МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Практичні роботи

Minor *«Розробник ігрових додатків»*

дисципліна *«Програмування віртуальної реальності»*

(назва дисципліни)

Виконав: студент 4 курсу групи  *545б*

напряму підготовки (спеціальності):

*123 Комп’ютерна інженерія*

(шифр і назва напряму підготовки / спеціальності)

*Лашин Д.О.*

(прізвище й ініціали студента)

Прийняв: *доц. каф 603, к.т.н, Лучшев П.О.*

(посада, науковий ступінь, прізвище й ініціали)

Національна шкала:

Кількість балів:

Оцінка ECTS:

Зміст

[1. Формування простору віртуальної реальності 3](#_Toc217145613)

[1.1. Моделювання та генерація ландшафту 3](#_Toc217145614)

[1.2. Моделювання поверхні води 7](#_Toc217145615)

[1.3. Моделювання атмосфери, туману 9](#_Toc217145616)

[2. Створення та моделювання об’єктів віртуальної реальності 11](#_Toc217145617)

[2.1. Об'єкти віртуальної реальності на основі фракталів 11](#_Toc217145618)

[2.2. Функціональне моделювання комплексних об'єктів (Signed Distance Function) 14](#_Toc217145619)

[Загальний перелік посилань 18](#_Toc217145620)

[Додаток А. Лістинг програми до практичної роботи № 1.1 19](#_Toc217145621)

[Лістинг програми до практичної роботи № 1.2 24](#_Toc217145622)

[Лістинг програми до практичної роботи № 1.3 26](#_Toc217145623)

[Додаток Б. Лістинг програми до практичної роботи № 2.1 29](#_Toc217145624)

[Лістинг програми до практичної роботи № 2.2 36](#_Toc217145625)

# 1. Формування простору віртуальної реальності

## 1.1. Моделювання та генерація ландшафту

### Завдання

### Необхідно обрати один із методів створення віртуального ландшафту та реалізувати програмний проєкт для побудови й відображення тривимірної поверхні. Згенерований ландшафт повинен відображатися у двох режимах: у вигляді каркасної сітки та з кольоровим градієнтом, що залежить від висоти поверхні.

### Теоретичні відомості

Для створення віртуального ландшафту було використано алгоритм **Perlin Noise (шум Перліна)**, реалізований у середовищі **Unity**. Даний алгоритм дозволяє отримати плавні та природні переходи висот за рахунок використання псевдовипадкових значень, які змінюються без різких стрибків.

Простір розбивається на сітку, у вершинах якої зберігаються градієнти. Для кожної точки поверхні обчислюється значення шуму з урахуванням відстані до сусідніх вершин та виконується згладжування результатів. Це дозволяє отримати реалістичний рельєф місцевості.

### Результати виконання практичної роботи

У ході виконання роботи було реалізовано генерацію ландшафту на основі алгоритму Perlin Noise. Поверхня коректно відображається у вигляді полігональної сітки, а також у режимі каркасної моделі. Додатково реалізовано градієнтне забарвлення поверхні залежно від її висоти.

Налаштування параметрів генерації ландшафту, режим відображення каркасу та варіанти кольорових схем представлені на рисунках 1.1–1.4.

