МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА ПРОГРАМУВАННЯ ТА МАТЕМАТИКИ

До захисту допускається

Завідувач кафедри

_____ Лифар В.О.

«____»_____ 2021 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

_____бакалавр_____

(освітньо-кваліфікаційний рівень)

НА ТЕМУ:

Програмне забезпечення рандомної генерації ландшафту для побудови 3D-сцен

Керівник роботи: _____ Захожай О. І.
(підпис) (ініціали, прізвище)

Студент: _____ Наумов А. В.
(підпис) (ініціали, прізвище)

Група: ІПЗ-17д

Сєвєродонецьк 2021

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

дипломної роботи студента гр. ІПЗ-17д Наумова А.В.

Науковий керівник

Доцент, д.т.н. _____ Захожай О. І.

Оцінка наукового керівника: _____

Рецензент: _____

ПІБ, місто роботи, посада

Оцінка рецензента: _____

Кінцева оцінка за результатами захисту:

_____

Голова ЕК _____ Лифар В. О.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Програмування та математики
Освітньо-кваліфікаційний рівень бакалавр
Напрям підготовки _____
(шифр і назва)
Спеціальність 121 «Інженерія програмного забезпеченя»
(шифр і назва)

ЗАТВЕРДЖУЮ:
Завідувач кафедри ПМ
_____В. О. Лифар
«_____» _____2021р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Наумов Антон В'ячеславович
(прізвище, ім'я, по батькові)

1. Тема роботи: Програмне забезпечення рандомної генерації ландшафту для побудови 3D-сцен
2. Керівник роботи: Захожай О. І., доктор технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від « » ___ 20___ №
3. Строк подання студентом роботи 07 червня 2021
4. Вихідні дані до роботи Аналіз варіантів вирішення алгоритмічної задачі. Розробка програмного забезпечення.
5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Аналітичний огляд, розробка програмного забезпечення генерації і візуалізації ланшафту, висновки.
6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

7. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

8. Дата видачі завдання 22 березня 2021 року

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Складання плану роботи	22.03.21 - 24.03.21	
2	Аналіз літератури	24.03.21 – 29.03.21	
3	Вивчення і підбирання матеріалу	29.03.21 – 20.04.21	
4	Написання розділів	20.04.21 – 25.05.21	
5	Оформлення пояснювальної записки	25.05.21 – 03.06.21	
6	Підготовка доповіді і слайдів для презентації	03.06.21 – 07.06.21	

Студент

_____ _____
(підпис) (прізвище та ініціали)

Науковий керівник

_____ _____
(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПК	персональний комп’ютер
ОС	операційна система
СКВ	система керування версіями
СК	система координат
ГПВЧ	генератор псевдовипадкових чисел
ДСТУ	державні стандарти України

ВСТУП

Все більшу роль у нашому житті грає різного роду комп'ютерні системи: складно уявити собі супермаркет без ПК або спеціалізованих касових систем на касах, у банках всі розрахунки проводяться на комп'ютерах, більша частина розрахунків при проектуванні нових будівель в будівельній фірмі проводиться на комп'ютерних системах і є ще велика кількість прикладів, як тісно комп'ютерні системи увійшли в наше життя.

Однією з областей застосування комп'ютерних систем у якості інструменту є – комп'ютерна графіка, розділом якої є тривимірна графіка.

Тривимірна графіка активно застосовується в науці і промисловості, наприклад, САПР, медична візуалізація, архітектурна візуалізація, комп'ютерні ігри.

Віртуальні ландшафти – одна з галузей тривимірної графіки, яка займається проектуванням і створенням тривимірних реалістичних ландшафтів.

Віртуальні ландшафти застосовуються для рішення широкого спектра практичних завдань, ось деякі з них:

- навчання пілотів і водіїв, на симуляторах;
- віртуальне випробування моделей транспортних засобів;
- розміщення презентаційних характеристик моделей продукції;
- створення ігрових сцен: віртуальні ландшафти активно застосовуються в ігровій індустрії;
- створення кінематографічних сцен;
- художній дизайн;
- аналіз сканованих ландшафтів може застосовуватися при пошуку корисних копалин і нафторозвідки;

Віртуальні ландшафти або заносяться в комп'ютер ззовні (дані може заносити художник або ж вони можуть заноситися сканером або подібним

обладнанням, на основі реальних ландшафтів), або генеруються автоматично за допомогою різних алгоритмів і формул (залежностей). До плюсів перших слід віднести більшу реалістичність і як правило велику барвистість, але тут є і мінуси: цей метод вимагає, як правило, великих витрат і більшого часу, крім того розмір отриманих таким чином ландшафтів як правило сильно обмежений. До плюсів других слід віднести швидкість генерації, крім того розмір згенерованих автоматично ландшафтів обмежується користувачем: існують алгоритми, які дозволяють генерувати практично безкраї простори – доки дозволяє пам'ять комп'ютера.

У даній роботі мова піде про систему, що виробляє генерацію віртуального ландшафту різними алгоритмами і візуалізує зображення реалістичного тривимірного ландшафту в реальному часі. Для вирішення цього завдання, як вже було згадано, в комп'ютерній графіці існує ціла гілка. Вона займається розробкою алгоритмів перетворення вхідних даних і побудовою на їх основі ландшафтів.

1.АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Характеристика області

1.1.1 Вступ

Існують два типи програм генераторів ландшафтів: перший, це відповідно генератори ландшафтів, другий – системи моделювання ландшафтів, з функцією автоматичної генерації. На даний момент, системи другого типу поширені більше. Це обумовлено тим, що результат генерації мало передбачуваний і як правило система моделювання відображає згенерований ландшафт певним чином (параметри відображення для ландшафтів, згенерованих рідним генератором налаштовані спеціальним чином, для одержання більш реалістичного зображення).

Рішення поставленої задачі передбачає вивчення вже існуючих алгоритмів і методів генерації та відображення ландшафту, створення генератора ландшафтів, створення програми візуалізації побудованих ландшафтів, тестування створених програм. В рамках даної роботи розглядаються існуючі алгоритми вирішення поставлених завдань, а також проведений аналіз інструментів для вирішення поставлених завдань.

Генерація ландшафту включає в себе такі етапи, як:

• створення карти висот;

• створення карти освітленості;

• генерація текстури;

• генерація карти об'єктів.

Файл ландшафту як мінімум повинний містити в собі карту висот і заголовок, який включає в себе як мінімум формат і розміри ландшафту.

Створення карти висот як правило включає в себе наступні етапи:

• генерація ключових точок на основі вхідних параметрів;

• генерація ландшафту в околицях ключових точок;

• нормалізацію отриманого ландшафту;

• обробка ландшафту різними фільтрами – згладжування, «долинізація» та інші.

Візуалізація ландшафту, як правило, включає в себе такі етапи, як:

1. всі об'єкти ландшафту поміщаються в площину;

2. призначаються параметри поверхонь об'єктів і встановлюється освітлення;

3. всі полігони перевіряються на предмет видимості і невидимі видаляються;

4. обрізані і коректно спотворені полігони проектуються на площину (як ніби на екран монітора).

Тестування включає в себе модульне тестування (тестування окремих функцій на етапі написання коду) і системне тестування програмного забезпечення на предмет надійності, ефективності і практичності.

1.1.2 Підстава для розробки

Система розробляється на підставі наказу ректора інституту на випускну кваліфікаційну роботу за направленням бакалавр.

1.1.3 Призначення

Програмне забезпечення призначене для генерації і побудови ландшафтів в реальному часі. Генерація здійснюється з використанням різних методів і параметрів, що може бути корисно при різних наукових дослідженнях.

1.1.4 Вимоги до програми або програмного виробу. Вимоги до функціональних характеристик

Система повинна представляти сукупність методичних і програмних засобів для генерації і побудови тривимірних віртуальних ландшафтів. Ландшафти повинні генеруватися з використанням декількох різних методів

генерації. Також при генерації повинні враховуватися різні параметри, різні для різних алгоритмів. Візуалізатор повинен візуалізувати згенеровані ландшафти з урахуванням текстурної карти та іншого.

Вимоги до складу і параметрів технічних засобів:

Система повинна працювати на ІОМ сумісних персональних комп'ютерах. Для успішного функціонування системи необхідно, щоб параметри комп'ютера задовольняли наступним:

— комп'ютер на базі процесора Intel або AMD з частотою> 1,3 ГГ ц;
— оперативна пам'ять не менше 512Мб;
— вільна пам'ять на жорсткому диску не менше 1Gb;
— відео пам’ять не менше 16Мб;
— монітор;
— клавіатура;
— миша.

Вимоги до інформаційної та програмної сумісності:

— для роботи необхідний персональний комп'ютер, який використовує архітектуру Intel або AMD;
— операційну систему сімейства Windows.

1.2 Огляд і обґрунтування вибору інструментальних засобів

1.2.1 Вибір інструментів для розробки генератора ландшафтів

Для реалізації програмного забезпечення спочатку необхідно визначитися з використовуваною мовою програмування. При виборі мови програмування варто врахувати, що прогама розробляється для ПК, а абсолютна більшість ПК на даний момент працюють під управлінням ОС сімейства MS Windows. До найбільш популярних обьектно-орієнтованих мов програмування відносяться наступні мови: Java, C, C#, C++. Нижче наведено короткий опис можливостей цих мов.

Java — об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems (в подальшому придбана компанією Oracle). Програми Java зазвичай транслюються в спеціальний байт-код, тому вони можуть працювати на будь-якій віртуальній Java-машині незалежно від комп'ютерної архітектури. Іншою важливою особливістю технології Java є гнучка система безпеки завдяки тому, що виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером) викликають негайне переривання.

C — мова програмування, розроблена у 1969-1973 роках співробітниками Bell Labs Кеном Томпсоном і Деннісом Рітчі як розвиток мови Бі. Спочатку була розроблена для реалізації операційної системи UNIX, але, згодом, була перенесена на безліч інших платформ. Завдяки близькості по швидкості виконання програм, написаних на Сі, до мови асемблера, ця мова отримала широке застосування при створенні системного програмного забезпечення, прикладного програмного забезпечення й для вирішення широкого кола завдань. Мова програмування Сі справила значний вплив на розвиток індустрії програмного забезпечення, а синтаксис став основою для таких мов програмування, як C++, C#, Java і D.

C++ — компільована статично типізована мова програмування загального призначення. C++ поєднує властивості як високорівневих, так і низькорівневих мов. Синтаксис C++ успадкований від мови C. Одним з принципів розробки було збереження сумісності з C. Тим не менш, C++ не є в строгому сенсі надмножеством C; безліч програм, які можуть однаково успішно транслюватися як компіляторами C, так і компіляторами C++, досить велика, але не охоплює всі можливі програми на C.

C# — об'єктно-орієнтована мова програмування. Розроблена в 1998-2001 роках групою інженерів під керівництвом Андерса Хейлсберга в

компанії Microsoft. C# відноситься до сім'ї мов з C-подібним синтаксисом, з них синтаксис найбільш близький до C++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (у тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі у форматі XML.

В якості мови програмування для реалізації даного проекту була обрана обьектно-орієнтована мова C#. Дана мова відповідає концепції ООП і має всі необхідні конструкції та бібліотеки для реалізації поставленої задачі.

Далі необхідно вибрати середовище розробки. Цей етап дуже важливий, оскільки від нього залежить як швидкість, так і в якійсь мірі - якість розробки. На даний момент існує безліч засобів розробки які підтримують C#, одні з них є безкоштовними - інші платними. Деякі з них підтримують системи контролю версій, інші не підтримують. Не всі з них мають досить документації, деякі не підтримують сучасний стандарт C#.

Був проведений порівняльний аналіз декількох середовищ розробки. Всі засоби які беруть участь в порівнянні - безкоштовні (за певних умов). Всі з них досить популярні і мають своє співтовариство і документацію, що дуже важливо.

Порівняльні характеристики засобів програмування Visual Studio.NET, MonoDevelop і SharpDevelop представлені в таблиці 1.2.1.1.

Таблиця 1.2.1.1 – Порівняльні характеристики засобів програмування

Параметри	Visual Studio.NET	MonoDevelop	SharpDevelop
Назва, версія, фірма, виробник, ОС	VS Community 2015 Microsoft Windows	MonoDevelop 5.1 MonoDevelop Team Linux, Mac, Windows	SharpDevelop 5.1 ICSharpCode Team Windows
Ліцензія	MS EULA; безкоштовна з умовами	GNU GPL; вільне ПО	GNU LGPL; вільне ПО

Підтримка стандарту C# 5.0	+	+	+
Підсвічування синтаксису	+	+	+
Візуальний редактор форм	+	GTK#	Візуальний редактор для WPF і форм Windows Forms (COM-компоненти не підтримуються)
Інтегрований відладчик	+	+	+
Інтегрована підтримка СКВ	+	+	Інтегрована підтримка SVN, Mercurial і Git.
Підтримка unit-тестів	+	+	+
Простота / складність роботи	просто	просто	просто
Наявність документації	багата кількість документації	багата кільість документації	не багата кількість документації

Перераховані вище гідності, вплинули на вибір засоба програмування. Для реалізації генератора віртуальних ландшафтів була обрана система програмування VS C# Community як компонент програмного засобу Microsoft Visual Studio Community 2015.

