Міністерство освіти й науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Кафедра автоматизації проектування енергетичних процесів і систем

Звіт
з циклу лабораторних робіт з дисципліни
«Методи синтезу віртуальної реальності»

Графічно-розрахункова робота
Варіант-23

Виконав:
студент 5-го курсу
групи ТР-22мп НН ІАТЕ
Савонік Ю.І.
Перевірив:
Демчишин А.А.

Київ-2023

Опис завдання:

- Повторно використати код із практичного завдання №2;
- реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні за допомогою матеріального інтерфейсу (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;
- додати звуковий фільтр високих частот (необхідно використовувати інтерфейс BiquadFilterNode). Додати елемент прапорця, який вмикає або вимикає фільтр.

Підготувати цифровий звіт, який містить:

- титульну сторінка;
- розділ з описом завдання (1 сторінка);
- розділ з описом теорії (2 сторінки);
- розділ з описом деталей впровадження (2 сторінки);
- розділ інструкції користувача зі скріншотами (2 сторінки);
- зразок вихідного коду (2 сторінки).

Теоретична частина:

Звук є механічним хвильовим процесом, який поширюється через середовище у вигляді змін в атмосферному тиску. Основні елементи звуку включають частоту, амплітуду, фазу та тембр.

- Частота: Частота звуку визначається кількістю коливань за одиницю часу і вимірюється в герцах (Гц). Частота визначає висоту звуку і сприймається людиною як низькочастотний звук (наприклад, глибокий бас) або високочастотний звук (наприклад, дзвінкий сопілка).

- Амплітуда: Амплітуда звуку визначається амплітудою коливань і вимірюється в децибелах (дБ). Амплітуда впливає на гучність звуку, де велика амплітуда відповідає гучному звуку, а мала амплітуда - тихому звуку.

- Фаза: Фаза визначає положення коливань звуку в часі. Вона вимірюється в радіанах або градусах. Фаза впливає на спосіб, яким звук поєднується з іншими звуками або які звуки можуть нейтралізувати один одного.

- Тембр: Тембр визначає характер звуку і розрізняє його від інших звуків тієї ж самої частоти та амплітуди. Тембр залежить від співвідношення та інтенсивності гармонік (складових частот) у звуці і дає можливість розрізнити різні інструменти чи голоси.

Сприйняття звуку людиною є складним процесом, який включає фізіологічні та психологічні аспекти. Звукові хвилі потрапляють у вухо, де вони сприймаються вушною раковиною і просуваються по зовнішньому слуховому проходу до барабанної перетинки. Далі звукові коливання передаються через внутрішнє вухо, де вони стимулюють відповідні рецептори - волоскові клітини, які перетворюють звукові сигнали на електричні імпульси. Ці імпульси передаються до мозку через слуховий нерв, де вони обробляються та інтерпретуються як звук.

Описані принципи сприйняття звуку та звуку самого по собі використовуються при розробці застосунків, що пов'язані зі звуком, для забезпечення бажаної якості, гучності, тимбру та інших звукових аспектів, які задовольняють потреби користувачів.

Звукова ж теорія та його реалізація в застосунку з використанням WebGL та JavaScript базуються на Web Audio API, який надає потужні засоби для синтезу, обробки та відтворення звуку у веб-додатках.

Розберемо основні аспекти Web Audio API.

Web Audio API є спеціальною частиною HTML5 та JavaScript, яка дозволяє створювати та маніпулювати звуковими даними в реальному часі. Це дає можливість застосовувати різні техніки синтезу, обробки та аналізу звуку.

Звук в цифровій формі представляється як послідовність амплітудних значень на часовій шкалі.Тут звуковий сигнал зазвичай має дві характеристики: частоту та амплітуду. Частота визначає висоту звуку і вимірюється в герцах (Гц), а амплітуда визначає гучність і вимірюється в децибелах (дБ).

Для створення звуків (синтезу) у застосунку можна використовувати різні методи синтезу, такі як аддитивний синтез, субтрактивний синтез, фазовий синтез тощо. За допомогою Web Audio API можна створити звукові об'єкти, такі як OscillatorNode, які генерують коливання з заданою частотою та формою хвилі, і з'єднати їх з вихідними вузлами для відтворення звуку.