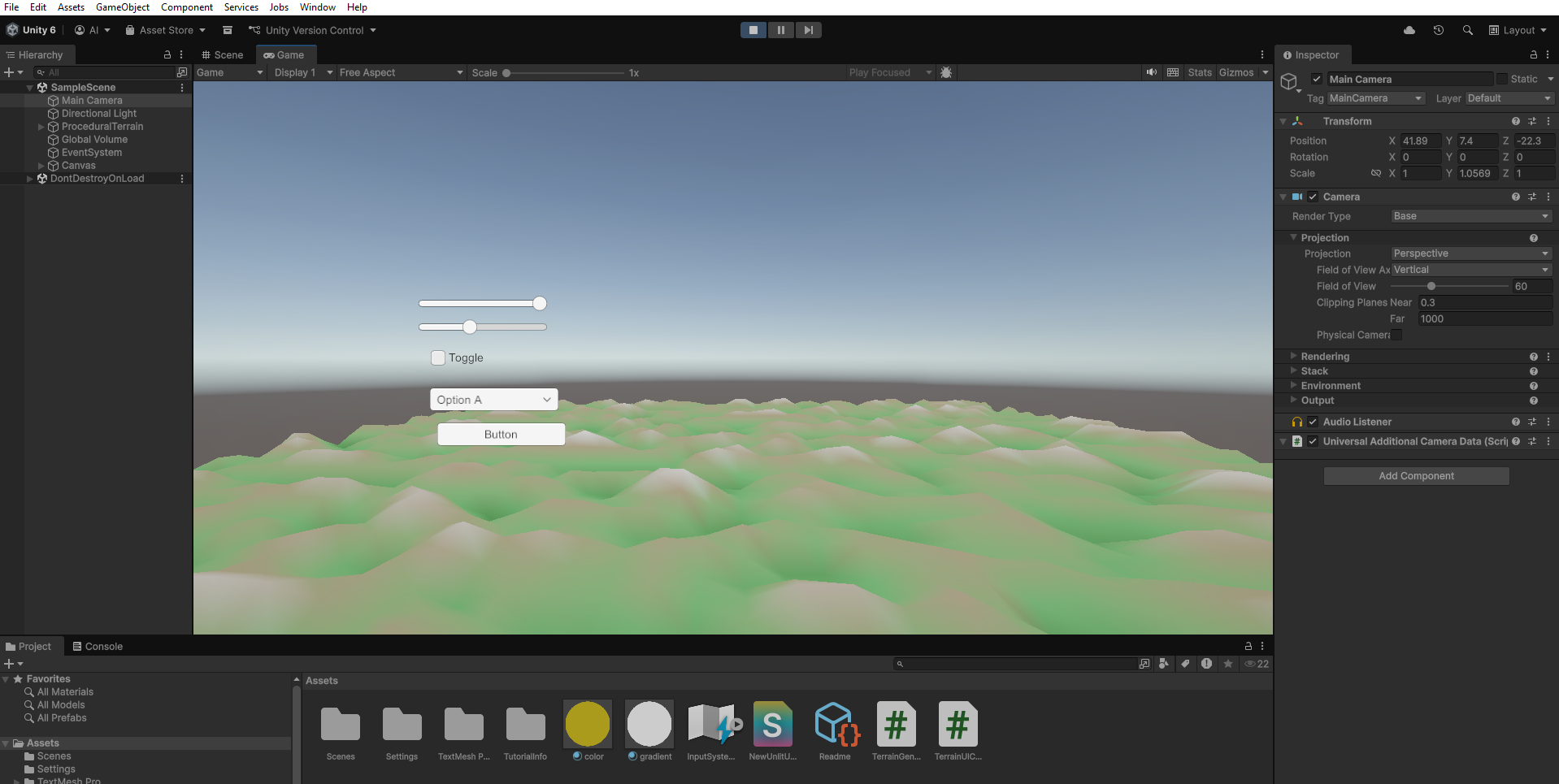


Рисунок 1.1 – Налаштування параметрів генерації ландшафту

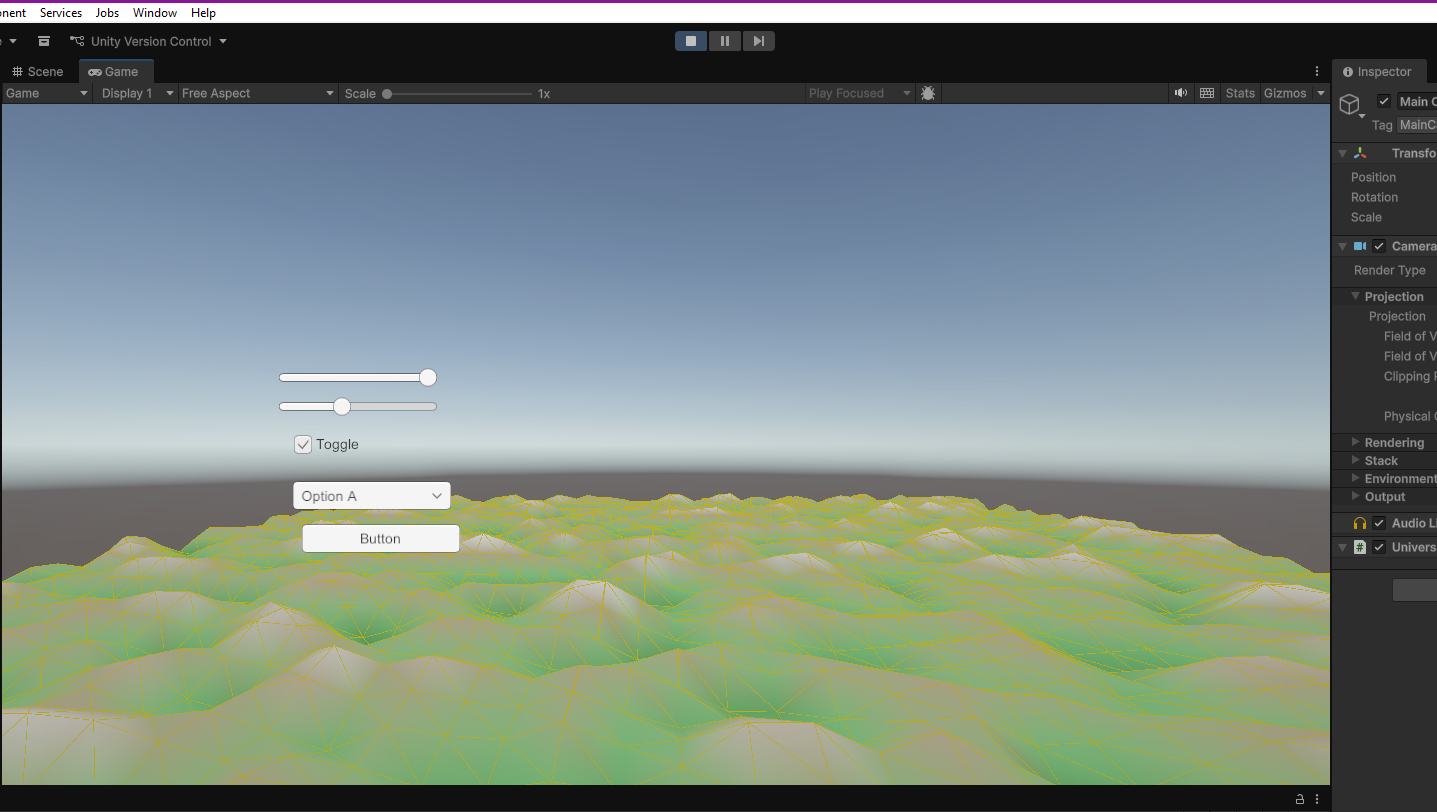


Рисунок 1.2 – Відображення каркасної сітки

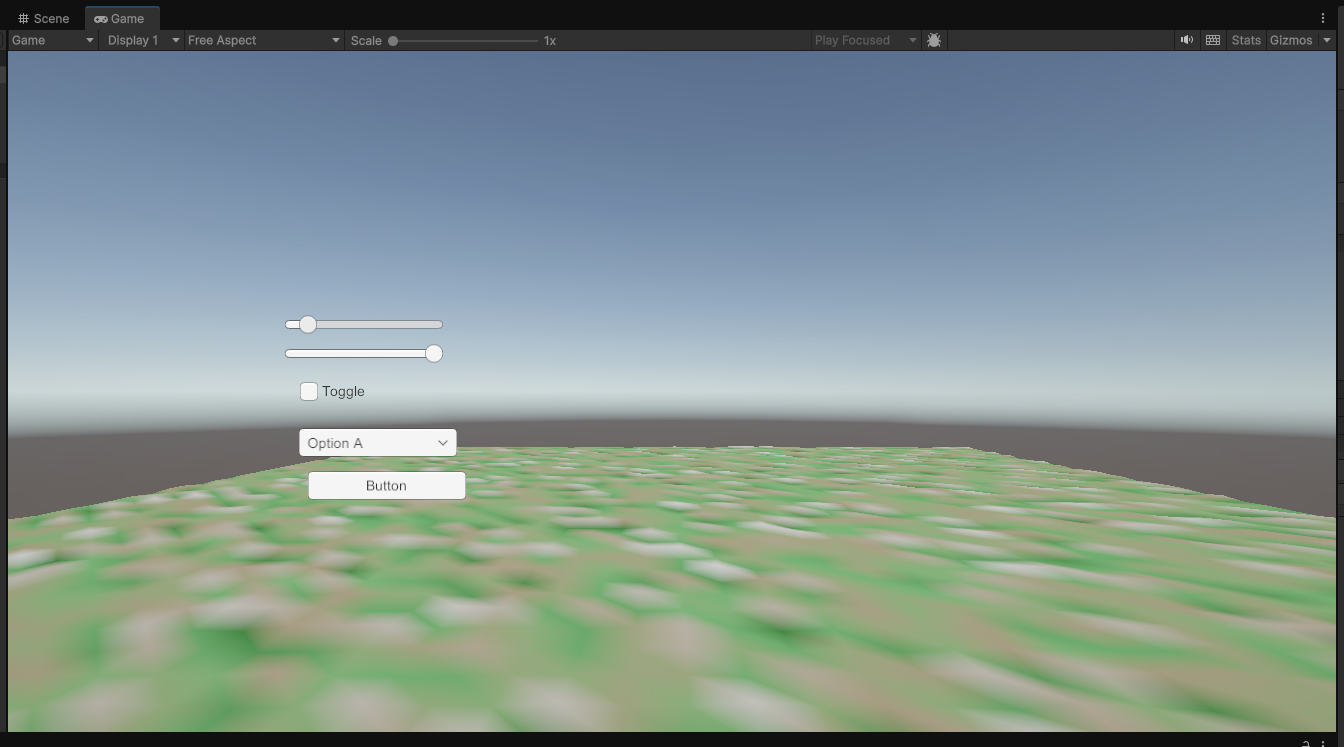


Рисунок 1.3 – Налаштування висот ландшафту

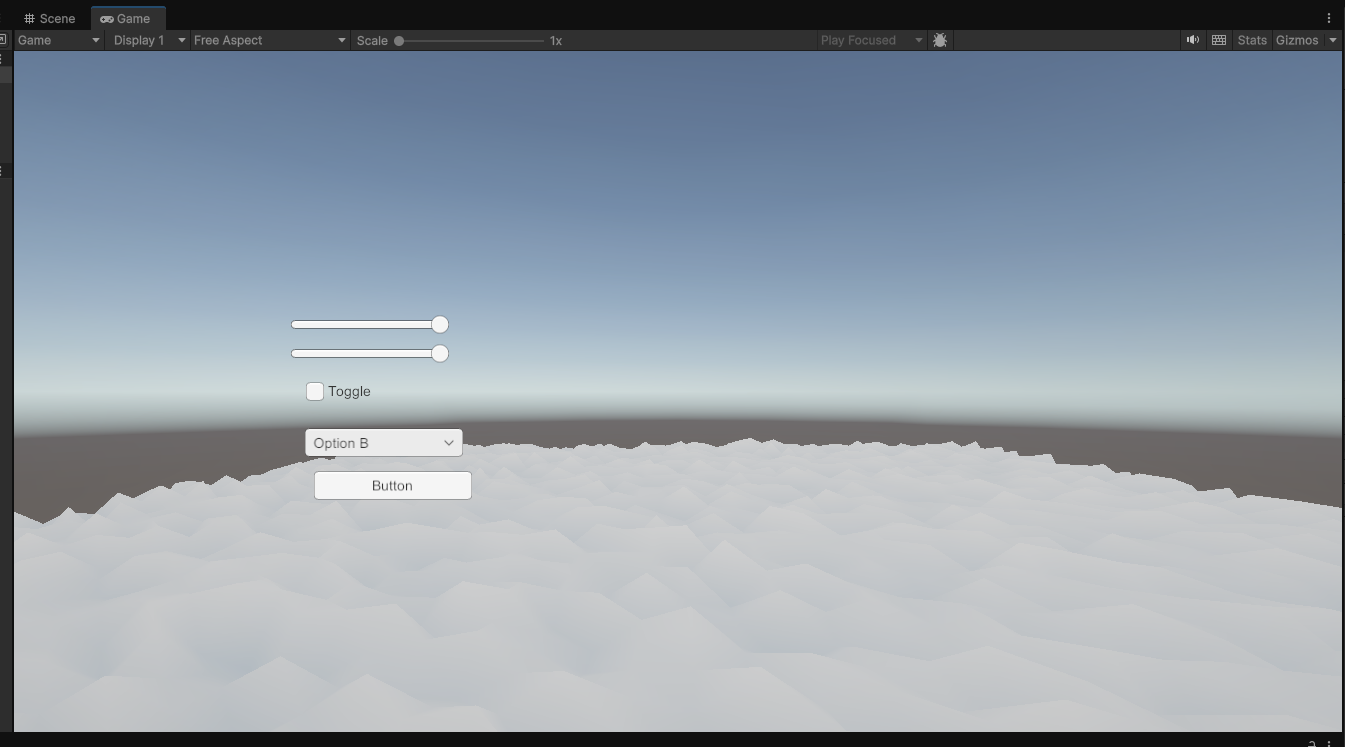


Рисунок 1.4 – Налаштування кольору ландшафту (сніжний)

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та частково підвищеного рівня складності, що відображено в таблиці 1.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 1.1 | | | | |
| № | Складність | Вимоги до роботи | Бали | Зроблено |
|  | Базовий рівень | Формування ландшафту у вигляді каркасної моделі | 2 | *2* |
|  | Градієнтне забарвлення поверхні залежно від висоти ландшафту | 2 | *2* |
|  | ООП. Реалізація об'єкта, що моделюється, у вигляді одного або декількох класів власної розробки | 1 | *1* |
|  | Підвищений рівень | Інтерактивне керування параметрами модельованого ландшафту (порожнина, рівень деталізації, розмальовка тощо) | 3 | *3* |
|  | Своєчасність виконання | Дострокове виконання (у строки попередньої роботи та раніше) – 2 бали, у зазначені терміни – 1 бал, після зазначених термінів – 0 балів | 2 | *0* |

## 1.2. Моделювання поверхні води

### Завдання

Метою даної роботи є створення спрощеної, але наочної моделі водної поверхні для використання у віртуальній сцені. Необхідно реалізувати статичну або динамічну поверхню води з використанням ефекту прозорості та можливістю зміни її параметрів.