1.2.2 Вибір інструментів для розробки візуалізатора ландшафтів

Для реалізації візуалізатора ландшафтів, так само як і для реалізації генератора, була обрана мова C#. Це пов'язано в першу чергу зі складністю переходу від однієї мови програмування до іншої під час розробки

програмних засобів, а також із забезпеченням сумісності між генератором і візуалізатором.

Для мови C# існує не багата кількість засобів візуалізації тривимірних об'єктів. Абсолютна більшість API бібліотек для розробки тривимірних додатків написано під C++ або C. Серед API або засобів для розробки тривимірних додатків на мові C# можна відзначити Managed DirectX (стара назва - DirectX .NET) і Unity (рушій гри) які розглянуті нижче.

DirectX — це набір API, розроблених для вирішення завдань, пов'язаних з програмуванням під Microsoft Windows. Найбільш широко використовується при написанні комп'ютерних ігор. Пакет засобів розробки DirectX під Microsoft Windows безкоштовно доступний на сайті Microsoft. Найчастіше оновлені версії DirectX поставляються разом з іграми.

Managed DirectX (MDX) є застарілим API від Microsoft для програмування DirectX на платформі .NET Framework. MDX може бути використаний з будь-якою мовою на .NET Framework (через CLR). MDX може бути використаний для розробки мультимедіа та інтерактивних програм (наприклад, ігри, скомпільовані тільки x86), забезпечує високу продуктивність і графічне представлення дозволяє програмісту використовувати сучасне графічне апаратне забезпечення під час роботи в середовищі .NET.

Managed DirectX був вперше випущений у 2002 році, щоб дозволити менш складний доступ до DirectX API через .NET. Managed DirectX SDK дозволяє розробникам отримати доступ до численних класів, які дозволяють рендеринг 3D-графіки (Direct3D) та інших DirectX API, в набагато простіший, об'єктно-орієнтованої манері. MDX, однак, не підтримує нові API-інтерфейси, такі як Direct3D 10, Xinput і XAudio 2.

Unity — це інструмент для розробки двох - і тривимірних додатків і ігор, працює під операційними системами Windows і OS X. Створені за допомогою Unity додатки працюють під операційними системами Windows,

OS X, Windows Phone, Android, Apple iOS, Linux, а також на ігрових приставках Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One. Є можливість створювати додатки для запуску веб-переглядачом за допомогою спеціального модуля Unity (Unity Web Player), а також за допомогою реалізації технології WebGL. Раніше була експериментальна підтримка реалізації проектів в рамках модуля Adobe Flash Player, але пізніше команда розробників Unity прийняла рішення щодо відмови від цього.

Додатки, створені за допомогою Unity, підтримують DirectX і OpenGL. Активно движок використовується як великими розробниками (Blizzard, EA, QuartSoft, Ubisoft), так і розробниками Indie-ігор (наприклад, Kerbal Space Program, Slender: The Eight Pages, Slender: The Arrival, Surgeon Simulator 2013, Baeklyse Apps: Guess the actor тощо) в силу наявності безкоштовної версії, зручного інтерфейсу і простоти роботи з движком.

Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Рушій підтримує дві сценарні мови: C# і модифікація JavaScript. Проект в Unity ділиться на сцени (рівні) - окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як, об'єкти (моделі), так і порожні ігрові об'єкти – тобто ті які не мають моделі. Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти. Також у них є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Так, у будь-якого предмета на сцені обов'язково присутній компонент Transform - він зберігає в собі координати місця розташування, повороту і розмірів по всіх трьох осях. У об'єктів з видимої геометрією також за замовчуванням присутній компонент Mesh Renderer, що робить їх модель їх видимою.

Також Unity підтримує фізику твердих тіл і тканини, фізику типу Ragdoll (ганчіркова лялька). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

При імпорті текстури в рушій можна згенерувати alpha-канал, mip-рівні, normal-map, light-map, карту відображень, проте безпосередньо на модель текстуру прикріпити не можна - буде створено матеріал, з яким буде призначений шейдер, і потім матеріал прикріпиться до моделі. Редактор Unity підтримує написання і редагування шейдерів. Крім того він містить компонент для створення анімації, анімацію також можна створити попередньо в 3D-редакторі та імпортувати разом з моделлю, а потім розбити на файли.

В якості засобу розробки візуалізатора був обраний Unity. Вибір був зумовлений мовою розробки, а також очевидними перевагами перед нечисленними конкурентами: система активно розвивається, зручний редактор, мультиплатформеність вихідних програм, велика кількість документації і велика спільнота.

1.3 Висновки до розділу 1

У даному розділі розглянуто предметну область дослідження. Виділено основні сутності предметної області та зв’язки між ними, також був проведений аналіз даних предметної області. Після чого була описана постановка задачі на випускну кваліфікаційну роботу у вигляді технічного завдання.

Були розглянуті питання, що стосуються мови розробки генератора і візуалізатора. Порівняльний огляд засобів розробки генератора віртуальних ландшафтів, а також обґрунтований вибір засобу розробки генератора. Вибір був зупинений на мові C# і засобі розробки Microsoft Visual Studio 2015.

Також були розглянуті засоби створення тривимірних додатків, для створення візуалізатора ландшафтів. Вибір засобу створення візуалізатора був зупинений на ігровому рушію Unity.

2. ВІЗУАЛІЗАЦІЯ ЗОБРАЖЕНЬ ЛАНДШАФТУ

2.1 Тесселяція поверхонь

Будуються тривимірні сцени з об'єктів, таких як зореліт, замок, шосе. Всі об'єкти рукотворні, на відміну від того, що знімає кінокамера. Важливо також, що всі об'єкти, навіть представляють собою об'ємні тіла, представляються тільки поверхнями, їх обмежуючими (і які можуть мати різну ступінь прозорості). Наприклад, акваріум представляється дном і гранями. Кожен об'єкт складається, таким чином, з набору поверхонь.

Точне завдання поверхні компактно, але є технічно непридатним для побудови сцен. Тому в 3D-графіці застосовують наближене уявлення гладкої поверхні безліччю дрібних плоских полігональних (тобто багатокутних) плиток, обробка яких елементарна. Така апроксимація (наближення) поверхні многогранником називається тесселяцією (tesselation - мозаїка) цієї поверхні. В якості плиток зазвичай використовуються трикутники.

Чим більше плиток в тесселяції, тим реалістичніше виглядає об'єкт і непомітніши шорсткості на стиках плиток. Загальне число плиток в сценах сучасних ігор обчислюється десятками тисяч. Збільшення кількості плиток є основним шляхом досягнення фотореалистической графіки, до чого впритул наблизилася споживча 3D-графіка. Разом з тим зростає і обсяг обчислень, вимагаючи більш продуктивних відеоплат.

Тесселяція більшості об'єктів провадиться лише один раз, до запуску сцени. Але деякі об'єкти тесселируются динамічно, при створенні кадру. Тесселяція вимагає плаваючої арифметики і виконується процесором (під управлінням програми). Об'єкти можуть задаватися і вже у протесселированом вигляді.

Тесселировані об'єкти задаються в деякій канонічній системі координат (далі СК), де їх координати постійні. Переміщення об'єкта по сцені проводиться його відображенням канонічної СК сцени.

У побудові кожного кадру беруть участь процесор і 3D-акселератор. При цьому послідовно проводять наступні кроки:

• розрахунок сцени;

• обробку полігонів;

• візуалізація.

Далі їдуть описання ціх кроків і супутних їм операцій.

2.2 Розрахунок сцени

При розрахунку кожної сцени (geometry setup) проводяться наступні дії:

1. Розміщення об'єктів (transformation). Кожен об'єкт, що потрапляє в кадр, відображається зі своєї канонічної СК сцени. Зауважимо, що СК сцени не статична, а пов'язана з гравцем і переміщується разом з ним.

2. Розрахунок освітлення (lighting). Проводиться для кожної плитки, виходячи з розташування ї інтенсивності джерел світла, а також тіней, похилої іншими плитками. Зазвичай розрахунок провадиться тільки для вершин полігону, а освітленість всієї плитки знаходиться інтерполяцією.

3. Вирізання (clipping) полігонів, що виходять за кадр.

4. Прив'язка текстур до полігонів - задається, яка текстура буде накладена на кожен полігон та відносне розташування полігону щодо текстури. Потрібно тільки для динамічної заміни текстури. Для статичних об'єктів на кшталт монстра, які задані в тесселированому вигляді, прив'язка вже виконана.

Створення сцени з реалістичним освітленням - одна з найбільших проблем тривимірної графіки. В реальності падаючий промінь світла зазнає величезну кількість відбиття та заломлення, тому дуже рідко можна зустріти різкі, не размиті тіні. Інша справа - комп'ютерна графіка. Тут число падінь і відбиття променя визначається лише апаратними можливостями комп'ютера. До певного моменту в тривимірній графіці переважали різкі тіні. Сцена, з

якої працює 3D-дизайнер, є лише спрощеною фізичною моделлю, тому отрендеренное зображення далеко не завжди схоже на справжнє. Але, незважаючи на це, освітлення в тривимірній сцені все ж можна наблизити до реального. Для цього потрібно виконати два правила: 1) встановити джерела світла і підібрати їх яскравість (параметри) таким чином, щоб сцена була рівномірно освітлена і 2) визначитися з налаштуваннями візуалізації освітлення. І те, і інше вимагає вміння і певної підготовки, в тому числі і теоретичної. У цій роботі ми не будемо розглядати конкретні приклади, а зупинимося на основних прийомах розстановки світла, які використовуються в тривимірній графіці. Проблема освітлення виникла задовго до появи тривимірної графіки. Перші, хто задавався цим питанням, були художники і фотографи, пізніше з цією проблемою стикалися кінооператори, тепер вона стала нагальною і для 3D-аніматорів. Найпоширенішою системою освітлення є система освітлення з трьох точок (або триточкова система). Незважаючи на те, що цей спосіб відмінно підходить для освітлення одного об'єкта (згадайте портрети у фотостудії), для складних тривимірних сцен він може не підійти. Вибір освітлення залежить від кількості об'єктів, відбивних властивостей їх матеріалів, а також від геометрії сцени. Для освітлення також є важливим, який тип джерела світла використовується. Так, наприклад, спрямовані джерела світла дозволяють сконцентрувати увагу на якомусь певному об'єкті, в той час як всеспрямоване точкове джерело - освітити сцену цілком.

Існує безліч прийомів, за допомогою яких можна так освітити сцену, щоб приховати дрібні недоліки і підкреслити важливі деталі. Так, наприклад, для того щоб додати об'єм тривимірної моделі, її досить освітити ззаду. При цьому з'явиться виразна межа, яка візуально відокремлює об'єкт від фону. Інший приклад: якщо за сюжетом у сцені потрібно освітити об'єкт наполовину, друга його половина повинна бути також підсвічується джерелом світла з малою інтенсивністю. Інакше затінена ділянка тривимірної моделі буде неприродно прихована в абсолютній темряві. Особливо це буде

помітно, якщо цей об'єкт розташований темною стороною до стіни. У цьому випадку світло повинно відбитися від стіни і слабо підкреслити контур затіненого боку об'єкта (так відбувається в реальності). Поряд з такими прийомами існують і загальні рекомендації, що стосуються того, як не потрібно висвітлювати сцену. Скажімо, джерело світла не повинно знаходитись набагато нижче освітлюваного об'єкта, оскільки це додасть моделі неприродний вигляд. Насправді найчастіше ми бачимо об'єкти, освітлені люстрою або сонцем, відповідно, і в тривимірних сценах джерело світла повинно розташовуватися зверху.

Текстура (map, texture - тканина, малюнок) - квадратні малюнки, що грають роль шпалер. Накладаються на полігони для надання поверхням реалістичності. Ви певно знаєте, що елемент зображення на екрані називається піксел (Picture Element). Текстура теж складається з елементів, так як це - растрове зображення, елемент текстури називається текселом (за аналогією з пікселом).

Розрахунок сцени іноді скорочено називають TL по імені основних кроків (Transformation and Lighting). Зауважимо, що сцена розраховується повністю, перш ніж буде передана для подальшої обробки.

Вся робота з розрахунку сцени проводиться в арифметики з плаваючою точкою, а сама сцена знаходиться в основній пам'яті.

Раніше розрахунок сцени виконувався центральним процесором. Однак, в данний момент розрахунок виконує графічний співпроцесор (GPU).

2.3 Обробка полігонів

Вчення про полігональні сітки — це великий підрозділ комп'ютерної графіки і геометричного моделювання. Безліч операцій, що проводяться над сітками, може включати булеву алгебру, згладжування, спрощення та багато інших. Різні уявлення полігональних сіток використовуються для різних цілей і додатків. Для передачі полігональних сіток по мережі

використовуються мережеві подання, такі як «потокові» і «прогресивні» сітки. Об'ємні сітки відрізняються від полігональних тим, що вони представляють і поверхня і об'єм структури, тоді як полігональні сітки явно представляють лише поверхню, а не обсяг. Так як полігональні сітки широко використовуються в комп'ютерній графіці, для них розроблено алгоритми трасування променів, виявлення зіткнень і динаміки твердих тел.

