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-2

Виконав:
студент 6-го курсу
групи ТР-21мн
Тітов В.М.
Перевірив:
Демчишин А.

Київ-2023

Опис завдання:

Реалізувати лабораторну роботу:

- 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 низьких частот (необхідно використовувати інтерфейс BiquadFilterNode). Додати елемент прапорця, який вмикає або вимикає фільтр.

Підготувати цифровий звіт, який містить:

- Титульну сторінка;
- Розділ з описом завдання;
- Розділ з описом теорії;
- Розділ з описом деталей впровадження;
- Розділ інструкції користувача зі скріншотами;
- Зразок вихідного коду.

Теоретична частина

У середовищі Unity використання аудіофункціоналу базується на використанні Unity Audio API. Це API дозволяє розробникам працювати з аудіоданими в реальному часі та створювати інтерактивні аудіо-сценарії в ігровому середовищі. Основні компоненти, які використовуються, включають AudioSources, AudioMixer, та AudioFilters.

AudioSource та AudioClip: Компоненти для відтворення звуку в грі. AudioClip містить аудіодані, а AudioSource відповідає за відтворення цих даних. За допомогою них можна керувати такими параметрами як гучність, панорамування та висота звуку.

AudioMixer та AudioMixerGroup: Використовуються для застосування ефектів обробки звуку в реальному часі. AudioMixerGroup дозволяє групувати та керувати різними аудіо-джерелами.

AudioFilter: Цей компонент дозволяє створювати власні фільтри та ефекти обробки звуку для кожного Audio Source.

Опис деталей реалізації

У даній роботі був написаний скрипт ShelfLowPassFilterController, який використовує компоненти Unity для створення звукового ефекту з низькочастотним фільтром. Основні етапи реалізації включають:

Аудіофункціонал: Запуск аудіо-джерела, налаштування параметрів фільтрації та відтворення звуку.

Сполучення із сенсорами: Цей скрипт використовує дані сенсора магнітометра для динамічної зміни параметрів фільтрації в залежності від магнітного заголовку.

Обробка даних магнітного заголовку: Визначення частоти фільтрації на основі значень магнітометру, що дозволяє створювати аудіо-ефекти, пов'язані з орієнтацією пристрою.

Візуалізація звукового джерела: Зміщення та обертання графічного представлення звукового джерела в ігровому просторі.

Управління фільтрацією: Додано можливість увімкнення/вимкнення низькочастотного фільтра за допомогою чекбокса.

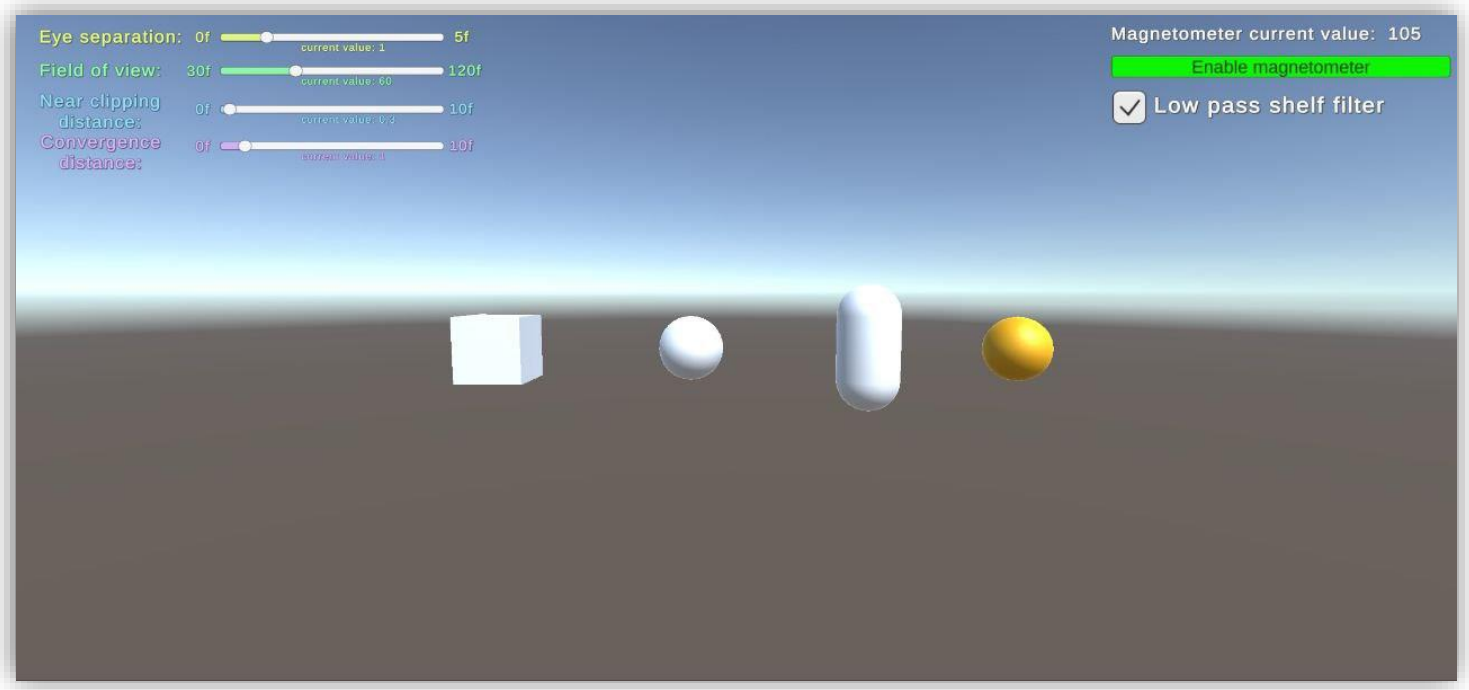
Інтеграція з аудіо-графікою: Даний скрипт взаємодіє з аудіо-графікою, застосовуючи фільтрацію до звукового потоку.