### Теоретичні відомості

Для візуалізації води у тривимірних сценах зазвичай використовують полігональні поверхні з накладеними матеріалами та шейдерами. Реалістичність води досягається за рахунок прозорості, відблисків та невеликих коливань поверхні, які імітують хвилі.

***Результати виконання практичної роботи***

У даній роботі водна поверхня реалізована у вигляді плоскої тривимірної моделі, на яку застосовано спеціальний матеріал з шейдером. Шейдер дозволяє регулювати рівень прозорості води, а також створює ефект хвиль за допомогою зміни координат вершин.

Реалізовано можливість інтерактивного керування параметрами води, такими як глибина та прозорість, що дозволяє адаптувати вигляд сцени під різні умови.

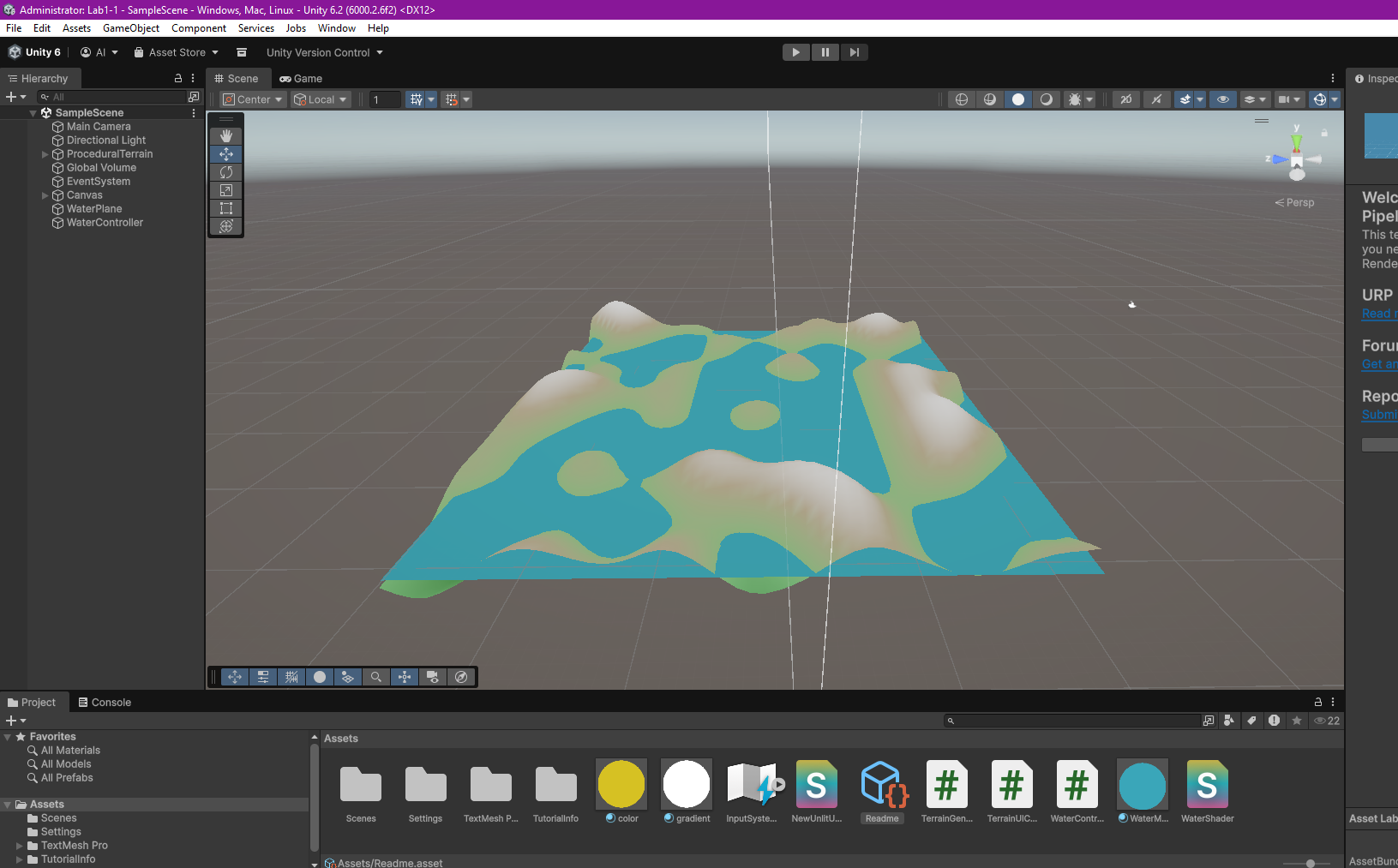


Рис. 1.5. – Статична модель водної поверхні

Результати виконання роботи наведені в таблиці 1.2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 1.2 | | | | |
| № | Складність | Вимоги до роботи | Бали | Оцінка |
| 1 | Базовий рівень | Статична модель поверхні | 1 | *1* |
| 2 | Використання ефекту прозорості | 1 | *1* |
| 3 | Динамічна модель поверхні | 2 | *2* |
| 4 | ООП. Реалізація моделі водної поверхні у вигляді одного або кількох класів власної розробки | 1 | *1* |
| 5 | Підвищений рівень | Інтерактивне керування параметрами  (рівень глибини, прозорість тощо)  водної поверхні, яку моделювали | 3 | *3* |
| 6 | Терміни | Дострокове виконання (у строки попередньої роботи та раніше) – 2 бали, у зазначені терміни – 1 бал, після зазначених термінів – 0 балів | 2 | *0* |

## 1.3. Моделювання атмосфери, туману

### Завдання

Завданням роботи є доповнення віртуального ландшафту атмосферними ефектами, такими як небо, туман та хмари. Дані елементи дозволяють створити відчуття глибини сцени та покращити візуальне сприйняття віртуального середовища.

### Теоретичні відомості

Атмосферні ефекти у тривимірній графіці реалізуються за допомогою налаштувань освітлення, градієнтів кольору неба та ефектів туману. Туман дозволяє приховати межі сцени та створити ілюзію великої відстані. Хмари можуть моделюватися як напівпрозорі об’єкти з анімацією текстур.

### Результати виконання практичної роботи

У результаті виконання роботи було реалізовано систему зміни часу доби, яка впливає на положення джерела світла та колір неба. Додано ефект туману з можливістю регулювання його щільності. Хмари реалізовані у вигляді окремих об’єктів з анімацією руху, що створює динамічну атмосферу сцени.