Об'єкти, створені за допомогою полігональних сіток повинні зберігати різні типи елементів, такі як вершини, ребра, грані, полігони поверхні. У багатьох випадках зберігаються лише вершини, ребра та грані, або полігони. Рендерер може підтримувати лише трьох-сторонні межі, так що полігони повинні бути побудовані з їх безлічі. Однак багато рендерів підтримують полігони з чотирма і більше сторонами, або вміють триангулировать полігони трикутники на льоту, роблячи необов'язковим зберігання сітки в триангулированной формі. Також в деяких випадках, таких як моделювання голови, бажано вміти створювати і трьох - і чотирьох-сторонні полігони.

Вершина — це позиція разом з іншою інформацією, такої як колір, нормальний вектор і координати текстури. Ребро — це з'єднання між двома вершинами. Грань — це замкнутий безліч ребер, в якому трикутна грань має три ребра, а чотирикутна - чотири. Полігон — це безліч граней. У системах, які підтримують багатосторонні межі, полігони і межі рівнозначні. Однак, більшість апаратного забезпечення для візуалізації підтримує лише межі з трьома або чотирма сторонам, так що полігони представлені як безліч граней. Математично, полігональна сітка може бути представлена у вигляді неструктурованої сітки, або неорієнтованого графа, з додаванням властивостей геометрії, форми і топології.

Поверхні, частіше звані групами згладжування, корисні, але не обов'язкові для групування гладких областей. Уявіть собі циліндр з кришками, такий як бляшанка. Для гладкого затінення сторін, всі нормалі повинні вказувати горизонтально від центру, тоді як нормалі кришок повинні

вказувати в +/-(0,0,1) напрямках. Якщо рендери як єдину, затінену по Фонгу поверхню, вершини складок мали б неправильні нормалі. Тому, потрібний спосіб визначення де припиняти згладжування для того, щоб групувати гладкі частини сітки, також, як полігони групують тристоронні межі. Як альтернатива наданню поверхонь/груп згладжування, сітка може містити іншу інформацію для розрахунку тих же даних, таку як розділяючий кут (полігони з нормалями вище цієї межі або автоматично розглядаються як окремі групи згладжування, або по відношенню до ребра між ними застосовується якась техніка, як наприклад поділ або скошування). Також, полігональні сітки з дуже високим дозволом менш схильні до проблем, для вирішення яких потрібні групи згладжування, так як їх полігони настільки малі, що потреба в них зникає. Крім того, інша альтернатива існує в можливості просто від'єднання самих поверхонь від решти сітки. Рендерер не намагається згладжувати ребра між несуміжними полігонами.

Обробкою полігонів (Polygon setup, іноді Triangle setup) займається виділений блок Polygon setup engine (рушій обробки полігонів) 3D акселератора графічного чіпсета. На блок з основної пам'яті по шині AGP надсилаються полігон за полігоном. Блок обробляє полігон і відразу відсилає його на блок візуалізації, працюючи як конвеєр.

Продуктивність блоку обробки полігонів називається Triangle throughput і являє собою пікову швидкість обробки трикутників в секунду. Реальна швидкість обробки трикутників може бути нижче, якщо, наприклад, процесор не встигає розраховувати і передавати необхідну відеоплати інформацію. Тобто, повільний процесор уповільнює роботу 3D-прискорювача.

Швидкість обробки сучасних чіпсетів становить 15 млн. трикутників у секунду і більше.

Обробка кожного трикутника включає:

1. Прийом координат вершин трикутника та їх атрибутів: освітленості, координат у текстурі (це називається текстурними координатами) та ін.

2. Корекцію перспективи трикутника. Тобто фігура масштабується залежно від її віддаленості.

3. Проектування трикутника на площину XY екрану

4. Обчислення пікселів, що входять в проекцію трикутника

5. Розрахунок для цих пікселів атрибутів. Розрахунок проводиться інтерполяцією значень, що задаються зазвичай на вершинах трикутника.

7. Посилку обробленого трикутника наступному блоку (рендеринг).

2.4 Рендеринг

Рендеринг (render - представляти) - перетворення розрахованої сцени (після фази обробки полігонів) у вигляді пікселів в буфер кадрів (для подальшого виведення на екран).

Прикладом візуалізації можуть служити радарні космічні знімки, які представляють на виході зображення даних, отриманих за допомогою радіолокаційного сканування поверхні космічного тіла, в діапазоні електромагнітних хвиль, невидимих оком.

Часто в комп'ютерній графіці (художньої та технічної) під рендерингом (3D-візуалізації) розуміють створення плоскої картинки — цифрового растрового зображення — за розробленою 3D-сцени. Синонімом у цьому контексті є візуалізація.

Візуалізація — один з найбільш важливих розділів в комп'ютерній графіці, і на практиці він тісно пов'язаний з іншими. Зазвичай програмні пакети тривимірного моделювання та анімації включають в себе також і функцію рендеринга. Існують окремі програмні продукти, що виконують рендеринг.

Залежно від мети, розрізняють пре-рендеринг, як досить повільний процес візуалізації, що застосовується в основному при створенні відео, і

рендеринг у режимі реального часу, наприклад, в комп'ютерних іграх. Останній часто використовує 3D-прискорювачі.

На даний момент розроблено безліч алгоритмів візуалізації. Існуюче програмне забезпечення може використовувати декілька алгоритмів для отримання кінцевого зображення.

Трасування кожного променя світла в сцені непрактично і займає неприйнятно довгий час. Навіть трасування малої кількості променів, достатнього, щоб отримати зображення, займає дуже багато часу, якщо не застосовується апроксимація (семплування).

Комп'ютерна програма, яка виробляє рендеринг, називається рендером (англ. render) або рендерером (англ. renderer).

Рендеринг виконується виділеним блоком Fill engine (двигун заповнення) 3D акселератора графічного чіпа, який обробляє "на льоту" потік трикутників від попереднього блоку.

Проводиться попіксельне заповнення трикутника. Для кожного пікселя спочатку обчислюється, чи є він видимим (зазвичай за допомогою z-буфера). Якщо піксель невидимий, то виводити його в буфер, очевидно, не потрібно.

Далі здійснюються такі дії:

1. Обчислюється накладається тексел (іноді вельми витончено).

2. Проводиться накладення тексела з урахуванням освітленості пікселя.

3. Опціонально на піксел накладаються ефекти, що збільшують реалістичність (туман, карта освітленості, карта рельєфу, тощо).

4. Проводиться запис пікселя в буфер кадрів.

Після заповнення буфера кадру весь кадр може бути опціонально підданий постобробці. Прикладом постобробки є згладжування, що покращує зображення, або накладення додаткового зображення на частину кадру для створення ефекту дзеркального відображення у частині кадру.

Пікову продуктивність блоку заповнення пікселів називають Fill rate (швидкість заповнення, промальовування, растеризації), і вимірюється вона в

кількості пікселів в секунду. Реальна швидкість промальовування багато менше, і тим менше, чим більше текселов накладається на один піксель, тому розраховується для найбільш простого режиму текстурування.

Зауважимо, що в 3D-трафіку, обов'язкові мінімум 2 буфера кадрів (БК), на відміну від 2D-графіки (це називаєте подвійною буферизацією, double buffering). Справа в тому, що зазвичай в процесі заповнення кожен піксель багаторазово переписується (наприклад, при постобробці) поки не вийде остаточний вигляд кадру. Тому па екран виводиться закінчений буфер кадрів, а в другому готується наступний кадр.

Для більшої швидкості графіки (гладкого відтворення) застосовуються також 3 і більше кадрових буфера, що є характеристикою чіпсету.

2.5 Текстурування

Існуючою тенденцією є збільшення розміру використовуваних текстур: чим більше текстура, тим більш реалістична поверхонь. Максимальний розмір текстури є параметром 3D-чіпа. Зараз цей параметр становить 1024x1024 і більше. Такі великі текстури використовують сучасні ігри. Всі сучасні 3D-чіпи підтримують можливість роботи з великими текстурами.

Поняття фільтрації текстур. Як буде ясно надалі, накладення лише одного тексела (найближчого) на піксел призводить до занадто грубим і навіть неприродним результатами. Тому накладають тексел, що є результатом інтерполяції декількох текселов. У 3D-графіці замість терміна інтерполяція часто застосовують термін фільтрація. Він стає зрозумілим, якщо вважати, що фільтруються спотворення зображення.

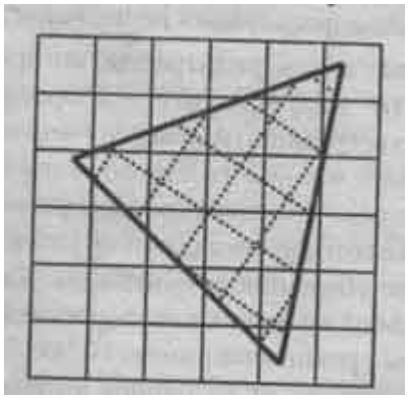

Рисунок 2.5.1 – Трикутник в системі координат текстури

При текстурировании типовою є ситуація, коли текселі і пікселі не збігаються. На рисунку 2.5.1 зображений трикутник в системі координат текстури. Для простоти пікселі і текселы мають форму квадрата. З малюнка видно, що кожний піксель накривається не одним, а трьома, а частіше чотирма текселами. Якщо обчислювати колір по одному текселу, то це призводить до так званої блочності - великим блокам однаково зафарбованих пікселів, яких не повинно бути.

Існують різні способи інтерполяції, що відрізняються числом залучених до неї текселов. Чим більше текселов, тим краще зображення, але і більше трудомісткість. Найпростіший з таких методів - білінійна інтерполяція (Bi-Linear Filtering), при якій колір розраховується як середнє чотирьох сусідніх текселов (обчислюється попарно операціями, що і дало назву).

В даний час застосовують досконаліші методи фільтрації.

Очевидно, що якщо дивитися на текстуру поблизу, то повинні бути чітко видні всі деталі, наприклад прожилки на мармурі. Якщо збільшити відстань до текстури, то деталі повинні зникати, а малюнок стає розмитим. Таким чином, для реалістичності картинки потрібна корекція текстурної поверхні в залежності від наближення чи віддалення від неї камери. Очевидно, що одна лише білінійна фільтрація не дозволяє це зробити.

Проблему різних LOD однієї текстуры вирішує метод фільтрації, який називається мипмэппинг (MIP mapping). MIP - абревіатура з латини (що

нечасто зустрінеш) Multum in Parvum - багато в одному. Це означає, що замість одного примірника текстури зберігаються кілька примірників, що відрізняються здатністю. Екземплярам приписуються рівні (Level of Detail - LOD) 0,1, 2 і т. д. в порядку убування деталізації. Кожен рівень являє собою квадрат з вдвічі меншою стороною, ніж на попередньому рівні (наприклад, 1024x1024, 512x512,..., 32x32). В залежності від віддаленості від того об'єкта, на який натягується текстура, вибирається відповідний примірник (який потім трохи масштабується, щоб краще відповідати розміру полігону), далі застосовується білінійна фільтрація.

Все сучасні 3D-чіпи підтримують найбільш точний різновид мипмэппинга - попіксельний мипмэппинг, коли LOD розраховується для кожного пікселя полігону (більш грубим є пополигонный мипмэппинг, сенс якого, напевно, цілком ясний). Крім того, що існує зовсім грубий автомипмэппинг, де використовується одна детальна текстура, а інші рівні формуються "на льоту" масштабуванням. Цей метод не підтримують всі чіпи, однак, у силу морального старіння цей параметр неістотний.

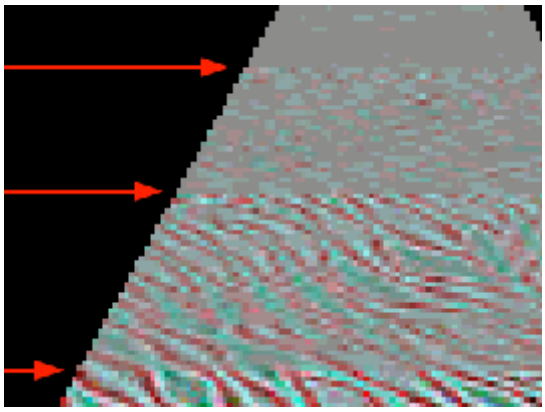

Рисунок 2.5.2 – Мипмэппинг на що йде вглиб сцени полігоні

Мипмэппинг добре працює, коли полігон має малу z-протяжність (майже паралельний площині спостереження). Однак, наприклад, полігони що йдуть углиб стіни мають велику z-протяжність. При використанні мипмэппинга ближні ділянки полігону будуть більш детально текстуровані, далекі - менш детально. На стику цих ділянок виникне явна смугастість (mip-banding, мипбэндинг) (див. рис. 2.5.2), якої не повинно бути.