Це дозволяє створювати інтерактивні аудіо-сценарії, де звук реагує на зміни в оточуючому середовищі та орієнтації пристрою.

Інструкція користувача:

Даний продукт надає змогу користувачам виконувати наступні дії:

1. Користувач має змогу включати та виключати магнітометр, оранжева сфера(це аудіо сфера) яка обертається навколо об’єктів завдяки даним з магнітометра



2. Також у даному продукту користувач має змогу накладати та вимикати фільтр низьких частот на музику що грає (за замовчування фільтр вимкнений)

Вихідний код

```
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

[RequireComponent(typeof(AudioSource))]
public class ShelfLowPassFilterController : MonoBehaviour
{
    public TextMeshProUGUI log;
    public TextMeshProUGUI magnetometerValue;
    public Toggle filterToggle;
    public GameObject soundSourceSphere;
    public GameObject ourObjects;
    private AudioSource audioSource;

    private float lastMagneticHeading = 0f;
    private float rotationSpeed = 30f;

    // Параметри низькочастотного фільтра
    private float lowPassCutoffFrequency = 5000f; // Відрегульвуємо за необхідності
    private AudioLowPassFilter lowPassFilter;

    public float compassSmooth = 3f; // Оголошуючи публічну змінну з назвою compassSmooth для управління згладжуванням обертання
    private float m_lastMagneticHeading = 0f; // Оголошуючи приватну змінну з назвою m_lastMagneticHeading для збереження попереднього значення магнітометру

    private void Start()
    {
        audioSource = GetComponent<AudioSource>(); // отримуємо компонент Аудіо з об’кту

        audioSource.Play(); // вмикаємо наш трек(музику)

        Input.location.Start();
        Input.compass.enabled = true;

        // Додаємо або отримуємо компонент AudioLowPassFilter
        lowPassFilter = audioSource.GetComponent<AudioLowPassFilter>();
        if (lowPassFilter == null)
        {
            lowPassFilter = audioSource.gameObject.AddComponent<AudioLowPassFilter>();
        }

        // Встановлюємо початкові параметри для низькочастотного фільтра
        lowPassFilter.cutoffFrequency = lowPassCutoffFrequency;
        lowPassFilter.enabled = false; // Заблоковано за замовчуванням

        // Кешуємо поточне значення компаса
        m_lastMagneticHeading = Input.compass.magneticHeading; // Зберігаємо початкове значення магнітного заголовку
    }

    void Update()
    {
        if (Input.compass.enabled)
        {
            try
```

```
        {
            UpdateSoundSourcePosition();

            // Оновлюємо параметри фільтра на основі магнітного заголовку
            lowPassCutoffFrequency = 5000f + (Input.compass.magneticHeading / 180f) * 2000f;
            lowPassFilter.cutoffFrequency = lowPassCutoffFrequency;

            // Застосовуємо фільтр, якщо він увімкнений
            if (filterToggle.isOn)
            {
                EnableLowPassFilter();
            }
            else
            {
                DisableLowPassFilter();
            }
        }
        catch (Exception e)
        {
            log.text = e.Message;
        }
    }
}

void UpdateSoundSourcePosition()
{
    magnetometerValue.text = Input.compass.magneticHeading.ToString("0.");
    float rotationAmount = Input.compass.magneticHeading - lastMagneticHeading;
    Vector3 rotationAxis = Vector3.up;
    Vector3 rotationCenter = ourObjects.transform.position;

    // Обертаємо звуковий джерело навколо центру обертання
    soundSourceSphere.transform.RotateAround(rotationCenter, rotationAxis, rotationAmount * rotationSpeed
* Time.deltaTime);

    lastMagneticHeading = Input.compass.magneticHeading;
}

void EnableLowPassFilter()
{
    lowPassFilter.enabled = true;
}

void DisableLowPassFilter()
{
    lowPassFilter.enabled = false;
}
}
```