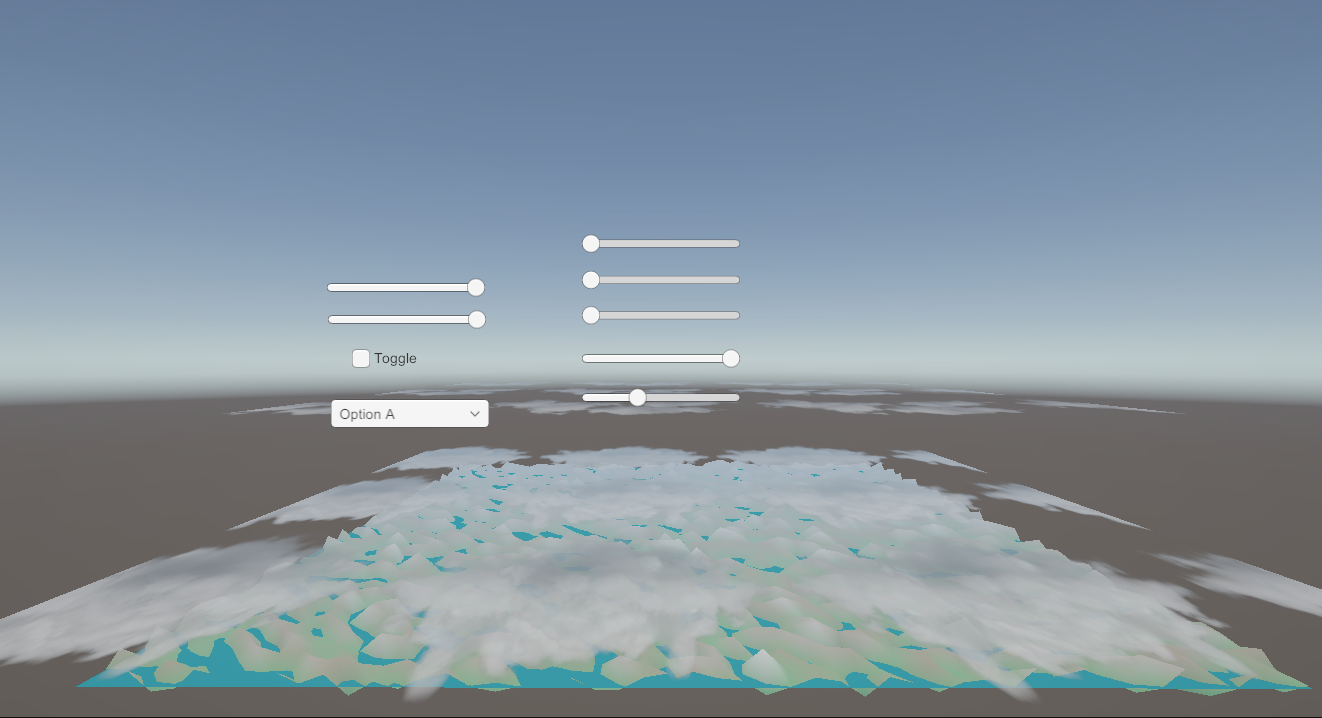


Рис 1.3. – Ландшафт з хмарами та ефектом туману

Результати виконання роботи наведені в таблиці 1.3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 1.3 | | | | |
| № | Складність | Вимоги до роботи | Бали | Оцінка |
| 1 | Базовий рівень | Небо. Реалізація у вигляді класу власної  розробки | 1 | *1* |
| 2 | Туман/димка – розмиття деталізації на  лінії горизонту. | 1 | *1* |
| 3 | Хмари. Реалізація у вигляді одного або  кількох класів (для різних типів хмар)  власної розробки | 3 | *3* |
| 4 | ООП. Реалізація роботи у вигляді одного  або кількох класів власної розробки | 1 | *1* |
| 5 | Підвищений рівень | Динаміка (рух хмар) та загальна параметризація атмосфери в залежності від часу доби (день, ніч, світанок, захід сонця) | 3 | *3* |
| 6 | Терміни | своєчасність виконання, бали:  достроково - 2, у строк - 1, з порушенням терміну виконання - 0 | 2 | *0* |

# 2. Створення та моделювання об’єктів віртуальної реальності

## 2.1. Об'єкти віртуальної реальності на основі фракталів

### Завдання

Метою роботи є створення процедурної моделі природного об’єкта з використанням **L-систем**. Необхідно реалізувати алгоритм генерації фрактальної структури та відобразити її у тривимірному просторі.

### Теоретичні відомості

L-системи використовуються для опису процесів росту рослин та інших природних форм. Вони базуються на ітеративному переписуванні рядків за заданими правилами. Для візуалізації отриманого результату застосовується метод «черепашкової графіки», який дозволяє інтерпретувати символи як команди руху та повороту у просторі.

### Результати виконання практичної роботи

У ході виконання роботи було реалізовано генерацію 3D-дерев на основі L-систем. Зі збільшенням кількості ітерацій структура дерева ускладнюється. Для підвищення реалістичності додано випадковість у довжину гілок та кути повороту, а також реалізовано генерацію листя.

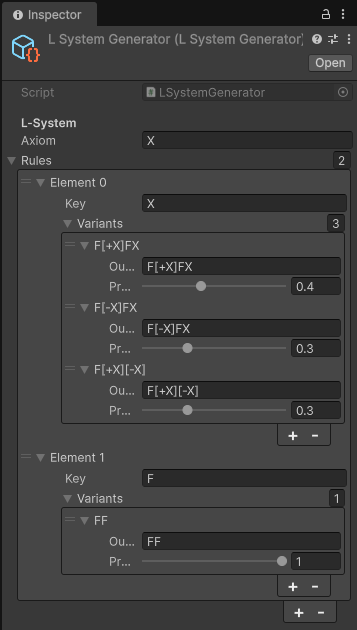


Рис. 2.1. – Аксіома та правила для L-системи

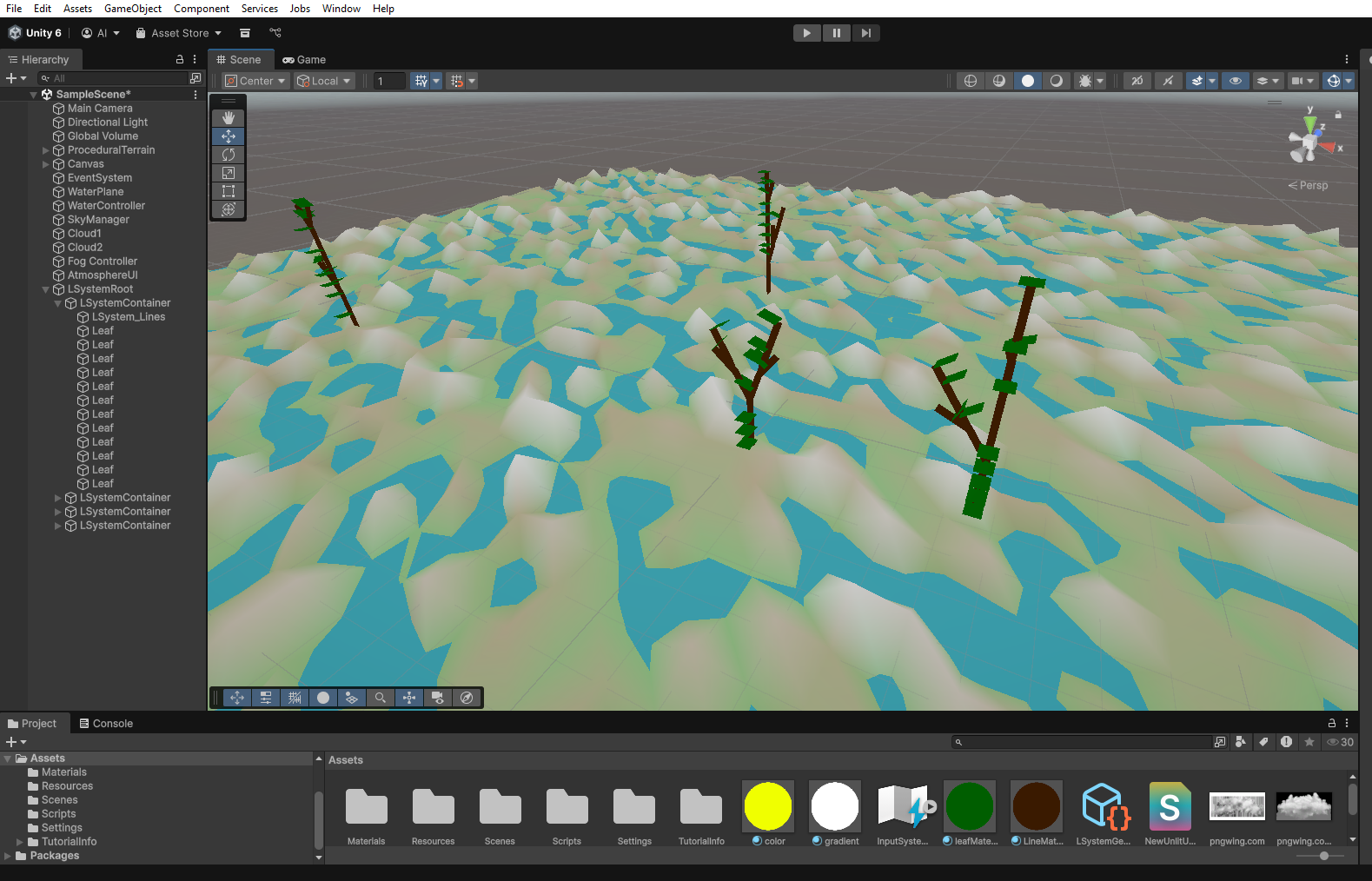


Рис. 2.1.2 – Згенеровані дерева

Результати виконання роботи наведені в таблиці 2.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 2.1 | | | | |
| № | Складність | Вимоги до роботи | Бали | Оцінка |
| 1 | Базовий рівень | Реалізовано генерацію 3D-фракталу (напр., проста 3D L-система) у вигляді каркасу з ліній (GL\_LINES або LineRenderer). | 4 | *4* |
| 2 | Підвищений рівень | Реалізовано складну L-систему та додано стохастичність (випадковість) у кути обертання, довжину гілок та/або  правило перетворення з вірогдністтю. | 2 | *2* |
| 3 | Самостійна робота | Згенероване 3D-дерево має об'єм  (використання програмно згенерованих Mesh циліндрів/конусів для гілок) та  додано листя. | 2 | *2* |
| 4 | Своєчасність | своєчасність виконання, бали:  достроково - 2, у строк - 1, з порушенням терміну виконання - 0 | 2 | *2* |

## 2.2. Функціональне моделювання комплексних об'єктів (Signed Distance Function)

### Завдання

Необхідно реалізувати візуалізацію тривимірної сцени з використанням **Signed Distance Function (SDF)** та алгоритму **Ray Marching**. Об’єкти сцени повинні формуватися на основі математичних функцій відстані.

### Теоретичні відомості

SDF дозволяють описувати геометрію об’єктів неявним способом, використовуючи математичні функції. Рендеринг таких сцен виконується за допомогою алгоритму Ray Marching, який покроково визначає відстань до найближчої поверхні та знаходить точку перетину променя з об’єктом.

### Результати виконання практичної роботи

Було реалізовано шейдер Ray Marching, який відображає сцену з декількох примітивів, створених за допомогою SDF. Для формування складних об’єктів використано булеві операції. Реалізовано освітлення та тіні, що значно покращує візуальну якість сцени.

