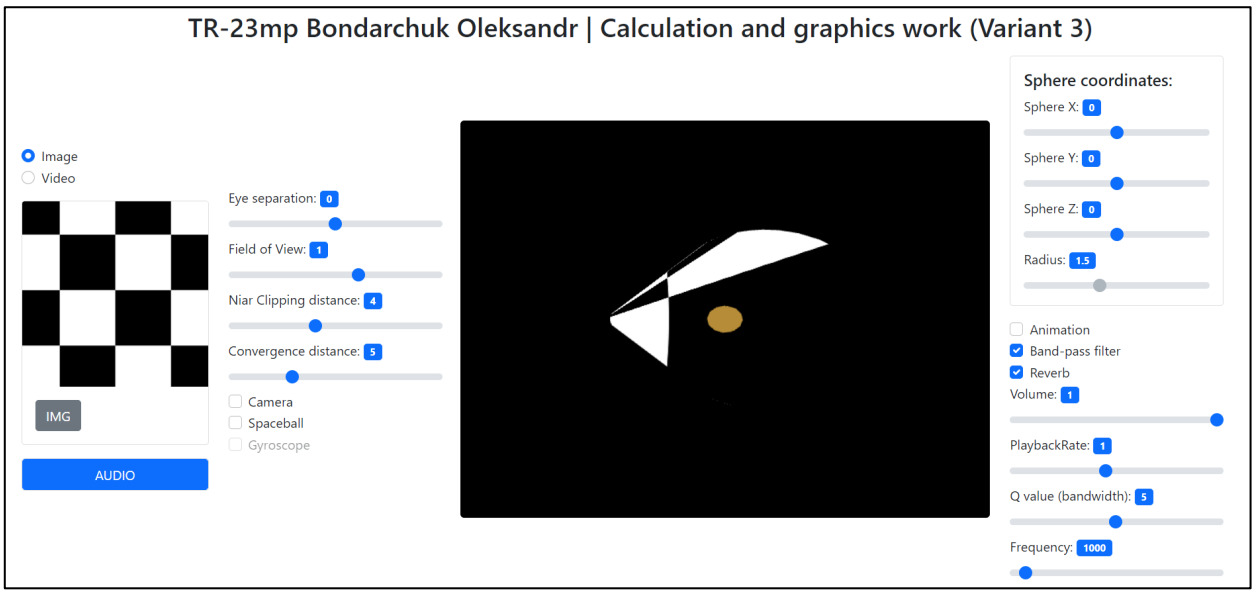

Рисунок 1. Сторінка веб-додатку.

1. **Вибір типу вмісту** (зображення або відео):
    - Оберіть одну з двох опцій: “Image” (Зображення) або “Video” (Відео) за допомогою радіокнопок.
    - Якщо обрано “Image”. Натисніть на кнопку “IMG” і виберіть зображення (формати .png, .jpg, .jpeg, .webp) у вікні вибору файлу.
    - Якщо обрано “Video”. Натисніть на кнопку “VIDEO” і виберіть відео (формат .mp4) у вікні вибору файлу.

2. **Додавання аудіо**:
    - Натисніть на кнопку “AUDIO”.
    - Виберіть аудіофайл (будь-який формат) у вікні вибору файлу.
    - Відтворення аудіофайлу почнеться автоматично.

3. **Налаштування параметрів відображення**:

   Налаштуйте наступні параметри за допомогою повзунків:

   - Eye separation (Відстань між очима): Регулює відстань між лівим та правим око для стереоскопічного відображення.
   - Field of View (Зорове поле): Регулює ширину зорового поля.
   - Niar Clipping distance (Ближня площина відсікання): Регулює відстань від центру до площини обрізки.
   - Convergence distance (Відстань зближення): Регулює відстань від центру до точки зближення.
   - Позначте чекбокси, якщо потрібно включити камеру, Spaceball (пристрій управління) або увімкнути обертання шару та звуку за допомогою гіросокопа. Якщо пристрій не підтримує гіроскоп, то ця опція буде вимкнена та буде виведено відповідне повідомлення.

4. **Відображення сфери**:

   Налаштуйте наступні параметри за допомогою повзунків:

   - Sphere coordinates (Координати сфери): Регулюється X, Y та Z координати сфери.
   - Radius (Радіус): Регулює радіус кола обертання сфери.
   - Позначте чекбокс “Animation” (Анімація), якщо потрібно включити анімацію обертання сфери навколо геометричного центру.

5. **Налаштування звуку**:

   Налаштуйте наступні параметри звуку за допомогою повзунків:

   - Volume (Гучність): Регулює гучність звуку.
   - PlaybackRate (Темп відтворення): Регулює швидкість відтворення звуку.
   - Q value (bandwidth) (Полоса пропускання): Регулює ширину полоси пропускання звукового фільтра.
   - Frequency (Частота): Регулює частоту звукового фільтра.
   - Позначте чекбокси “Band-pass filter” (Смуговий фільтр) та “Reverb” (Реверберація), якщо потрібно включити ці ефекти.