Web Audio API надає широкий набір звукових ефектів та обробки звуку. Наприклад, можна застосовувати фільтри, реверберацію, еквалайзери, затримку тощо. Звукові ефекти можна додавати до аудіо-графу, створеного звуковими об'єктами, та налаштовувати параметри для досягнення бажаного звучання.

Web Audio API дозволяє відтворювати звукові дані за допомогою AudioBufferSourceNode. Цей об'єкт може бути заповнений звуковими даними з аудіофайлу або створений динамічно з використанням синтезу. Звук можна відтворювати у реальному часі або записувати у файл.

За допомогою WebGL, яке є 3D-графічною технологією для вебу, можна візуалізувати звукові дані. Наприклад, можна створити спектрограму звуку, візуалізацію амплітуди або інші графічні ефекти, що відображають характеристики звуку в часовій та частотній областях.

Деталі впровадження:

JavaScript:

В програмі JavaScript, спочатку було додано змінні, які необхідні для:

1. Фізичної зміни джерела звуку (змінні rotation та position).
2. Підключення Web Audio API (змінна context).
3. Джерела звуку (змінні soundFileName та source).
4. Представлення джерела звуку у просторі (змінна panner).
5. Додавання фільтру (змінна highpassFilter).
6. Активації початку відтворювання звуку (змінна playButton).

Далі було створено функцію, що повертає масив з точок, які відображають джерело звуку у просторі у вигляді сфери. Для створення такого масиву використовується наступна формула:

$$x = rotation_x + (radius * \cos(alpha) * \sin(beta))$$
$$y = rotation_y + (radius * sin(alpha) * \sin(beta))$$
$$z = rotation_z + (radius * \cos(alpha))$$

Формула 1. Формула сфери у точці rotation

Так як CORS-політика забороняє вмикання аудіо- чи відеоконтенту одразу при переході за посиланням, то була додана кнопка. У функції init на її натиск додана підписка, що просто викликає функцію CreateSound. У цій функції відбувається відразу декілька дій:

1. Створюється аудіоконтекст (context) та новий запит (за допомогою XMLHttpRequest), який надсилає запит на сервер. Як тільки він завантажється, то відбувається декодування звуку.
2. Створення (за допомогою вбудованої функції createPanner), налаштування(за допомогою вбудованих властивостей) та під'єднання представлення джерела звуку у просторі (panner) до аудоконтексту.
3. Створення(за допомогою вбудованої функції createBiquadFilter), налаштування фільтру високих частот (за допомогою вбудованих властивостей) та під'єднання фільтру до джерела звуку у просторі.
4. Надсилання запиту.
5. Початок відіграшу звукового файлу.
6. Деактивація кнопки.

У функції init також відбувається підписка на рух гіроскопа (за допомогою події devicemotion.rotationRate). У разі, якщо значення panner не пусте, то відбуваються наступні дії:

1. Отримуються дані з гіроскопа і конвертуються у звичайні швидкості.
2. Для обертання навколо фігури застосовуються наступні формули:

$$rotation_x = R * \cos(position_y) * \cos(position_x)$$

$$rotation_y = R * \cos(position_y) * \sin(position_z)$$
$$rotation_z = height * \sin(position_y)$$

Формула 2. Формула еліпсоїда з a = b = R, c = height.

3. Встановлення позиції джерела звуку у точці rotation.
4. Виклик функції Redraw, що оновлює дані для відображення на сцені.

У функції Redraw відбувається також перевірка наявності активного прапорця filter. Якщо він активний, то встановлюються користувацькі налаштування фільтру, інакше -за замовчуванням (фільтр вважається неактивним).

HTML:

Для початку зчитування руху з гіроскопа та початку відтворення пісні була додана кнопка “Play”. Якщо користувач не натискає на дану кнопку, то всі зчитування ігноруються.

Також був доданий чекбокс для зчитування його активності у файлі main.js. Якщо він активний, то встановлюються користувацькі налаштування фільтру, інакше -за замовчуванням (фільтр вважається неактивним).

Вершинний шейдер:

Додаткового коду немає.