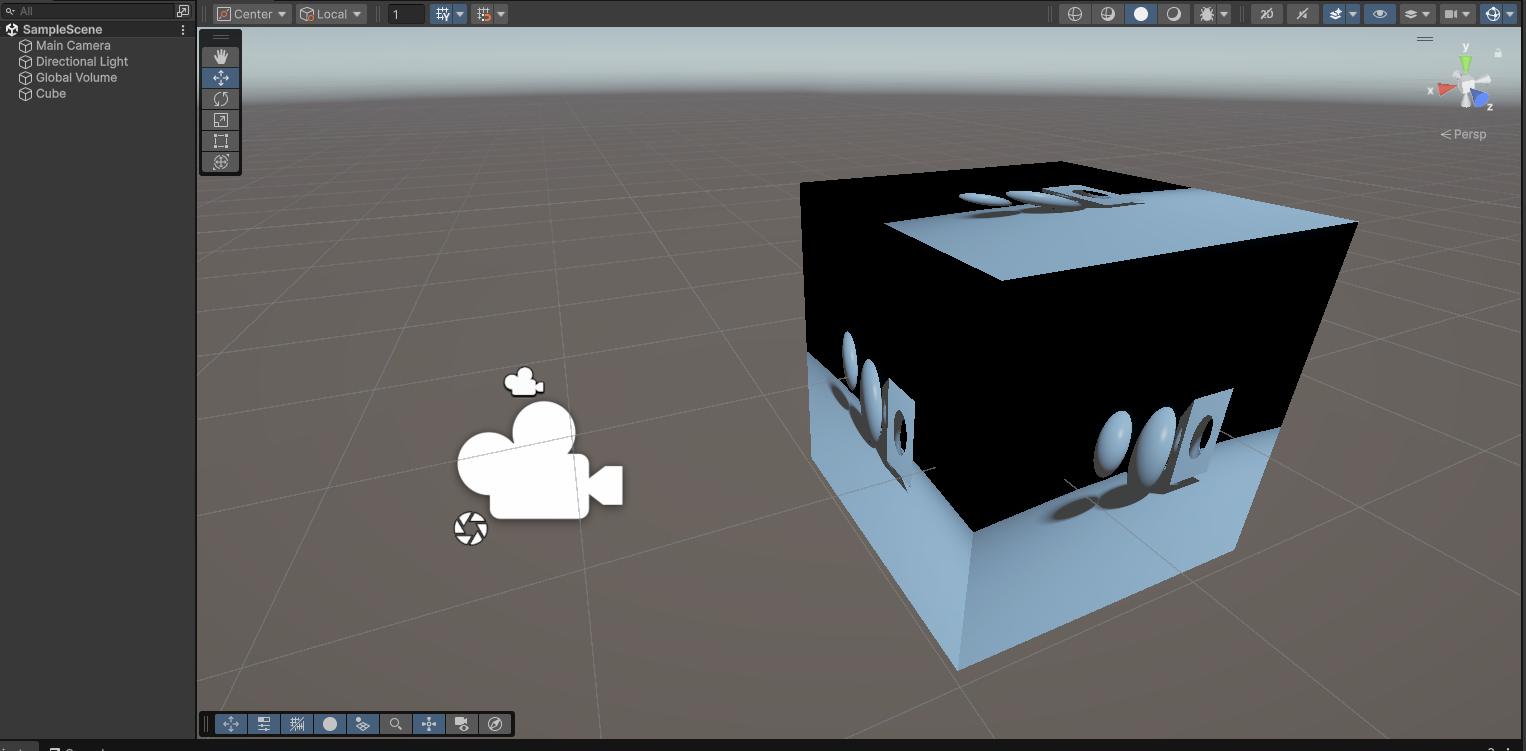


Рис. 2.2.1 – Примітив Unity, який слугує контейнером для SDF-сцени

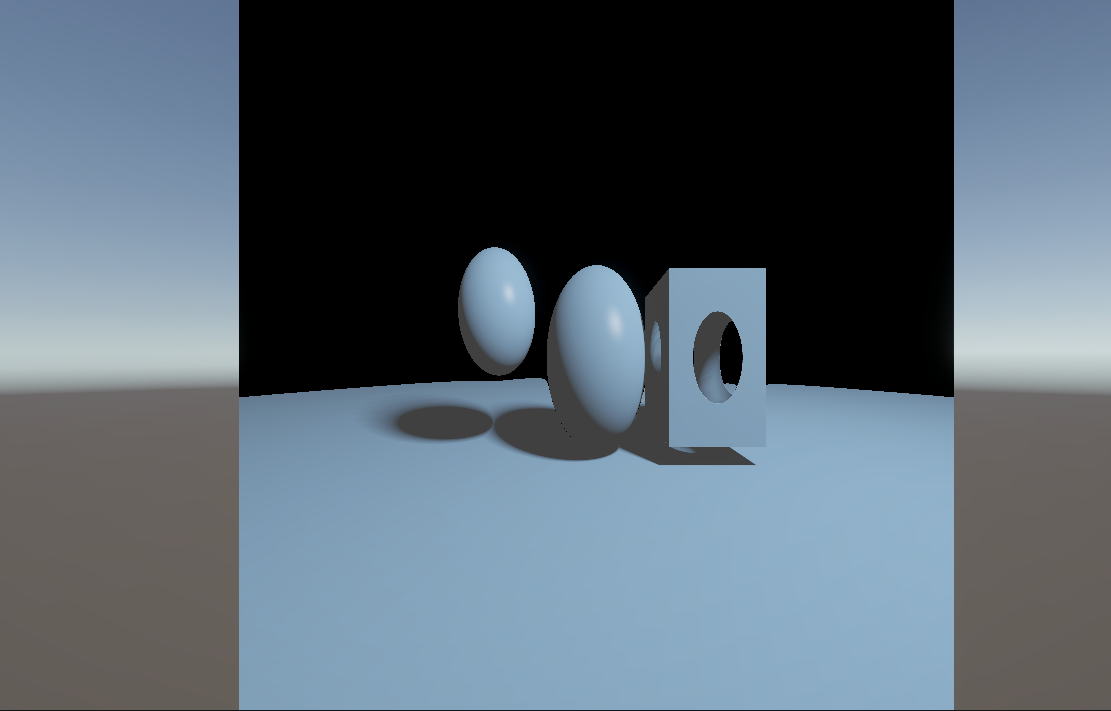


Рис. 2.2.2 – SDF-сцена

Результати виконання роботи наведені в таблиці 2.2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 2.2 | | | | |
| № | Складність | Вимоги до роботи | Бали | Оцінка |
| 1 | Базовий рівень | Реалізовано Ray Marching та рендеринг примітиву, який задано простою SDF-функцією. | 4 | *4* |
| 2 | Підвищений рівень | Реалізовано примітиви з використанням  булевих операцій (наприклад, об'єднання min()). | 2 | *2* |
| 3 | Самостійна робота | Реалізовано складну сцену (декілька об'єкти та булевих операцій) та коректне освітлення (з тінями, розрахованими через Ray Marching). | 2 | *2* |
| 4 | Своєчасність | своєчасність виконання, бали:  достроково - 2, у строк - 1, з порушенням терміну виконання - 0 | 2 | *2* |

# Загальний перелік посилань

1. Microsoft. glDrawArrays function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/gldrawarrays>.
2. Generating Fractal Trees in Unity - Режим доступу до ресурсу: <https://www.youtube.com/watch?v=VXegg-HGT0s>

# Додаток А. Лістинг програми до практичної роботи № 1.1

### Код файлу (TerrainGenerator.cs)

using System.Collections.Generic;

using UnityEngine;

[RequireComponent(typeof(MeshFilter), typeof(MeshRenderer))]

public class TerrainGenerator : MonoBehaviour

{

[Header("Coloring")]

public int colorMode = 0; // 0 - стандарт, 1 - сніжний, 2 - пустеля

[Header("Grid")]

public int width = 50;

public int length = 50;

public float cellSize = 1f;

[Header("Height (Perlin)")]

public float heightScale = 5f;

public float noiseScale = 0.1f;

public int seed = 0;

[Header("Display")]

public bool showWireframe = false;

public Material surfaceMaterial;

public Material lineMaterial;

Mesh mesh;

MeshFilter mf;

MeshRenderer mr;

GameObject wireframeObj;

MeshFilter wfFilter;

MeshRenderer wfRenderer;

void Awake()

{

mf = GetComponent<MeshFilter>();

mr = GetComponent<MeshRenderer>();

Generate();

}

[ContextMenu("Generate Terrain")]

public void Generate()

{

Random.InitState(seed);

if (mf == null) mf = GetComponent<MeshFilter>();

if (mr == null) mr = GetComponent<MeshRenderer>();

mesh = BuildMesh();

mf.sharedMesh = mesh;

if (surfaceMaterial != null)

mr.sharedMaterial = surfaceMaterial;

CreateOrUpdateWireframe(mesh);

UpdateWireframeVisibility();

}

Mesh BuildMesh()

{

int vertsX = width + 1;

int vertsZ = length + 1;

Vector3[] verts = new Vector3[vertsX \* vertsZ];

Color[] colors = new Color[verts.Length];

Vector2[] uv = new Vector2[verts.Length];

float minY = float.MaxValue;

float maxY = float.MinValue;

float offsetX = Random.Range(-10000f, 10000f);

float offsetZ = Random.Range(-10000f, 10000f);

for (int z = 0; z < vertsZ; z++)