Той же результат - трілінейная фільтрація. Тут так само, як і в мипмэппинге, використовується LOD-набір текстур, але якщо в мипмэппинге використовується текстура лише одного рівня деталізації, то тут - двох рівнів, які беруть в "вилку" цей піксель. На піксел послідовно накладаються відповідні текселы цих двох рівнів. Всі сучасні 3D-чіпи підтримують трилинейную фільтрацію. Однак не всі чіпи вміють виконувати таку фільтрацію за такт, що і є параметром швидкості.

У близьких об'єктах в один піксел може потрапляти кілька десятків текселов. Тепер згадаємо, що всі наведені вище методи засновані на білінійної фільтрації, де враховуються всього 4 тексела. Проблему вирішує анізотропна фільтрація (anisotropic filtering), де враховуються 16-32 текселов. Термін анізотропія (неоднорідність за напрямами), вказує на те, що число враховуються текселов виявляється різним у залежності від нахилу полігону (у попередніх методах - фіксоване, тобто фільтрація изотропная). Прикладом переваги анізотропної фільтрації є плакат з написом, розташований під кутом до площини екрану. При анізотропної інтерполяції напис на плакаті буде розбірлива, а при ізотропному - розмита.

Таким чином, анізотропна інтерполяція - найбільш точна, але вимагає великої пропускної здатності при доступі до пам'яті (з текстурами). Анізотропна фільтрація підтримується не всіма чіпсетами.

Безпосереднє растеризування будь-яких безперервних кривих на екрані призводить до їх зазубренности (сходовий ефект). Причому прямі лінії ведуть себе нічим не краще. Все це відбивається при рендерінгу пікселів на кордоні полігонів. Такий клас дефектів зображення, викликаних грубої дискретністю, називається алиасингом (aliasing - нерівність, ступінчастість).

Для придушення аліасінга застосовуються різні види антиалиасинга - фільтрації, що полягає в точному обчисленні кольору пікселів (насамперед крайових).

Антиаліасінг буває:

* крайової (Edge Anti-Aliasing) - застосовується до пікселів на межі полігону;

* повний (Full Anti-Aliasing) - застосовується до всіх пікселів.

Як ні парадоксально це звучить, але крайової антиаліасінг важко реалізувати в повному обсязі (наприклад, неможливо заздалегідь знати, де перетнуться полігони). Далі, цей антиаліасінг має підтримувати сам додаток (і давати можливість вмикати/вимикати його). Тому сучасні 3D-чіпи застосовують повний антиаліасінг. Суть полягає в тому, щоб обчислювати кожен піксел з великою роздільною здатністю, а потім записувати його з заданим дозволом. Апаратна підтримка антиалиасинга є параметром якості чіпсету. Проте навіть сучасні чіпи, які мають можливість реалізувати повний антиаліасінг, витрачають на це стільки ресурсів, що реально використовувати цю можливість можна тільки у невисоких дозволах і тільки у найшвидших сучасних чіпів.

Всі сучасні чіпсети підтримують попіксельне накладення ефекту туману. Однак поряд з прямим використанням туману (fogging) для додання реалістичності віддалених об'єктів виявилося абсолютно необхідним використовувати його для заднього фону. Справа в тому, що розміри пікселя ставлять межа для зображення дрібних віддалених об'єктів під час рендеринга, тому обмежуються деякої граничної z-площиною, до якої розраховується сцена. Але тоді прилеглий до цієї межі шар треба заповнювати туманом, інакше об'єкт, що перетинає цю граничну задню площину, буде несподівано з'являтися або зникати, що нерідко можна бачити в ряді ігор, що активно користуються туманом на деяких старих акселераторах.

Найбільш популярним є метод табличного затуманення (Range based), коли щільність туману змінюється ступінчасто з відстанню, що задається таблицею.

Накладення рельєфу (Bump mapping, bump - опуклість) - додання поверхні рельєфності, тобто нерівності. Справа в тому, що більшість поверхонь в сценах не гладкі. Прикладами є шорсткі стіни печер і будівель, рельєф на металі, а також дрібна брижі на воді. Всі ці ефекти різко збільшують реалістичність. Накладення рельєфу проводиться на будь-яку поверхню і без збільшення кількості полігонів, за рахунок зміни освітленості, коли одна частина горбка рельєфу освітлена, а інша знаходиться в тіні.

Методи накладення рельєфу:

* Embossing Bump Mapping, вже підтриманий апаратно всіма чіпами метод видавлювання, який, однак, не дає справжнього просторового ефекту. Це роблять більш досконалі технології, перелічені нижче.

* Environment Mapped bump mapping (EMBM, використання карти середовища).

* Dot Produce bump mapping (DTBM, скалярний рельєфне текстурування, а також NormalMaps).

Для підвищення якості зображення практично на кожен оброблюваний піксел трикутника накладаються кілька текселов. Для підвищення швидкості рендеринга всі сучасні 3D-чіпи застосовують мультитекстурування (multi-texturing) - накладення на піксел більше одного тексела за такт. Синонімом є термін однопрохідного мультитекстурування. Параметром мультитекстурирования є число текселов, накладених на піксел.

Однією з тенденцій сучасної 3D-графіки є використання все більших обсягів текстур. Технології компресії (стиснення) текстур дозволяють зберігати текстури в стиснутому вигляді на жорсткому диску, відеопам'яті і при пересиланні. І тільки останній стадії, коли текстура надсилається на графічний чіп для накладання, вона їм декомпрессируется, тим самим ці технології дозволяють:

* використовувати більше текстур;

* знижувати навантаження на внутрішню шини, зменшуючи затримки;

* зберігати текстури в компактному вигляді.

Для стиснення текстур використовуються спеціальні методи, придатні для роботи в реальному часі і мають гарантований коефіцієнт стиснення. Вони дещо погіршують оригінал щодо градації (відмінності) кольорів і точності контурів, причому для деяких текстур значно (тому розробник повинен сам вибрати ті текстури, які використовуються в стислому вигляді). Сьогодні стиснення текстур вже підтримується великою кількістю ігор, і в майбутньому використання стиснення представляється неминучим, якщо врахувати, що одна текстура 2048x2048x32 займає в початковому вигляді 16 Мбайт.

2.6 Висновки до розділу 2

У даному розділі були розглянуті особливості реалізації тривимірних сцен, етапи побудови тривимірних сцен і супутні операції.

Детально розглянуті такі етапи побудови тривимірних сцен, як: розрахунок сцени (включає в себе: розміщення об'єктів, розрахунок освітленості, відсікання полігонів, прив'язка текстур до полігонів), обробка полігонів, рендеринг сцени, текстурування (мипмеппинг, трілінейная фільтрація, анизатропная фільтрація, аліасінг).

3. ГЕНЕРАЦІЯ ЗОБРАЖЕНЬ ЛАНДШАФТУ

3.1 Алгоритми генерації псевдовипадкових чисел

Для генерації ландшафтів існує безліч різних алгоритмів. Однак їх незмінно об'єднує одне: всі вони так або інакше використовують генератори псевдовипадкових чисел (далі ГПВЧ) у своїй роботі. Більшість алгоритмів тісно пов'язані з ГПВЧ.

Від швидкості роботи ГПВЧ залежить і швидкість генерації ландшафту, однак, як правило, не значно (в більшості випадків, час генерації псевдовипадкових чисел займає менше 5% від роботи всього алгоритму). А ось якість згенерованих ландшафтів залежить від ГПВЧ досить сильно: чим більш непередбачувані результати дозволяє одержати ГПВЧ, тим більш реалістичними виходять ландшафти.

У багатьох мовах програмування є класи і методи ГПВЧ зі словом random у назві, так що вони будуть очевидною відправною точкою.

Генератор псевдовипадкових чисел видає послідовність випадкових значень в залежності від вхідного сіда. В об'єктно-орієнтованих мовах ГПВЧ – це зазвичай об'єкт, ініціалізований за допомогою сіда. Після цього метод об'єкта може повторно викликатися для генерації псевдовипадкових чисел. На рисунку 3.1 показаний принцип роботи ГПВЧ.

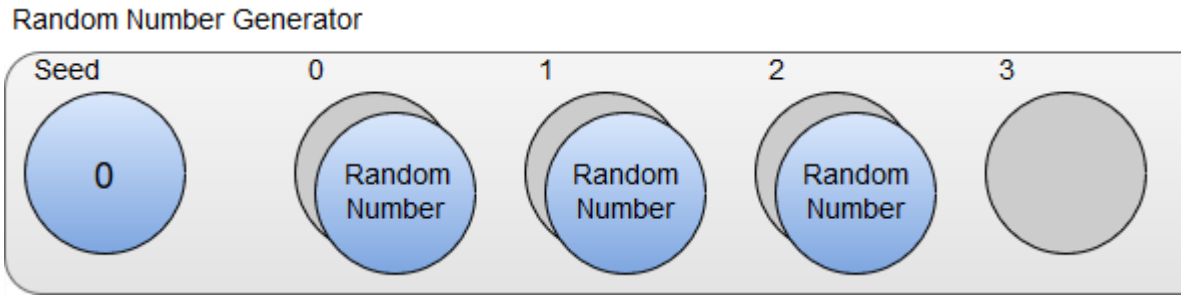

Рисунок 3.1 – Принцип роботи ГПВЧ

Сідом (seed) може бути число, рядок, або інші дані, що використовуються як вхідне значення для отримання випадкового результату на виході. Відмінною рисою сіда є те, що один і той самий сід завжди видає

однаковий результат, але навіть найменша його зміна може призвести до зовсім іншого результату.

Тут важливо розуміти, що як показано на рисунку 3.2, не можливо отримати третє випадкове число, поки не отримано перше і друге. Це не просто недогляд в реалізації, така сама суть ГПВЧ – він генерує кожне нове число, використовуючи попереднє у своїх обчисленнях. Тому ми і говоримо про випадкової послідовності.

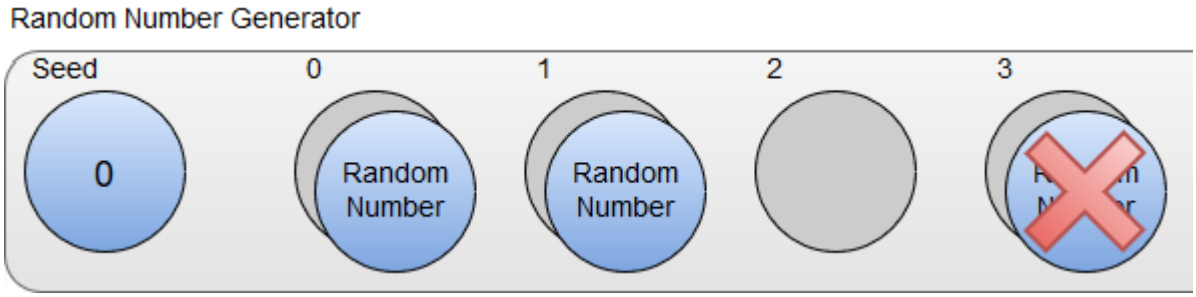

Рисунок 3.2 - Особливості роботи ГПВЧ

Це значить, що поки вам потрібно кілька розташованих по порядку випадкових чисел, то ГПВЧ відмінно з цим впораються, але якщо потрібно конкретне значення (скажімо 26-е за рахунком), то тут вам не пощастило. Звичайно, можна 26 разів викликати ГПВЧ і використовувати останнє число, але це погана ідея.

Якщо ви генеруєте все і відразу, то конкретні числа з послідовності вам, можливо, і не знадобляться. Інша справа, якщо щось генерується на льоту і по частинах.

Наприклад, є три секції ігрового світу: A, B і C. Гравець починає в секції A, вона генерується з допомогою 100 випадкових чисел. Потім гравець просувається в секцію B, яка генерується 100 інших випадкових чисел. В цей же час секція A знищується, звільняючи місце в пам'яті. Далі йде секція C, для створення якої використовуються інші 100 значень і також видаляється секція B.

Однак, якщо гравець захоче повернутися в секцію B, її потрібно генерувати з тих же значень, що і в перший раз, щоб вона виглядала незмінною.

Для усунення даного недоліку часто використовують ГПВЧ з різними значеннями сіда. Це досить поширена омана. Суть в тому, що як показано на рисунку 3.3, числа однієї послідовності випадкові по відношенню один до одного, але числа того ж порядку з різних послідовностей випадковими по відношенню один до одного не будуть, навіть якщо так і здасться на перший погляд.