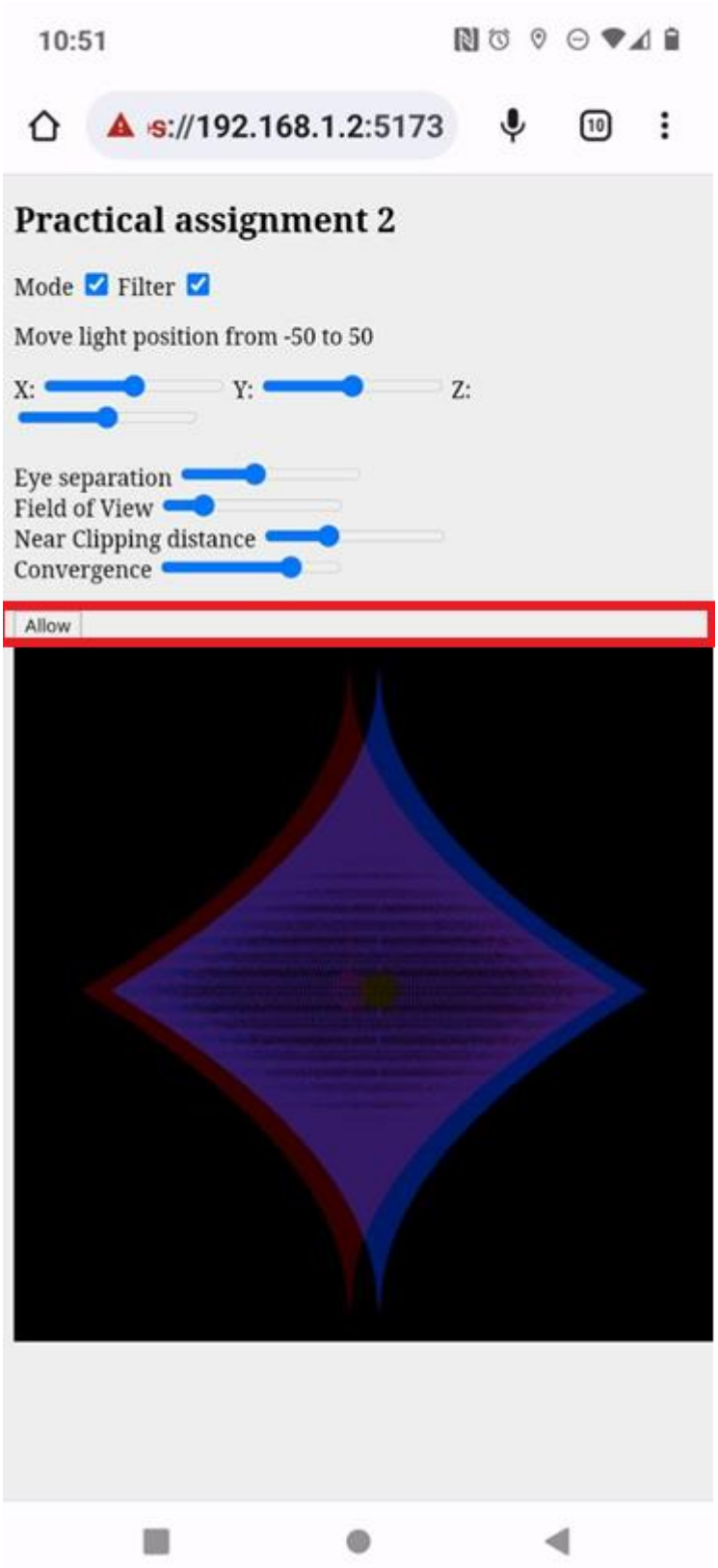
Фрагментний шейдер:

Додаткового коду немає.

Інструкція користувача:

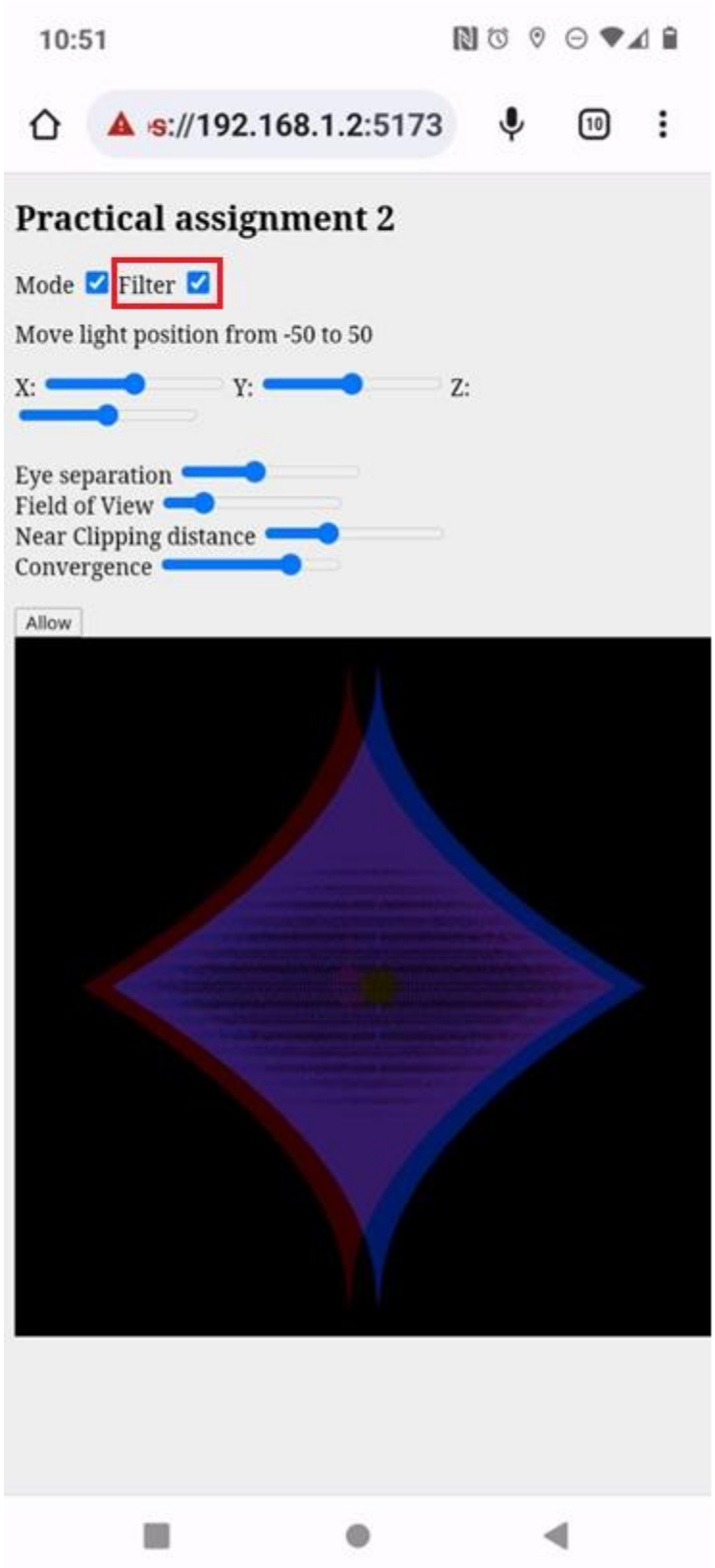
Користувач має можливість:

1) Активувати початок відтворювання пісні.



Для того, щоб активувати початок відтворювання пісні та почати зчитувати дані з гіроскопу, необхідно натиснути на кнопку, що виділена на скріншоті. При натиску на кнопку, вона деактивується.

2) Накладати на звук фільтр високих частот:



Для того, щоб накласти на звук фільтр високих частот, необхідно використовувати прапорець, що виділений на скріншоті. При його активності, фільтр накладається на звук. У разі його неактивного стану, налаштування фільтру повертаються до стану, при якому фільтр не впливаає на звук.

Вихідний код:

main.js:

```
'use strict';

let gl;                         // The webgl context.
let surface;                    // A surface model
let sound;
let shProgram;                  // A shader program
let spaceball;                  // A SimpleRotator object that lets the user rotate the
view by mouse.
let height = 5;
let p = 3;
let R = (height * height) / (2*p);
let radius = 0.3;
const PI = Math.PI;
let length;
let inputData;
var playButton;
let rotation = {x:0, y: 0, z:0};
let position = {x:1, y: 0, z:0};
var context;
let soundFileName = "perfectSound.mp3";
let source;
let panner;
let highpassFilter;

function GetRadiansFromDegree(angle) { …
}
// Constructor
function Model(name) { …
}

// Constructor
function ShaderProgram(name, program) { …
}
function UpdateInputData()
{
 …
}

function draw() { …
}

function GetCurrentZPosition(h){ …
}
// surface - parabolic humming-top
// x = ((|z| - h)^2 / 2*p)) * cosB
// y = ((|z| - h)^2 / 2*p)) * sinB
// z = z
function CreateSurfaceData()
{ …
}
```

```
function CreateSoundData()
{
    let vertexList = [];

    const stepU = inputData.surfaceType.checked ? 5 : 9;

    for (let u = 0; u <= 360; u += stepU)
    {
        for(let v = 0; v <= 360; v += stepU)
        {
            let alpha = GetRadiansFromDegree(u);
            let beta = GetRadiansFromDegree(v);

            vertexList.push(rotation.x +  (radius *  Math.cos(alpha) *
Math.sin(beta)),rotation.y + (radius *  Math.sin(alpha) * Math.sin(beta)),rotation.z +
(radius *  Math.cos(beta)));
        }
    }

    length = 360;
    return vertexList;
}

/* Initialize the WebGL context. Called from init() */
function initGL() {  …
}

function CreateSound()
{
    context = new window.AudioContext();
    CreateHighpassFilter();
    CreatePanner();
    const request = new XMLHttpRequest();
    source = context.createBufferSource();
    request.open("GET", soundFileName, true);
    request.responseType = "arraybuffer";

    request.onload = () => {
        const audioData = request.response;

        context.decodeAudioData(audioData, (buffer) => {

            source.buffer = buffer;
            source.connect(highpassFilter);
            highpassFilter.connect(panner);
            panner.connect(context.destination);
            source.loop = true;
        }, (err) => {alert(err)}
        );
    };

    request.send();
```

```
    source.start(0);
    playButton.disabled = true;
    playButton.style.display = 'none';
}

function CreateHighpassFilter()
{
    highpassFilter = context.createBiquadFilter();
    highpassFilter.type = "highpass";
    highpassFilter.frequency.value = 1500;
}

function CreatePanner()
{
    panner = context.createPanner();
    panner.panningModel = "HRTF";
    panner.distanceModel = "inverse";
    panner.refDistance = 1;
    panner.maxDistance = 1000;
    panner.rolloffFactor = 1;
    panner.coneInnerAngle = 360;
    panner.coneOuterAngle = 0;
    panner.coneOuterGain = 0;
}

function createProgram(gl, vShader, fShader) { …
}
/**
 * initialization function that will be called when the page has loaded
 */
function init() {
    let canvas;
    try {
        canvas = document.getElementById("webglcanvas");
        gl = canvas.getContext("webgl");
        if ( ! gl ) {
            throw "Browser does not support WebGL";
        }
    }
    catch (e) {
        document.getElementById("canvas-holder").innerHTML =
            "<p>Sorry, could not get a WebGL graphics context.</p>";
        return;
    }
    try {
        initGL();  // initialize the WebGL graphics context
    }
    catch (e) {
        document.getElementById("canvas-holder").innerHTML =
            "<p>Sorry, could not initialize the WebGL graphics context: " + e + "</p>";
        return;
    }
```

```
    window.addEventListener('devicemotion', (event) => {
        if(panner)
        {
            let dt = 0.05;
            position.x += GetRadiansFromDegree(event.rotationRate.alpha) * dt;
            position.y += GetRadiansFromDegree(event.rotationRate.beta) * dt;
            position.z += GetRadiansFromDegree(event.rotationRate.gamma) * dt;

            rotation.x = R * Math.cos(position.y)*Math.cos(position.x);
            rotation.y = height * Math.sin(position.y);
            rotation.z = R * Math.cos(position.y)*Math.sin(position.z);

            panner.setPosition(rotation.x, rotation.y, rotation.z);
            panner.setOrientation(0,0,0);
            sound.BufferData(CreateSoundData());
        }
    })

    playButton = document.getElementById("play-button");
    playButton.addEventListener('click', function(){

                CreateSound();
    });
    spaceball = new TrackballRotator(canvas, draw, 0);
    Update();
}

function Update()
{
    draw()
    window.requestAnimationFrame(Update);
}

function Redraw() {
    highpassFilter.frequency.value = inputData.filter.checked ? 1500 : 0;
    surface.BufferData(CreateSurfaceData());
    sound.BufferData(CreateSoundData());
    draw();
}
```