Зразок вихідного коду

Код, що відповідає за зчитування даних з гіроскопа:

```
function readGyroscope() {
    var gyroPresent = navigator.userAgentData.mobile;

    if (gyroPresent) {
        timeStamp = Date.now();
        let sensor = new Gyroscope({
            frequency: 60
        });
        sensor.addEventListener('reading', e => {
            x = sensor.x
            y = sensor.y
            z = sensor.z
        });
        sensor.start();
    } else {
        $('#GyroscopeCheckbox').prop('disabled', true);
        alert('Gyroscope not supported');
    }

}
```

Частина коду, що відповідає за відрисовку сфери:

```
function draw() {
    if (audioPanner) {
        audioFilter.frequency.value = centerFrequencyInput;
        audioFilter.Q.value = Q_value;
        audio.volume = Volume;
        audio.playbackRate = PlaybackRate;

        if (GyroscopeRotate) {
            audioPanner.setPosition(
                (x * 1000).toFixed(2),
                (y * 1000).toFixed(2),
                (z * 1000).toFixed(2)
            );
        } else {
            audioPanner.setPosition(
                World_X * 1000,
                World_Y * 1000,
                World_Z * 1000
            );
        }
    }

    gl.clearColor(0, 0, 0, 1);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    if (camera_ind) {
        DrawWebCamVideo();
    }

    gl.clear(gl.DEPTH_BUFFER_BIT);
    stereoCamera.ApplyLeftFrustum();
    gl.colorMask(true, false, false, false);
    DrawSurface();
    DrawSphere();

    gl.clear(gl.DEPTH_BUFFER_BIT);
```

```
    stereoCamera.ApplyRightFrustum();
    gl.colorMask(false, true, true, false);
    DrawSurface();
    DrawSphere();
    gl.colorMask(true, true, true, true);
}

function DrawSphere() {
    let modelView = [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1];
    let translateToPointZero;
    if (GyroscopeRotate) {
        translateToPointZero = m4.translation(x.toFixed(2), y.toFixed(2), (z
- 20).toFixed(2));
    } else {
        translateToPointZero = m4.translation(World_X, World_Y, World_Z -
20);
    }

    gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false,
m4.multiply(stereoCamera.mModelViewMatrix,
m4.multiply(m4.multiply(stereoCamera.mProjectionMatrix,
translateToPointZero), modelView)));
    gl.bindTexture(gl.TEXTURE_2D, SphereTexture);
    gl.uniform1i(shProgram.iTexture, 0);

    Sphere.Draw();
}
```

Код, що відповідає за підключення та відтворення аудіо:

```
   audio = document.getElementById("audio");

        audio.addEventListener("pause", () => {
            audioContext.resume();
        });

        audio.addEventListener("play", () => {
            if (!audioContext) {
                audioContext = new(window.AudioContext ||
window.webkitAudioContext)();
                audioSource = audioContext.createMediaElementSource(audio);

                audioPanner = audioContext.createPanner();
                audioFilter = audioContext.createBiquadFilter();
                reverbNode = audioContext.createConvolver();

                audioPanner.panningModel = "HRTF";
                audioPanner.distanceModel = "linear";
                audioFilter.type = "bandpass";
                audioFilter.frequency.value = centerFrequencyInput;
                audioFilter.Q.value = Q_value;

                audio.volume = Volume;
                audio.playbackRate = PlaybackRate;

                const reverbUrl = "audio/reverb/BIG_HALL_E001_M2S.wav";
                const reverbRequest = new XMLHttpRequest();
                reverbRequest.open("GET", reverbUrl, true);
                reverbRequest.responseType = "arraybuffer";

                reverbRequest.onload = function () {
                    const audioData = reverbRequest.response;
                    audioContext.decodeAudioData(audioData, function
(buffer) {
                        reverbNode.buffer = buffer;
                        audioSource.connect(audioPanner);
                        audioPanner.connect(audioFilter);
```

```
                audioFilter.connect(reverbNode);
                reverbNode.connect(audioContext.destination);
            });
        };

        reverbRequest.send();

        audioContext.resume();
    }
});

const filter = document.getElementById("filter_check");

filter.addEventListener("change", function () {
    if (filter.checked) {
        audioPanner.disconnect();
        audioPanner.connect(audioFilter);
        audioFilter.connect(audioContext.destination);
    } else {
        audioPanner.disconnect();
        audioPanner.connect(audioContext.destination);
    }
});

const reverbCheckbox = document.getElementById('reverbCheckbox');

reverbCheckbox.addEventListener('change', () => {
    if (reverbCheckbox.checked) {
        audioFilter.disconnect();
        audioFilter.connect(reverbNode);
        reverbNode.connect(audioContext.destination);
    } else {
        reverbNode.disconnect();
        audioFilter.disconnect();
        audioFilter.connect(audioContext.destination);
    }
});

audio.play();
```