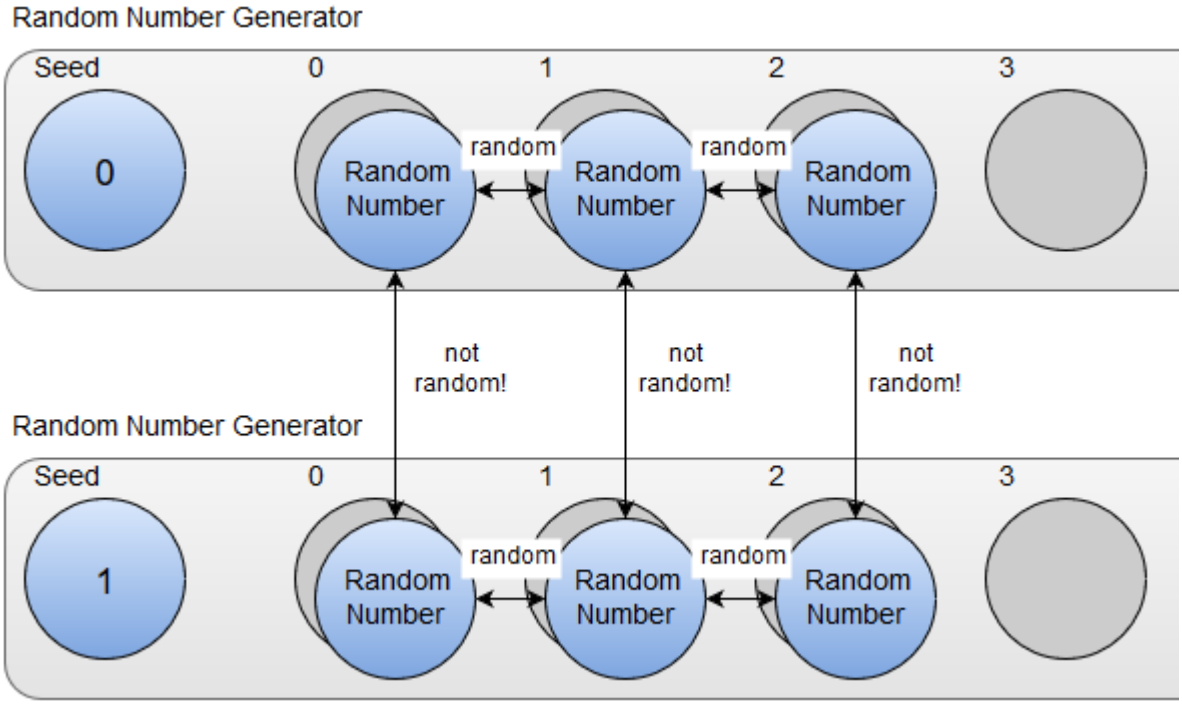

Рисунок 3.3 Взаємозв'язок псевдовипадкових послідовностей

Так що, якщо взяти перші числа зі 100 послідовностей, вони не будуть випадковими по відношенню один до одного. Той же результат буде спостерігати і з 10-ю, 100-ма або 10000-ми порядковими значеннями послідовностей.

На рисунку 3.4 зображений візерунок з перших чисел 65536 послідовностей, згенерованих з seed-ом від 0 до 65535.

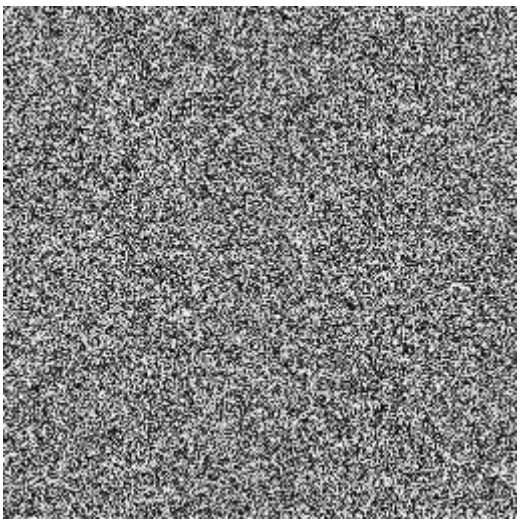

Рисунок 3.4 – Перші числа 65536 послідовностей

Візерунок досить рівномірно розподілений, але він не зовсім випадковий.

Можна укласти, що ГПВЧ знадобляться, тільки якщо вам не потрібен доступ до порядковим значенням послідовностей. В іншому випадку варто звернути увагу на випадкові хеш-функції.

Хешування - перетворення масиву вхідних даних довільної довжини в (вихідний) бітовий рядок фіксованої довжини, що виконується певним алгоритмом. Функція, що реалізує алгоритм і виконує перетворення, називається «хеш-функцією або функцією згортки». Вихідні дані називаються вхідним масивом, «ключем» або «повідомленням». Результат перетворення (вихідні дані) називається «хешем», «хеш-кодом» або «хеш-сумою».

В загальному випадку (згідно з принципом Діріхле) немає однозначної відповідності між початковими (вхідними) даними і хеш-кодом (вихідними даними). Значення, що повертиються хеш-функцією (вихідні дані) менш різноманітні, ніж значення вхідного масиву (вхідні дані). Випадок, при якому хеш-функція перетворює кілька різних повідомлень в однакові зведення називається «колізією». Імовірність виникнення колізій використовується для оцінки якості хеш-функцій.

Основне використання в процедурній генерації – надання одного або декількох цілих чисел у якості вхідних даних і отримання випадкового значення в якості результату. Наприклад, у великих ландшафтах, де за один раз генерується тільки мала їх частина, основна потреба – отримання випадкового числа, пов'язаного з вхідним вектором (таким, як локація), так щоб це число завжди залишалося незмінним при незмінному вхідному значенні. На відміну від ГПВЧ, як показано на рисунку 3.5, тут немає послідовностей – числа можна отримувати в будь-якому бажаному порядку.

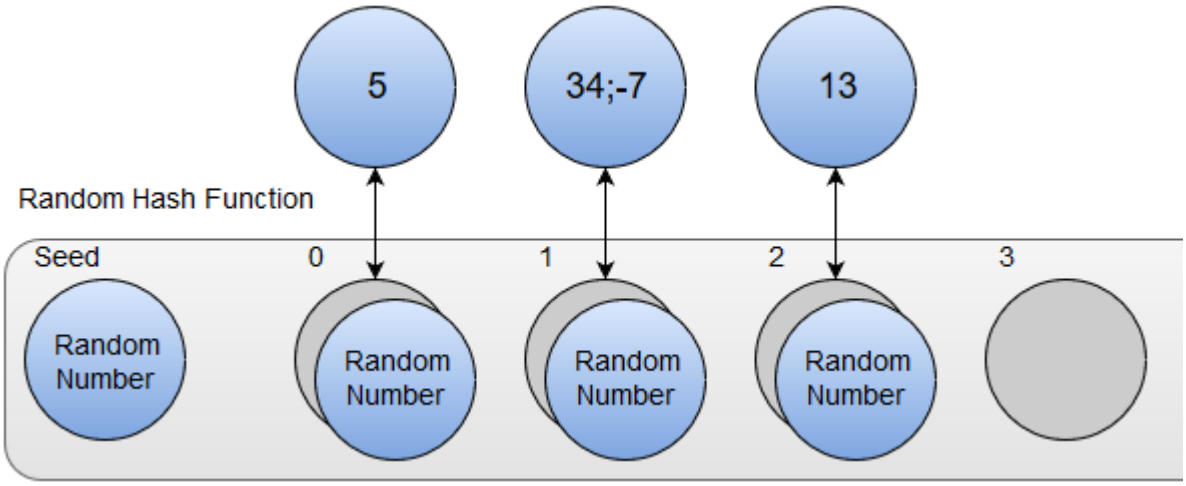

Рисунок 3.5 – Принцип роботи хеш-функції

Процедурна генерація – нетипове використання хеш-функцій, так що не всі вони будуть слушними, або видаючи поганий розподіл, або вимагаючи невиправдано багато ресурсів.

Одна з традиційних галузей застосувань хеш-функцій – реалізація структур даних, таких як словники. Найчастіше вони дуже швидкі, але зовсім позбавлені елемента випадковості, тому що потрібні головним чином для більш ефективного виконання алгоритмів.

Інша область їх застосування – криптографія. Такі хеш-функції показують хорошу випадковість, але при цьому дуже повільні, тому що вимоги до криптографічно стійким функцій набагато вище, ніж до значень, які лише виглядають випадковими.

Генератори випадкових чисел можна комбінувати з хеш-функціями. Розумний підхід полягає у використанні генератор випадкових чисел з

різними сидами, але так, щоб сіди проходили через випадкову хеш-функцію, а не застосовувалися безпосередньо.

Як показано на рисунку 3.6 - ідеальним підходом буде використання випадкової хеш-функції для генерації сидів локацій на основі їх координат і подальше використання сіда для генерації послідовності випадкових чисел.

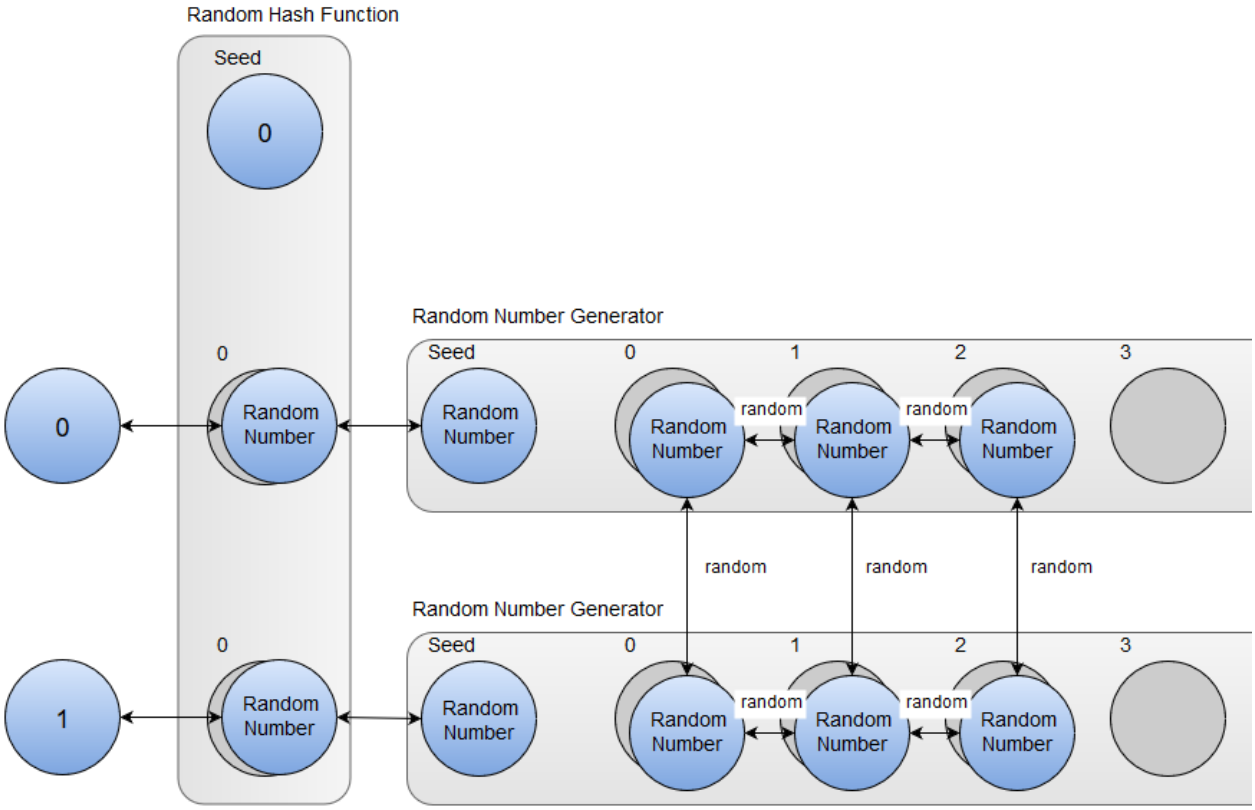

Рисунок 3.6 – Комбінування хеш-функції та ГПВЧ

Якщо необхідний контроль над порядком випадкових чисел використовується оптимізована для процедурної генерації версія відповідної випадкової хеш-функції (наприклад, xxHash).

Якщо порядок випадкових чисел не важливий, найпростішим способом їх отримання буде ГПВЧ, наприклад клас System.Random в C#. Якщо усі числа повинні бути випадковими по відношенню один до одного, вони повинні входити в одну послідовність (инициализированную одним сідом), або сиди різних послідовностей повинні спочатку пропускатися через випадкову хеш-функцію.

3.2 Алгоритми генерації ландшафтів і їх особливості

У даному підрозділі мова піде про принципи представлення даних для зберігання інформації о тривимірних ландшафтах, а також про алгоритми генерації тривимірних зображень ландшафтів та їх особливості.

Існує кілька основних принципів подання даних для зберігання інформації о ландшафтах: регулярна сітка висот, иррегулярна сітка вершин і зв'язків їх з'єднуючих і посегментна карта.

При використанні регулярної карти висот, дані представлені у вигляді двовимірного масиву. Як показано на рисунку 3.7 - вже задані дві координати (x, y — по висоті і ширині масиву), і третя координата задається значенням в конкретній комірці, це висота.