{

for (int x = 0; x < vertsX; x++)

{

float px = x \* cellSize;

float pz = z \* cellSize;

float nx = (px + offsetX) \* noiseScale;

float nz = (pz + offsetZ) \* noiseScale;

float h = Mathf.PerlinNoise(nx, nz);

float y = (h - 0.5f) \* 2f \* heightScale;

int idx = z \* vertsX + x;

verts[idx] = new Vector3(px, y, pz);

uv[idx] = new Vector2((float)x / width, (float)z / length);

if (y < minY) minY = y;

if (y > maxY) maxY = y;

}

}

for (int i = 0; i < verts.Length; i++)

{

float t = Mathf.InverseLerp(minY, maxY, verts[i].y);

colors[i] = HeightToGradient(t);

}

int[] tris = new int[width \* length \* 6];

int tIdx = 0;

for (int z = 0; z < length; z++)

{

for (int x = 0; x < width; x++)

{

int i0 = z \* vertsX + x;

int i1 = i0 + 1;

int i2 = i0 + vertsX;

int i3 = i2 + 1;

tris[tIdx++] = i0;

tris[tIdx++] = i2;

tris[tIdx++] = i1;

tris[tIdx++] = i1;

tris[tIdx++] = i2;

tris[tIdx++] = i3;

}

}

Mesh m = new Mesh();

m.indexFormat = verts.Length > 65535 ? UnityEngine.Rendering.IndexFormat.UInt32 : UnityEngine.Rendering.IndexFormat.UInt16;

m.vertices = verts;

m.triangles = tris;

m.uv = uv;

m.colors = colors;

m.RecalculateNormals();

m.RecalculateBounds();

return m;

}

Color HeightToGradient(float t)

{

switch (colorMode)

{

case 1: // сніжний

return Color.Lerp(new Color(0.6f, 0.7f, 0.8f), Color.white, t);

case 2: // пустеля

return Color.Lerp(new Color(0.8f, 0.7f, 0.4f), new Color(0.9f, 0.85f, 0.5f), t);

default: // стандарт

if (t < 0.35f)

return Color.Lerp(new Color(0.05f, 0.25f, 0.05f), new Color(0.2f, 0.6f, 0.2f), t / 0.35f);

else if (t < 0.7f)

return Color.Lerp(new Color(0.2f, 0.6f, 0.2f), new Color(0.5f, 0.4f, 0.25f), (t - 0.35f) / 0.35f);

else

return Color.Lerp(new Color(0.5f, 0.4f, 0.25f), Color.white, (t - 0.7f) / 0.3f);

}

}

void CreateOrUpdateWireframe(Mesh sourceMesh)

{

if (wireframeObj == null)

{

wireframeObj = new GameObject("Wireframe");

wireframeObj.transform.SetParent(transform, false);

wfFilter = wireframeObj.AddComponent<MeshFilter>();

wfRenderer = wireframeObj.AddComponent<MeshRenderer>();

wireframeObj.transform.localPosition = Vector3.zero;

}

// створення унікального набору ребер

Vector3[] verts = sourceMesh.vertices;

int[] tris = sourceMesh.triangles;

var edgeSet = new HashSet<Edge>();

for (int i = 0; i < tris.Length; i += 3)

{

int a = tris[i];

int b = tris[i + 1];

int c = tris[i + 2];

AddEdge(edgeSet, a, b);

AddEdge(edgeSet, b, c);

AddEdge(edgeSet, c, a);

}

// створення лінійної сітки за допомогою топології Lines

Mesh lineMesh = new Mesh();

List<Vector3> lineVerts = new List<Vector3>();

List<int> lineIndices = new List<int>();

int idx = 0;

foreach (var e in edgeSet)

{

lineVerts.Add(verts[e.v1]);

lineVerts.Add(verts[e.v2]);

lineIndices.Add(idx++);

lineIndices.Add(idx++);

}

lineMesh.SetVertices(lineVerts);

lineMesh.SetIndices(lineIndices.ToArray(), MeshTopology.Lines, 0);

lineMesh.RecalculateBounds();

wfFilter.sharedMesh = lineMesh;

if (lineMaterial != null) wfRenderer.sharedMaterial = lineMaterial;

}

void AddEdge(HashSet<Edge> set, int a, int b)

{

Edge e = new Edge(a, b);

if (set.Contains(e)) return;

set.Add(e);

}

struct Edge

{

public int v1, v2;

public Edge(int a, int b)

{

if (a < b) { v1 = a; v2 = b; }

else { v1 = b; v2 = a; }

}

}

void UpdateWireframeVisibility()

{

if (wireframeObj != null)

wireframeObj.SetActive(showWireframe);

}

void OnValidate()

{

if (!Application.isPlaying)

{

try { Generate(); } catch { }

}

else

{

UpdateWireframeVisibility();

}

}

}

### Код файлу (TerrainUIController.cs)

using UnityEngine;

using UnityEngine.UI;

public class TerrainUIController : MonoBehaviour

{

public TerrainGenerator terrain;

public Slider heightSlider;

public Slider noiseSlider;

public Dropdown colorModeDropdown;

public Toggle wireframeToggle;

void Start()

{

// Ініціалізируємо UI за даними значеннями

heightSlider.value = terrain.heightScale;

noiseSlider.value = terrain.noiseScale;

wireframeToggle.isOn = terrain.showWireframe;

}

public void OnHeightChanged(float value)

{

terrain.heightScale = value;

terrain.Generate();

}

public void OnNoiseChanged(float value)

{

terrain.noiseScale = value;

terrain.Generate();

}

public void OnWireframeToggled(bool state)

{

terrain.showWireframe = state;

terrain.Generate();

}

public void OnColorModeChanged(int index)

{

terrain.colorMode = index;

terrain.Generate();

}

public void OnRegenerateClicked()

{

terrain.Generate();

}

}

# Лістинг програми до практичної роботи № 1.2

***Код файлу (WaterShader.shader)***

Shader "Custom/SimpleWater"

{

Properties

{

\_Color ("Water Color", Color) = (0,0.5,0.7,0.6)

\_WaveSpeed ("Wave Speed", Range(0.1,5)) = 1

\_WaveHeight ("Wave Height", Range(0,0.5)) = 0.1

\_Transparency ("Transparency", Range(0,1)) = 0.5

}

SubShader

{

Tags { "Queue"="Transparent" "RenderType"="Transparent" }

LOD 200

Blend SrcAlpha OneMinusSrcAlpha

ZWrite Off

Cull Off

Pass

{

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#include "UnityCG.cginc"

struct appdata

{

float4 vertex : POSITION;

float2 uv : TEXCOORD0;

};

struct v2f

{

float4 pos : SV\_POSITION;

float2 uv : TEXCOORD0;

};

float \_TimeY;

float4 \_Color;

float \_WaveSpeed;

float \_WaveHeight;

float \_Transparency;

v2f vert (appdata v)

{

v2f o;

float wave = sin(v.vertex.x \* 0.1 + \_Time.y \* \_WaveSpeed) \*\_WaveHeight

+ cos(v.vertex.z \* 0.1 + \_Time.y \* \_WaveSpeed)\*\_WaveHeight;

v.vertex.y += wave;

o.pos = UnityObjectToClipPos(v.vertex);

o.uv = v.uv;

return o;

}

fixed4 frag (v2f i) : SV\_Target

{

fixed4 col = \_Color;

col.a = \_Transparency;

return col;

}

ENDCG

}

}

}

***Код файлу (WaterController.cs)***

using UnityEngine;

using UnityEngine.UI;

public class WaterController : MonoBehaviour

{

public Material waterMaterial;

[Header("UI Elements")]

public Slider transparencySlider;

public Slider depthSlider;

private float baseHeight = 0f;

void Start()

{

if (waterMaterial != null)

{

transparencySlider.value = waterMaterial.GetFloat("\_Transparency");

}

baseHeight = transform.position.y;

transparencySlider.onValueChanged.AddListener(OnTransparencyChanged);

depthSlider.onValueChanged.AddListener(OnDepthChanged);

}

// Регулювання прозорості води

public void OnTransparencyChanged(float value)

{

if (waterMaterial != null)

waterMaterial.SetFloat("\_Transparency", value);

}

// Регулювання глибини води

public void OnDepthChanged(float value)

{

Debug.Log($"Depth changed: {value}");

transform.position = new Vector3(transform.position.x, baseHeight + value, transform.position.z);

}

}

# Лістинг програми до практичної роботи № 1.3

***Код файлу (TumanControlled.cs)***

using UnityEngine;

public class TumanControlled : MonoBehaviour

{

[Range(0, 0.1f)] public float fogDensity = 0.02f;

public Color fogColor = new Color(0.7f, 0.8f, 0.9f);

void Start()

{

RenderSettings.fog = true;

RenderSettings.fogMode = FogMode.ExponentialSquared;

}

void Update()

{

RenderSettings.fogDensity = fogDensity;

RenderSettings.fogColor = fogColor;