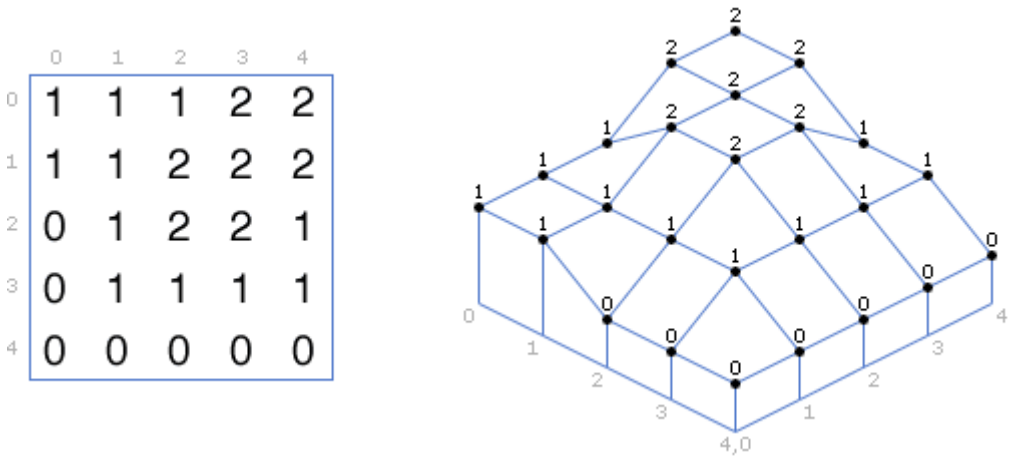

Рисунок 3.7 – Регулярна карта висот

Як правило, карту висот зберігають у файлах зображень. Це дозволяє легко вносити зміни і більш-менш наочно переглядати дані. Тоді двома координатами буде положення конкретного пікселя на картинці, а третя координата буде представлена кольором (чим вище значення, пряма залежність від яскравості пікселя — тим більше значення висоти для цієї точки). Зазвичай такі картинки містяться в монохромному варіанті, але можна використовувати і всі кольори радуги. Другий варіант дає нам більше градацій висоти, ніж передбачувані 256 градацій у разі монохромного подання.

На рисунку 3.8 зображений приклад карти висот (побудовано за допомогою програми ReLife3D).

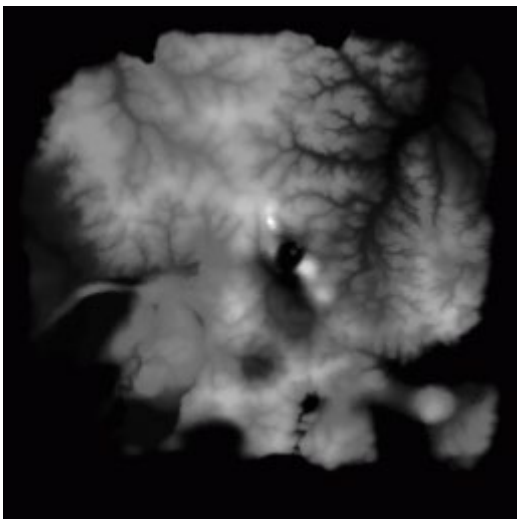

Рисунок 3.8 – Приклад карти висот

За допомогою цього способу можна уявити досить великі простори. Але у нього є один суттєвий недолік — занадто багато описів для точок, а також, в деяких випадках, спостерігається надмірність даних (наприклад, коли задається проста площину, то в цьому випадку для побудови простої площині буде використовуватися безліч точок, хоча можна було б обійтися трьома). Хоча і ця сама надмірність може піти нам на користь, наприклад, при освітленні.

У цього методу існує і кілька плюсів:

• наочність, в будь-якій програмі перегляду графічних файлів можна відразу побачити всю інформацію;

• простота зміни цих даних, так як існує безліч програм для роботи з растровою графікою;

• легкість знаходження координат та висоти на карті;

• ще один плюс — так як вершинні точки розташовані регулярно і досить близько, можна більш правильно і досить акуратно проводити динамічне освітлення (найчастіше, освітленість вершини безпосередньо залежить від відстані від цієї вершини до джерела освітлення). Це і є та сама користь від надмірності даних.

Ще один спосіб представлення даних ландшафтів — иррегулярная сітка вершин і зв'язків що їх з'єднують. Часто такі рішення застосовуються в

спеціалізованих пакетах для ігор (наприклад, редактор рівнів для Серйозного Сема) або спеціальних пакетах для роботи з тривимірною графікою (наприклад - 3Dmax, Maya та багато інших). І зберігаються у вигляді тривимірних моделей. Це дає основний виграш у порівнянні з картами висот - використовується значно менше інформації для побудови ландшафту. Нам необхідно зберігати тільки значення висот кожної вершини і зв'язки що з'єднують ці вершини. Це дає нам виграш у швидкості при передачі величезних масивів інформації з AGP, в процесі візуалізації ландшафту.

Але крім плюсів у цього способу є і безліч недоліків:

• алгоритми побудови ландшафтів в основному призначені для регулярних карт висот. Оптимізація таких алгоритмів під цей спосіб вимагає значних зусиль;

• складності при динамічному освітленні — вершини розташовані досить далеко один від одного і нерівномірно;

• зберігання, перегляд, модифікація такого ландшафту також представляє складності. При використанні карт висот можна користуватися досить простими і стандартними засобами піксельної графіки.

На рисунку 3.9 зображений принцип іррегулярні сітки висот і зв'язків.

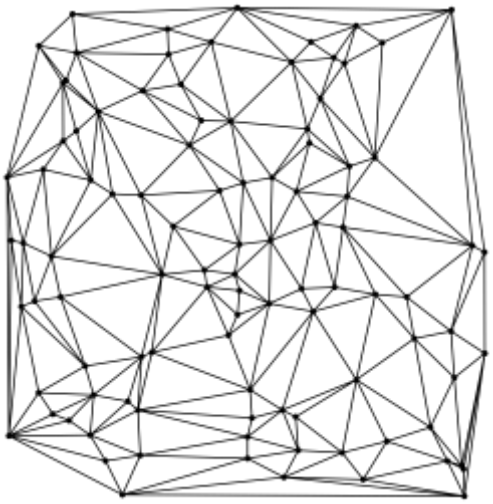

Рисунок 3.9 – Принцип іррегулярні сітки висот і зв'язків

Спосіб третій — посегментная карта шаблонів, у даному способі також використовуються регулярна або иррегулярна карта. Тільки замість висот у

ньому зберігаються індекси ландшафтних сегментів. Як ці сегменти представлені, як правило, не має значення. Вони можуть бути регулярними, і іррегулярнимі (можуть використовуватися і ті, і інші одночасно). Як правило цей варіант представлення даних ландшафту використовується в іграх і кінематографічних сценах. Окремо зберігається бібліотека шаблонів (моделей) таких як: посуд, меблі, машини і навіть будівлі, а посегментная карта зберігає лише їх взаєморозташування. Також, часто використовується комбінований метод - регулярна (або иррегулярна) карта висот зберігається поряд з посегментной картою шаблонів. Це дає нам наступні переваги:

• можливість подання величезних відкритих просторів;

• крім самих ландшафтів у таких блоках можна зберігати й інформацію о будівлях, рослинах, специфічних ландшафтних рішеннях (наприклад, печери або скелі, що нависають один над одним) – що може бути зручно для ігрових ландшафтів;

• можливість створення декількох варіантів одного і того ж сегменту, але за різною мірою деталізації. В залежності від швидкості або завантаженості комп'ютера можна вибирати більш або менш деталізовані варіанти (так звані LOD ландшафти — LOD — Level Of Detail).

Однак, у цього способу є і мінуси:

• проблема стикування різних сегментів;

• неочевидність даних. Поглянувши на карту блоків, неможливо уявити, як це має виглядати на виході;

• проблема модифікації. На даний момент не існує єдиного стандарту подання даних для посегментних карт, у зв'язку з чим при реалізації ландшафтів даним методом доводиться розробляти свій стандарт і редактор.

Мною був обраний перший варіант представлення даних о ландшафтах у зв'язку з його адаптованостю до алгоритмів генерації реалістичних тривимірних ландшафтів. Иррегулярная карта ж вимагає додаткових обчислень (часом досить об'ємних і витратних) не тільки при створенні і

модифікації, але й при перегляді ландшафту – що на мій погляд не допустимо. Економія місця якої дозволяє отримати іррегулярна карта висот для реалістичних ландшафтів у більшості випадків незначна і при необхідності можна отримати більшу економію застосовуючи зовнішні спеціалізовані або універсальні методи стиснення даних.

Для генерації ландшафту існує безліч алгоритмів. Однак не всі з них дають прийнятні результати, крім того, деякі занадто повільні (час генерації також має велике значення). Нижче розглянуті деякі з них: проста random-на генерація, Холмовий алгоритм та алгоритм Diamond-Square.

Алгоритм random-ої генерації дуже простий:

1. Усі клітинки карти висот заповнюються довільними псевдовипадковими числами.

2. Карта згладжується велику кількість разів (зазвичай понад 10 разів).

3. Проводиться нормалізація ландшафту.

Даний алгоритм дуже простий, і на перший погляд досить хороший. Він згадується у багатьох місцях, як найпростіший алгоритм генерації ландшафтів. Однак він дає неприйнятні результати і досить витратний.

Усі клітинки карти висот заповнюються довільними псевдовипадковими числами. Потім отриманий ландшафт згладжується певну кількість разів (число проходів згладжуючим фільтром ще називають ступенем згладжування) – цей етап і дозволяє отримати скільки-небудь прийнятний результат. На рисунку 3.10 зображені результати роботи даного методу.

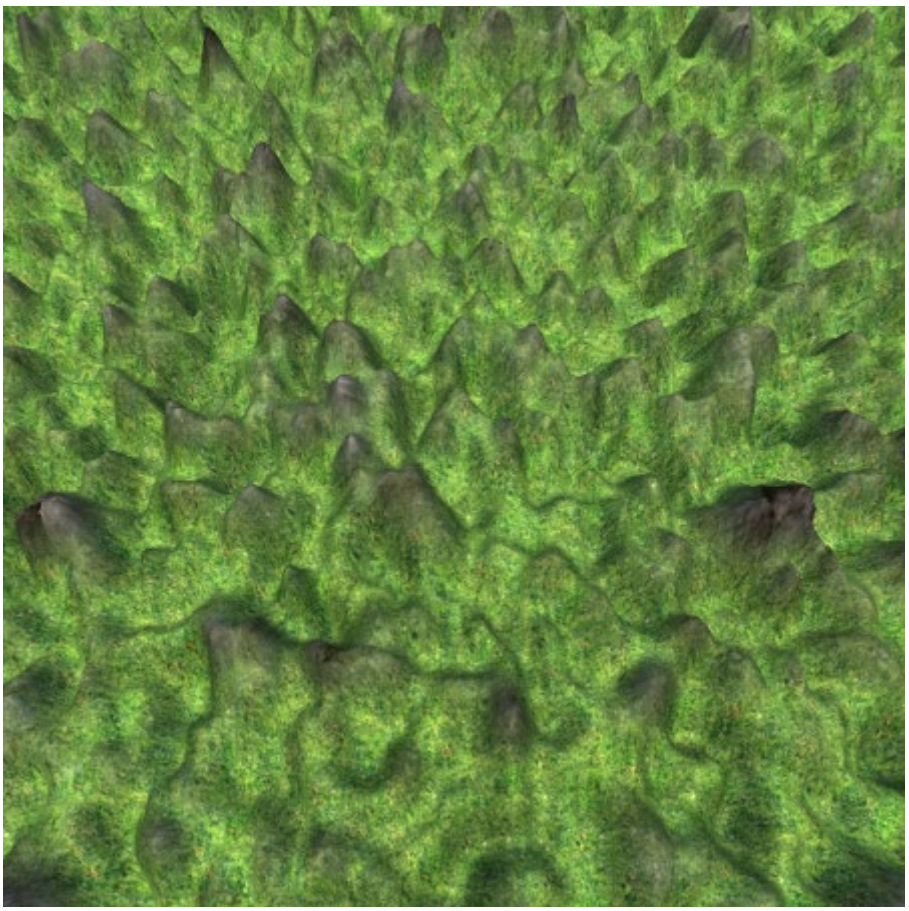

Рисунок 3.10 – Результати роботи random-ої генерації

Весь ландшафт необхідно кілька раз обробити згладжуючим фільтром, а потім результат нормалізувати. Як видно даний алгоритм дає мало прийнятні результати роботи, однак головним його недоліком є його повільна робота.

Наступним розглянутим алгоритмом генерації буде Холмовий алгоритм.

Холмовий алгоритм (Hill Algoritm) - це простий ітераційний алгоритм, заснований на декількох вхідних параметрах. Алгоритм викладено в наступних кроках:

1. Створюємо двовимірний масив та ініціалізуємо його нульовим рівнем (заповнюємо всі клітинки нулями);

2. Беремо випадкову точку на ландшафті або біля його кордонів (за межами), а також беремо випадковий радіус у заздалегідь заданих межах.

Вибір цих меж впливає на вид ландшафту — небудь він буде пологим, або скелястим;

3. У вибраній точці "піднімаємо" пагорб заданого радіуса;

4. Повертаємося до другого кроку і так далі до вибраного кількості кроків. Від вибраного кількості кроків потім буде залежати зовнішній вигляд нашого ландшафту;

Перший, другий і четвертий тривіальні кроки, тепер займемося третім кроком. Фактично пагорб — це в нашому випадку половина кулі, чим більше радіус — тим більше пагорб (і вище). Математично це схоже на перевернутий параболу. Для отримання висоти точки в нашому горбі можна використовувати наступну формулу:

$$z = h - \frac{((x_2 - x_1)^2 + (y_2 - y_1)^2) * h}{r^2},$$

де (x1, y1) — задана точка, r — обраний радіус, h – вибрана висота пагорба, (x2, y2) — центр пагорба.

Однак, можна взяти висоту дорівнює квадрату радіуса (h = r^2), тоді формула набуде вигляду:

$$z = r^2 - ((x_2 - x_1)^2 + (y_2 - y_1)^2).$$

Що б згенерувати ландшафт повністю нам необхідно побудувати безліч таких пагорбів. Але є ще дві речі, на які необхідно звернути увагу. Перше — необхідно ігнорувати негативні значення висоти пагорба. Друге — при генерації наступних пагорбів краще додавати отримане значення для даного пагорба до вже існуючих значень. Це дозволяє побудувати більш правдоподібний ландшафт, ніж правильно окреслені округлі пагорби. Результати роботи даного методу зображено на рисунку 3.11.

Далі рекомендується отриману карту згладити і інтерполювати. Для більшої реалістичності гори можна згенерувати окремо (окремим алгоритмом) і потім розмістити в довільному порядку на карті.

У багатьох випадках ми можемо використовувати вже розглянутий нами алгоритм для генерації ландшафтів. Але іноді необхідно згенерувати

острова, або острів. В цьому нам допоможе той же алгоритм, правда злегка модифікований.

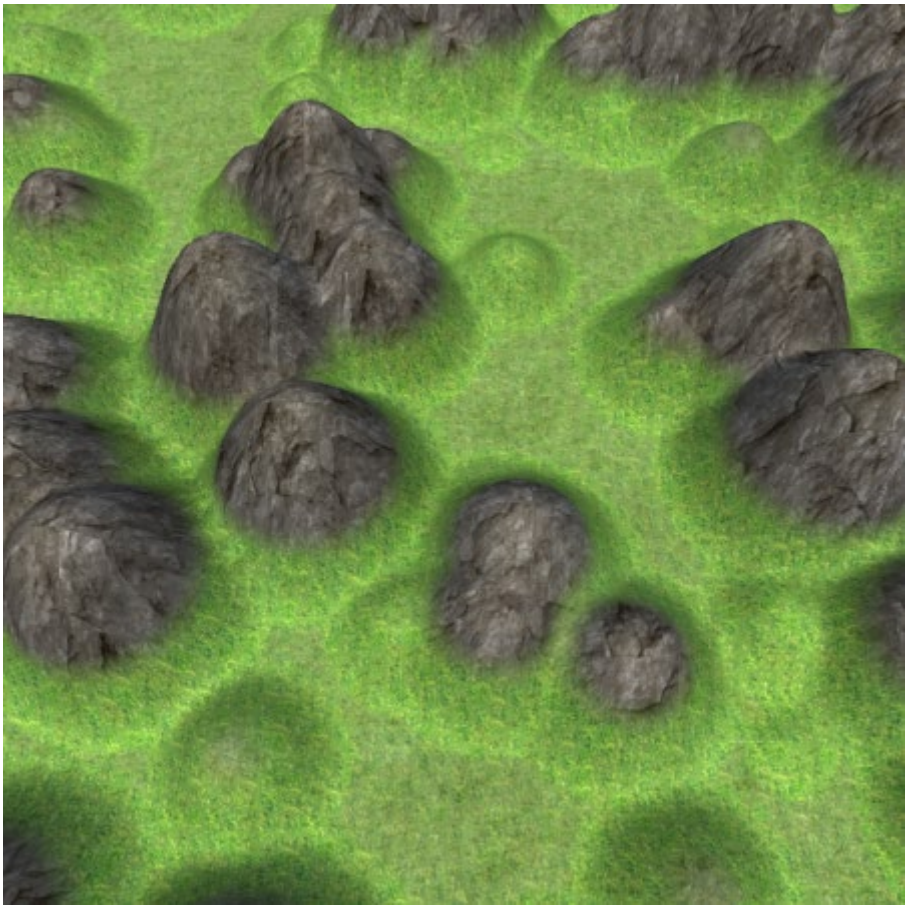

Рисунок 3.11 – результати роботи Холмового алгоритму

У вихідному алгоритмі ми вибирали центральну точку випадковим чином, і вона могла розташовуватися в будь-якій частині ландшафту. Тепер же нам цікаво, щоб пагорби були розташовані ближче до центру. Щоб зробити це, введемо дві змінних (які потім будемо випадковим чином змінювати), назвемо їх відстань і кут. Відстань буде означати, як далеко від центру знаходиться центральна точка для одиночного пагорба. Воно може змінюватися від нуля (прямо по центру карти висот) до половини величини карти висот мінус радіус пагорба. Це дозволить нам уникнути ситуацій перетину пагорбів з краєм карти висот. Кут буде показувати, в якому напрямку від центру нам потрібно буде поставити пагорб. Змінюється в межах від 0 до двох Пі. Використовуючи ці два значення, ми можемо

отримати значення (x, y) для центральної точки конкретного пагорба і використовувати їх як і в простому алгоритмі.

Ось як нам можна отримати значення x та y:

$$theta = random(0, \pi)$$

$$distance = random(0, size/2 - radius)$$

$$y = size/2 + cos(theta) * distance$$

$$x = size/2 + sin(theta) * distance$$

тут size — розмір карти висот, distance — відстань, theta — кут. Радіус повинен бути менше половини розміру карти висот.

Плюси даного алгоритму: певна передбачуваність результату (в результаті роботи даного алгоритму гарантовано буде гірський ландшафт), простота алгоритму, а також швидкість генерування ландшафту (швидкість генерування ландшафту даним алгоритмом досить висока і це частково є наслідком попереднього пункту).

Мінуси: у даній реалізації гори дуже схожі (незважаючи на різну висоту і радіус), відсутність шорсткостей і відсутність рівнин, западин і багато іншого.

Алгоритм Diamond-Square — метод генерації карт висот у комп'ютерної графіці. Ідея вперше була введена Фурньє, Фусселем і Карпентером на конференції siggraph 1982.

Алгоритм починає роботу з 2D сітки, потім з чотирьох початкових значень, випадковим чином генерує карту висот, упорядковану у вигляді сітки з точок так, щоб вся площина була покрита квадратами.

Алгоритм Diamond-Square є розширення одновимірного алгоритму Midpoint displacement на двовимірну площину. Ландшафти, отримані з його допомогою, як правило, називають фрактальними, хоча, слід визнати, насправді вони не такі вже одноманітні — навпаки, як ми побачимо нижче, їх не дуже приємним властивістю є те, що у великому масштабі вони стають

відносно гладкими, а в дрібному перетворюються на подобу наждачного паперу.

Почнемо з більш простого алгоритму Midpoint displacement. Він працює не на двовимірній площині, а на одновимірному відрізку (тому з його допомогою можна, наприклад, створити лінію горизонту).

Те, що ріднить цей алгоритм з фракталами — це його рекурсивна поведінка. Спочатку будь-яким чином задається висота на кінцях відрізка і відрізок розбивається крапкою посередині на дві під-відрізка. Потім цю точку зміщують на випадкову величину, далі розбиття і зміщення повторюється для кожного з отриманих під-відрізків. І так далі — поки відрізки не стануть довжиною в один піксель. Важливе зауваження: випадкові зміщення повинні бути пропорційні довжинам відрізків, на яких произведятся розбиття. Наприклад, при розбитті відрізка довжиною l —точка посередині нього повинна мати висоту

$$h = \frac{h_l + h_r}{2} + random(-R * l; R * l),$$

де hL і hR — висоти на лівому і правому кінці відрізка, а константа R визначає «шорсткість» (roughness) получающейся ламаної і є головним параметром в даному алгоритмі.

Для отримання двовимірної карти висот випадкове значення присвоюється всім чотирьом кутам карти. Слід врахувати, що для коректної роботи даного алгоритму карта повинна бути квадратною. Як показано на рисунку 3.12 точка в центрі виходить усередненням висот всіх 4 кутових точок, а кожна серединна точка на стороні великого квадрата — усередненням пари точок, що лежать на кінцях відповідної сторони. Далі необхідно привнести трохи шуму — зрушити випадковим чином центральну точку вгору або вниз (у розмірах, пропорційних стороні квадрата) — і можна повторювати рекурсивно дії для отриманих під-квадратів.

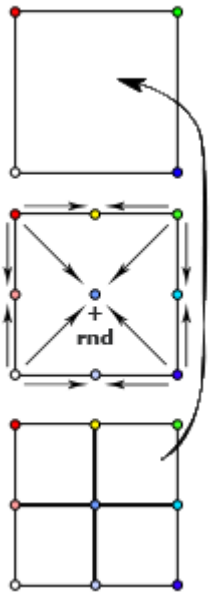

Рисунок 3.12 – Алгоритм Midpoint displacement

Однак, це ще не Diamond-Square — даний алгоритм, як правило, теж називають алгоритмом midpoint displacement і незважаючи на те, що він дає вже відносно прийнятні результати, готової картинці без особливої праці можна помітити її «прямолінійну» натуру.

Алгоритм Diamond-Square — алгоритм, який дозволяє отримувати «справжні» фрактальні ландшафти — відрізняється від двовимірного Midpoint displacement тим, що складається з двох кроків. Як показано на рисунку 3.13 - перший — т. зв. «square» —визначає центральну крапку в квадраті шляхом усереднення кутових і додаванням власного displacement'a — випадкового відхилення. Другий крок — «diamond» — покликаний визначити висоту точок, що лежать на серединах сторін. Тут усереднюються не дві точки — «зверху» і «знизу» (якщо говорити про точки на вертикальній стороні), але і пара точок «ліворуч» і «праворуч» — тобто ще дві отриманих на кроці «square» центральних точки. Важливо зауважити, що ці дві висоти, які дісталися нам на попередньому кроці, повинні бути вже пораховані — тому обрахування потрібно вести «шарами», спочатку для всіх квадратів виконати крок «square» — потім для всіх ромбів виконати крок «diamond» — і перейти до менших квадратів.

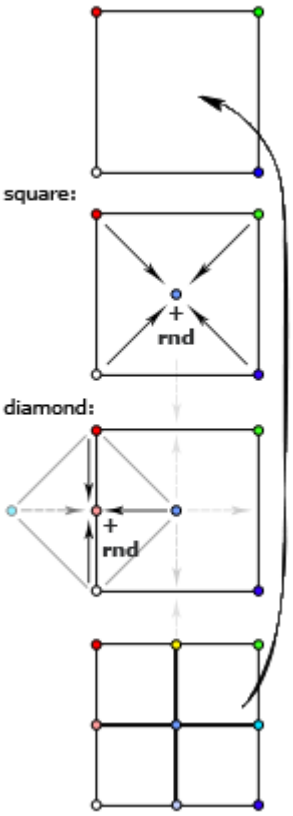

Рисунок 3.13 – Алгоритм Diamond-Square

Крім необхідності використовувати, скажімо так, обхід в ширину замість обходу в глибину, є ще одна тонкість — ситуація на краях ландшафту. Справа в тому, що на етапі «diamond» алгоритм використовує висоту точок, які знаходяться за межами поточного квадрата і, можливо, всієї карти. Є кілька варіантів вирішення цієї проблеми: або вважати ці висоти рівними 0 (або 1, або будь-який інший константі; це зручно для занурення країв нашого ландшафту під воду), або при спробі вважати висоту точки, що лежить на 64 пікселя ліворуч від лівої межі карти, зчитувати висоту точки, яка відповідає 64 точки від правої межі. На рисунку 3.14 показаний результат роботи даного алгоритму.

До плюсів даного алгоритму слід віднести велику «природність» сгенерированих ним ландшафтів. До мінусів даного алгоритму слід віднести менша, порівняно з попереднім алгоритмом швидкість генерації ландшафту, а також обмеження у формі ландшафту (ландшафт повинен бути обов'язково квадратним) пов'язане з особливостями алгоритму.