}

}

***Код файлу (AtmosphereUI.cs)***

using UnityEngine;

using UnityEngine.UI;

public class AtmosphereUI : MonoBehaviour

{

public SkyManager sky;

public TumanControlled fog;

public CloudBase[] clouds;

public Slider timeSlider;

public Slider fogSlider;

public Slider cloudSpeedSlider;

void Start()

{

timeSlider.onValueChanged.AddListener(OnTimeChanged);

fogSlider.onValueChanged.AddListener(OnFogChanged);

cloudSpeedSlider.onValueChanged.AddListener(OnCloudSpeedChanged);

}

void OnTimeChanged(float val)

{

sky.timeOfDay = val;

}

void OnFogChanged(float val)

{

fog.fogDensity = val;

}

void OnCloudSpeedChanged(float val)

{

foreach (var c in clouds)

c.speed = val;

}

}

***Код файлу (CloudBase.cs)***

using UnityEngine;

public class CloudBase : MonoBehaviour

{

public float speed = 0.5f;

protected Material mat;

protected virtual void Start()

{

mat = GetComponent<MeshRenderer>().material;

}

protected virtual void Update()

{

Animate();

}

protected void Animate()

{

Vector2 offset = mat.mainTextureOffset;

offset.x += Time.deltaTime \* speed \* 0.02f;

mat.mainTextureOffset = offset;

}

}

***Код файлу (CirrusClouds.cs)***

using UnityEngine;

public class CirrusClouds : CloudBase

{

protected override void Start()

{

base.Start();

mat.color = new Color(1f, 1f, 1f, 0.5f);

speed \*= 2f;

}

}

***Код файлу (CumulusClouds.cs)***

using UnityEngine;

public class CumulusClouds : CloudBase

{

protected override void Start()

{

base.Start();

mat.color = new Color(1f, 1f, 1f, 0.8f);

}

}

***Код файлу (SkyManager.cs)***

using UnityEngine;

public class SkyManager : MonoBehaviour

{

[Range(0, 24)] public float timeOfDay = 12f; // час (0–24)

public Light sun;

public Gradient skyColor; // градієнт кольору неба

public Gradient ambientColor; // цвет освітлення

void Update()

{

UpdateSunPosition();

UpdateLighting();

}

void UpdateSunPosition()

{

// Обертання сонця цілодобово

float rotation = (timeOfDay / 24f) \* 360f - 90f;

sun.transform.rotation = Quaternion.Euler(rotation, 170, 0);

}

void UpdateLighting()

{

RenderSettings.ambientLight = ambientColor.Evaluate(timeOfDay / 24f);

Camera.main.backgroundColor = skyColor.Evaluate(timeOfDay / 24f);

sun.intensity = Mathf.Clamp01(Mathf.Cos((timeOfDay - 12) / 12f \* Mathf.PI));

}

}

# Додаток Б. Лістинг програми до практичної роботи № 2.1

***Код файлу (LSystemInterpreter.cs)***

using System.Collections.Generic;

using UnityEngine;

public class LSystemInterpreter : MonoBehaviour

{

[Header("Generator")]

public LSystemGenerator generator;

public int iterations = 4;

[Header("Turtle")]

public float baseLength = 1.0f;

[Range(0f, 1f)] public float lengthVariance = 0.2f;

public float baseAngle = 25f;

public float angleVariance = 10f;

[Header("Branch mesh")]

public bool generateMeshBranches = true;

public Material branchMaterial;

public float startRadius = 0.2f;

public float radiusDecay = 0.6f;

public int radialSegments = 10;

[Header("Leaves")]

public Material leafMaterial;

public float leafSize = 0.25f;

public float leafProbability = 0.5f;

[Header("Display")]

public bool useLineRendererFallback = false;

public Material lineMaterial;

public bool showWireframe = false;

private Transform container;

[ContextMenu("Generate L-System")]

public void Generate()

{

Clear();

if (generator == null)

{

Debug.LogError("LSystemGenerator asset not assigned!");

return;

}

string system = generator.Generate(iterations);

Interpret(system);

}

public void Clear()

{

if (container != null) DestroyImmediate(container.gameObject);

container = new GameObject("LSystemContainer").transform;

container.SetParent(transform, false);

}

struct TurtleState

{

public Vector3 pos;

public Quaternion rot;

public float radius;

public TurtleState Clone() => new TurtleState { pos = pos, rot = rot, radius = radius };

}

void Interpret(string system)

{

Stack<TurtleState> stack = new Stack<TurtleState>();

TurtleState state = new TurtleState

{

pos = transform.position,

rot = transform.rotation,

radius = startRadius

};

Vector3 lastPos = state.pos;

LineRenderer lr = null;

if (useLineRendererFallback)

{

var go = new GameObject("LSystem\_Lines");

go.transform.SetParent(container, false);

lr = go.AddComponent<LineRenderer>();

lr.material = lineMaterial;

lr.widthMultiplier = startRadius \* 0.8f;

lr.positionCount = 0;

lr.useWorldSpace = false;

}

foreach (char c in system)

{

if (c == 'F')

{

float len = baseLength \* Random.Range(1f - lengthVariance, 1f + lengthVariance);

Vector3 newPos = state.pos + state.rot \* (Vector3.up \* len);

if (useLineRendererFallback)

{

lr.positionCount += 2;

lr.SetPosition(lr.positionCount - 2, state.pos);

lr.SetPosition(lr.positionCount - 1, newPos);

}

else if (generateMeshBranches)

{

Mesh mesh = BranchMeshGenerator.CreateCylinderMesh(state.pos, newPos, state.radius, state.radius \* radiusDecay, radialSegments);

GameObject branch = new GameObject("Branch");

branch.transform.SetParent(container, true);

var mf = branch.AddComponent<MeshFilter>();

mf.sharedMesh = mesh;

var mr = branch.AddComponent<MeshRenderer>();

mr.sharedMaterial = branchMaterial;

if (showWireframe)

{

var lrObj = new GameObject("wire");

lrObj.transform.SetParent(branch.transform, false);

var l = lrObj.AddComponent<LineRenderer>();

l.material = lineMaterial;

l.positionCount = 2;

l.SetPosition(0, state.pos);

l.SetPosition(1, newPos);

l.widthMultiplier = Mathf.Max(0.01f, state.radius \* 0.2f);

}

}

if (Random.value < leafProbability)

{

var leafRot = Quaternion.LookRotation((state.rot \* Vector3.up).normalized);

LeafGenerator.CreateLeafObject(newPos, leafRot, container, leafMaterial, leafSize);

}

state.pos = newPos;

state.radius \*= radiusDecay;

}

else if (c == '+')

{

float a = baseAngle + Random.Range(-angleVariance, angleVariance);

state.rot \*= Quaternion.Euler(0, 0, a);

}

else if (c == '-')

{

float a = baseAngle + Random.Range(-angleVariance, angleVariance);

state.rot \*= Quaternion.Euler(0, 0, -a);

}

else if (c == '&')

{

float a = baseAngle + Random.Range(-angleVariance, angleVariance);

state.rot \*= Quaternion.Euler(a, 0, 0);

}

else if (c == '^')

{

float a = baseAngle + Random.Range(-angleVariance, angleVariance);

state.rot \*= Quaternion.Euler(-a, 0, 0);

}

else if (c == '\\')

{

float a = baseAngle + Random.Range(-angleVariance, angleVariance);

state.rot \*= Quaternion.Euler(0, a, 0);

}

else if (c == '/')

{

float a = baseAngle + Random.Range(-angleVariance, angleVariance);

state.rot \*= Quaternion.Euler(0, -a, 0);

}

else if (c == '[')

{

stack.Push(state.Clone());

}

else if (c == ']')

{

if (stack.Count > 0) state = stack.Pop();

}

else if (c == 'L')

{

var leafRot = Quaternion.LookRotation((state.rot \* Vector3.up).normalized);

LeafGenerator.CreateLeafObject(state.pos, leafRot, container, leafMaterial, leafSize);

}

lastPos = state.pos;

}

}

}

***Код файлу (LSystemGenerator.cs)***

using System.Collections.Generic;

using System.Text;

using UnityEngine;

[CreateAssetMenu(fileName = "LSystemGenerator", menuName = "Procedural/LSystemGenerator")]

public class LSystemGenerator : ScriptableObject

{

[System.Serializable]

public class RuleVariant

{

public string output;

[Range(0f, 1f)] public float probability = 1f;

}

[System.Serializable]

public class Rule

{

public char key;

public List<RuleVariant> variants = new List<RuleVariant>();

}

[Header("L-System")]

public string axiom = "F";

public List<Rule> rules = new List<Rule>();

public string Generate(int iterations)

{

string current = axiom;

for (int it = 0; it < iterations; it++)

{

var sb = new StringBuilder();

foreach (char c in current)

{

Rule r = rules.Find(x => x.key == c);

if (r != null && r.variants.Count > 0)

{

sb.Append(ApplyStochasticRule(r.variants));

}

else

{

sb.Append(c);

}

}

current = sb.ToString();

}

return current;

}

string ApplyStochasticRule(List<RuleVariant> variants)

{

float sum = 0f;

foreach (var v in variants) sum += v.probability;

if (sum <= 0f) return variants[0].output;

float r = Random.value \* sum;

float acc = 0f;

foreach (var v in variants)

{

acc += v.probability;

if (r <= acc) return v.output;

}

return variants[variants.Count - 1].output;

}

}

***Код файлу (UnlitDoubleSided.shader)***

Shader "Custom/UnlitDoubleSided"

{

Properties

{

\_Color ("Color", Color) = (0,1,0,1)

\_MainTex ("Texture", 2D) = "white" {}

\_Cutoff ("Alpha Cutoff", Range(0,1)) = 0.4

}

SubShader

{

Tags { "Queue"="AlphaTest" "RenderType"="TransparentCutout" }

Cull Off

Pass

{

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#include "UnityCG.cginc"

struct appdata

{

float4 vertex : POSITION;

float2 uv : TEXCOORD0;

};

struct v2f

{

float4 pos : SV\_POSITION;

float2 uv : TEXCOORD0;

};

sampler2D \_MainTex;

fixed4 \_Color;

float \_Cutoff;

v2f vert (appdata v)

{

v2f o;

o.pos = UnityObjectToClipPos(v.vertex);

o.uv = v.uv;

return o;

}

fixed4 frag (v2f i) : SV\_Target

{

fixed4 col = tex2D(\_MainTex, i.uv) \* \_Color;

clip(col.a - \_Cutoff);

return col;

}

ENDCG

}

}

}

***Код файлу (LeafGenerator.cs)***

using UnityEngine;

public static class LeafGenerator

{

public static Mesh CreateLeafMesh(float width = 0.2f, float height = 0.3f)

{

Mesh m = new Mesh();

Vector3[] verts = new Vector3[4]

{

new Vector3(-width\*0.5f, 0f, 0f),

new Vector3(width\*0.5f, 0f, 0f),

new Vector3(-width\*0.5f, height, 0f),

new Vector3(width\*0.5f, height, 0f)

};

int[] tris = new int[6] { 0, 2, 1, 1, 2, 3 };

Vector3[] normals = new Vector3[4] { Vector3.forward, Vector3.forward, Vector3.forward, Vector3.forward };

Vector2[] uv = new Vector2[4] { new Vector2(0, 0), new Vector2(1, 0), new Vector2(0, 1), new Vector2(1, 1) };

m.vertices = verts;

m.triangles = tris;

m.normals = normals;

m.uv = uv;

m.RecalculateBounds();

return m;

}

public static GameObject CreateLeafObject(Vector3 position, Quaternion rotation, Transform parent, Material mat, float size = 0.25f)

{

GameObject go = new GameObject("Leaf");

go.transform.SetParent(parent, false);

go.transform.localPosition = position;

go.transform.localRotation = rotation;

go.transform.localScale = Vector3.one \* size;

var mf = go.AddComponent<MeshFilter>();

mf.mesh = CreateLeafMesh(1f, 1.2f);

var mr = go.AddComponent<MeshRenderer>();

mr.sharedMaterial = mat;

return go;

}

}

***Код файлу (BranchMeshGenerator.cs)***

using System.Collections.Generic;

using UnityEngine;

public static class BranchMeshGenerator

{

public static Mesh CreateCylinderMesh(Vector3 start, Vector3 end, float startRadius, float endRadius, int radialSegments = 12)

{

Mesh m = new Mesh();

Vector3 axis = end - start;

float height = axis.magnitude;

if (height <= 1e-6f) return m;

var verts = new List<Vector3>();

var normals = new List<Vector3>();

var uvs = new List<Vector2>();

var tris = new List<int>();

Vector3 up = axis.normalized;

Vector3 forward = Vector3.Slerp(up, Vector3.up, 0.0f);

Vector3 side = Vector3.Cross(up, forward);

if (side.sqrMagnitude < 1e-6f) side = Vector3.Cross(up, Vector3.right);

side.Normalize();

Vector3 forwardFixed = Vector3.Cross(side, up).normalized;

for (int i = 0; i < radialSegments; i++)

{

float theta = (2f \* Mathf.PI \* i) / radialSegments;

float cx = Mathf.Cos(theta);

float cz = Mathf.Sin(theta);

Vector3 dir = side \* cx + forwardFixed \* cz;

verts.Add(start + dir \* startRadius);

normals.Add(dir);

uvs.Add(new Vector2(i / (float)radialSegments, 0));

verts.Add(start + axis + dir \* endRadius);

normals.Add(dir);

uvs.Add(new Vector2(i / (float)radialSegments, 1));

}

for (int i = 0; i < radialSegments; i++)

{

int i0 = i \* 2;

int i1 = ((i + 1) % radialSegments) \* 2;

tris.Add(i0);

tris.Add(i0 + 1);

tris.Add(i1 + 1);

tris.Add(i0);

tris.Add(i1 + 1);

tris.Add(i1);

}

m.SetVertices(verts);

m.SetNormals(normals);

m.SetUVs(0, uvs);

m.SetTriangles(tris, 0);

m.RecalculateBounds();

return m;

}

}

# Лістинг програми до практичної роботи № 2.2

***Код файлу (RayMarching.shader)***

Shader "Custom/RayMarching"

{

Properties

{

\_MaxDistance ("Max Distance", Float) = 50

\_MaxSteps ("Max Steps", Int) = 128

\_Epsilon ("Surface Epsilon", Float) = 0.001

}

SubShader

{

Tags { "RenderType"="Opaque" }

Pass

{

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#include "UnityCG.cginc"

struct appdata

{

float4 vertex : POSITION;

float2 uv : TEXCOORD0;

};

struct v2f

{

float4 pos : SV\_POSITION;

float2 uv : TEXCOORD0;

};

float \_MaxDistance;

int \_MaxSteps;

float \_Epsilon;

// SDF

float sdSphere(float3 p, float r)

{

return length(p) - r;

}

float sdBox(float3 p, float3 b)

{

float3 q = abs(p) - b;

return length(max(q, 0)) + min(max(q.x, max(q.y, q.z)), 0);

}

float opUnion(float d1, float d2)

{

return min(d1, d2);

}

float opSubtraction(float d1, float d2)

{

return max(d1, -d2);

}

float opIntersection(float d1, float d2)

{

return max(d1, d2);

}

float map(float3 p)

{

// Підлога

float ground = p.y + 1.0;

// Сфера 1

float sphere1 = sdSphere(p, 1.0);

// Куб

float box = sdBox(p - float3(2.0, 0.0, 0.0), float3(0.8, 0.8, 0.8));

float sphereCut = sdSphere(p - float3(2.0, 0.0, 0.0), 0.9);

float hollowBox = opSubtraction(box, sphereCut);

// Сфера 2

float sphere2 = sdSphere(p - float3(-2.0, 0.5, 0.0), 0.7);

// Сцена

float scene = opUnion(sphere1, hollowBox);

scene = opUnion(scene, sphere2);

scene = opUnion(scene, ground);

return scene;

}

// RAY Marching

float rayMarch(float3 ro, float3 rd)

{

float t = 0.0;

for (int i = 0; i < \_MaxSteps; i++)

{

float3 p = ro + rd \* t;

float d = map(p);

if (d < \_Epsilon)

return t;

t += d;

if (t > \_MaxDistance)

break;

}

return -1.0;

}

v2f vert(appdata v)

{

v2f o;

o.pos = UnityObjectToClipPos(v.vertex);

o.uv = v.uv;

return o;

}

float3 calcNormal(float3 p)

{

float e = 0.001;

float dx = map(p + float3(e, 0, 0)) - map(p - float3(e, 0, 0));

float dy = map(p + float3(0, e, 0)) - map(p - float3(0, e, 0));

float dz = map(p + float3(0, 0, e)) - map(p - float3(0, 0, e));

return normalize(float3(dx, dy, dz));

}

float lambert(float3 n, float3 l)

{

return max(0, dot(n, l));

}

float phong(float3 n, float3 l, float3 v)

{

float3 r = reflect(-l, n);

return pow(max(dot(v, r), 0), 32);

}

float softShadow(float3 ro, float3 rd, float maxDist)

{

float t = 0.02;

float res = 1.0;

for (int i = 0; i < 64; i++)

{

float h = map(ro + rd \* t);

if (h < \_Epsilon)

return 0.0;

res = min(res, 10.0 \* h / t);

t += h;

if (t > maxDist) break;

}

return saturate(res);

}

fixed4 frag(v2f i) : SV\_Target

{

float2 uv = i.uv \* 2.0 - 1.0;

uv.x \*= \_ScreenParams.x / \_ScreenParams.y;

float3 ro = float3(0, 0, -4);

float3 rd = normalize(float3(uv, 1));

float t = rayMarch(ro, rd);

if (t > 0)

{

float3 p = ro + rd \* t;

float3 normal = calcNormal(p);

float3 lightPos = float3(5, 5, -5);

float3 lightDir = normalize(lightPos - p);

float3 viewDir = normalize(ro - p);

float diff = lambert(normal, lightDir);

float spec = phong(normal, lightDir, viewDir);

float shadow = softShadow(p + normal \* 0.01, lightDir, 10.0);

float3 color = float3(0.3, 0.6, 1.0);

float3 result = color \* diff;

result += spec \* 0.4;

result \*= shadow;

result += 0.05;

return fixed4(result, 1);

}

return fixed4(0, 0, 0, 1);

}

ENDCG

}

}

}