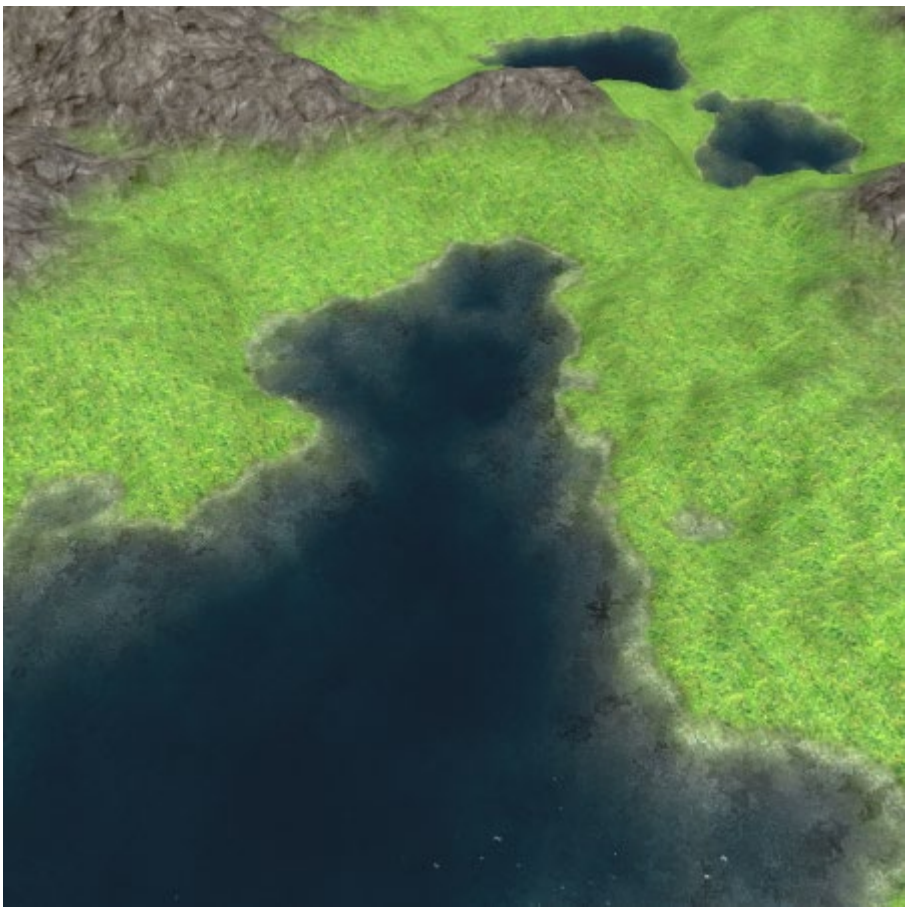

Рисунок 3.14 – Результат роботи алгоритму diamond-square

Такі основні алгоритми побудови карт висот для генерації ландшафтів. На мій погляд, найбільш реалістичні результат дає diamond-square — хоча і він не позбавлений деяких недоліків. Наприклад, створивши карту, яка гарна виглядає при сильному наближенні — при перегляді цілком ви побачите безліч дрібних острівців (а то і зовсім суцільний шум) замість кількох великих материків і океанів. Самоподібності не виходить.

3.3 Пост-обробка ландшафту

Більшість алгоритмів генерації тривимірних ландшафтів передбачають пост-обробку вихідних ландшафтів, таку як нормалізація, долінізація і згладжування.

При генерації карти висот ландшафту зазвичай не враховується вихід цих значень за деякі межі (наприклад — якщо ландшафт буде зберігатися у форматі int16, то необхідно, щоб усі значення знаходилися в межах від -

32768 до 32767). Для цього нам необхідно провести нормалізацію значень. Математично нормалізація — це процес отримання даних з одного межі, і переведення його в інші межі. На рисунку 3.15 показано як це виглядає графічно.

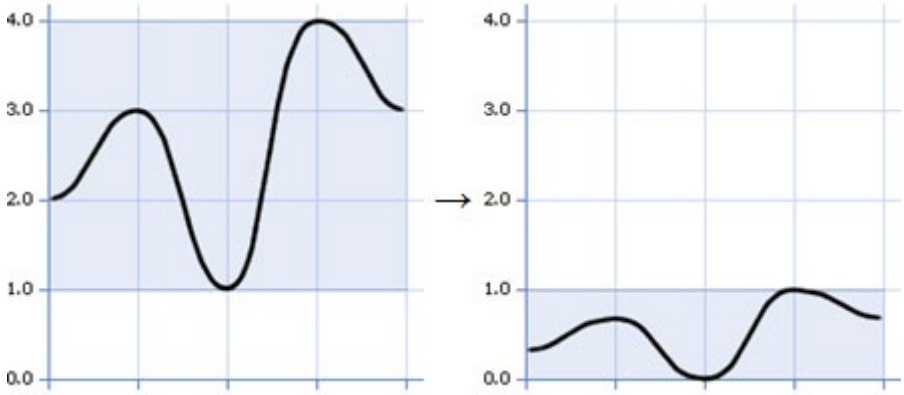

Рисунок 3.15 – Нормалізація ландшафту

Щоб це зробити ми виробляємо наступні дії:

• спершу проходимо по всьому масиву і запам'ятовуємо найбільше і найменше значення;

• після того, як ми дізналися ці значення, ми знову проходимо по всьому ландшафту і виробляємо нормалізацію конкретних значень в межі від 0 до 1. У вигляді формули це виглядає так:

$$z_{norm} = \frac{z - z_{min}}{z_{max} - z_{min}}$$

Після цього ми маємо ландшафт, нормалізований і готовий до подальшого використання. Тепер перейдемо до питання про долінізації ландшафту.

Більшість з алгоритмів генерації тривимірних ландшафтів є слабо контрольованими: наприклад, у ландшафтах що генеруються Холмовим алгоритмом звичайно занадто мало рівнин. Схили пагорбів надто круті, що обумовлено параболічної формулою їх знаходження. Виправити цей недолік можна за допомогою долинизации. Перед долинизацией ландшафт повинен бути нормалізований, тобто всі значення повинні знаходиться в межах від 0 до 1. Ідея "долинизации" полягає в наступному — взяти від кожного

значення квадратний корінь. Це більшою мірою впливає на середні значення, практично не зачіпаючи мінімумів і максимумів. На рисунку 3.16 показано як це виглядає графічно.

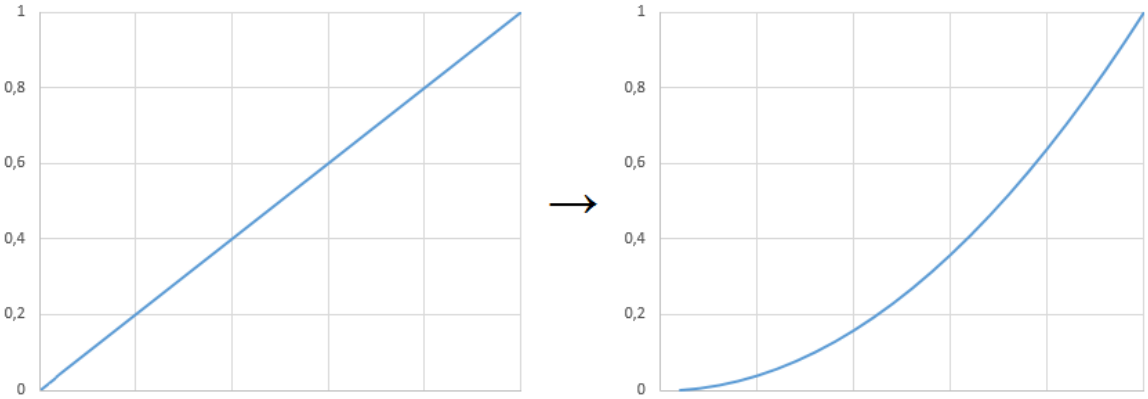

Рисунок 3.16 – Долинизация гірського ландшафту

Ця процедура, на перший погляд, дуже схожа на інтерполяцію – коли проміжні значення перебувають з використанням певної формули. Однак, є суттєві відмінності: нове значення присвоюється всім точкам крім екстремумів (а не тільки проміжним точкам) і при обчисленні нового значення точки не враховуються значення найближчих точок, ні яких-небудь інших точок на карті висот. Нове значення обчислюється за певною формулою (в даному прикладі – формула $h_{new} = \sqrt{h_{old}}$) зачіпає лише дану точку.

Плюси даного підходу очевидні: для знаходження значення N-ної точки необхідно лише старе значення N-ної точки – у зв'язку з цим швидкість роботи даної процедури значно вище швидкості роботи згладжування.

Для долинизации гірського ландшафту було достатньо простої формули, проте іноді потрібно застосовувати більш складні залежності. Наприклад, берега річок й морів на ландшафтах що генеруються за допомогою алгоритму Diamond-Square часто виходять дуже пологими. Позбутися від цього можна з допомогою долинизации за формулою $h_{new} = h_{old}^{10x-8}$. На рисунку 3.17 показано як це виглядає графічно.

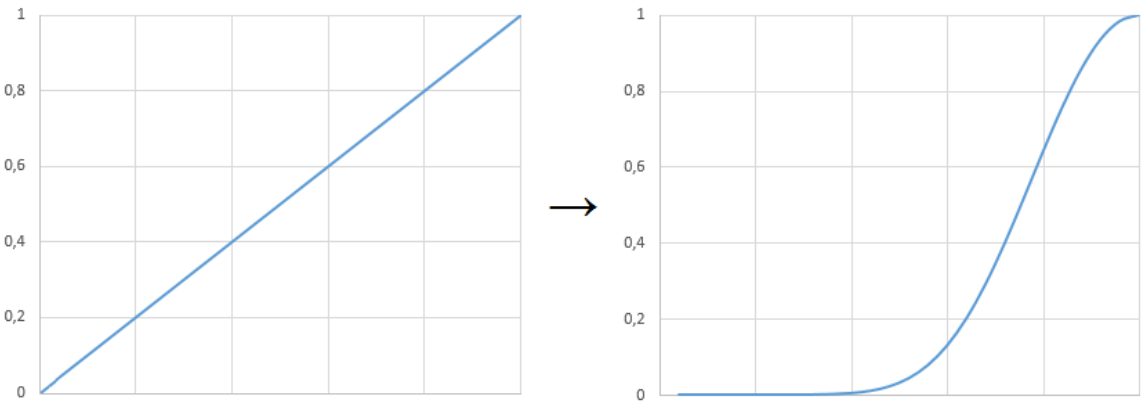

Рисунок 3.17 – Долинизация берегів річок і морів

Згенеровані ландшафти часто володіють деякою "зубчастістю" або деякою кількістю артефактів. Для усунення цих ефектів зазвичай використовується згладжування.

Згладжування — технологія, використовувана для усунення ефекту «зубчасті», що виникає на краях одночасно виведеної на екран безлічі окремих плоских або об'ємних зображень. Згладжування було придумано в 1972 році в Массачусетському технологічному інституті в Architecture Machine Group, яка пізніше стала основною частиною MediaLab.

Основний принцип згладжування — пікселі, сусідні з граничним пікселем зображення, приймають проміжне значення між кольором зображення і кольором фону, створюючи градієнт і розмиваючи межу.

Результат роботи найпростішої варіації згладжування показаний на рисунку 3.18.

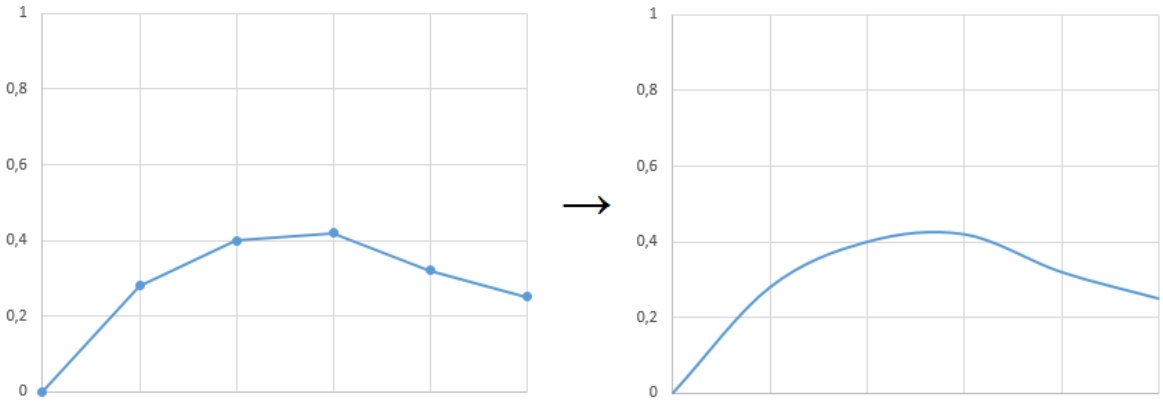

Рисунок 3.18 – Згладжування ландшафту

У загальному випадку згладжування передбачає наступну процедуру: нове значення висоти кожної точки на карті висот виходить як середнє арифметичне прилеглих висот. Також часто використовуються різні модифікації описаного алгоритму.

Як видно з описаного вище, що згенеровані ландшафти часто мало придатні для використання і вимагають додатково етап обробки: нормалізації, долінізації і згладжування. Нормалізація ландшафту дозволяє задати певні межі для генерованого ландшафту, долінізація і згладжування є додатковими процедурами що дозволяють отримати більш реалістичну картинку на виході.

3.4 Реалізація комбінованого алгоритму генерації

Всі розглянуті вище алгоритми генерації тривимірних віртуальних ландшафтів мають свої слабкі і сильні сторони. Алгоритм Diamond-Square краще підходить в одному випадку, Холмовий алгоритм в іншому і так далі. Зміст комбінованого алгоритму полягає в тому, щоб використовувати кожен з розглянутих алгоритмів там – де він найбільше підходить.

Генерований тривимірний ландшафт був умовно розділений на дві частини: частина під водою і частина над водою. Річки, моря та все, що нижче рівня моря генерував модифікований алгоритм Diamond-Square. Частина над рівнем моря – рівнини та гори, генерував Холмовий алгоритм.

Алгоритм Diamond-Square відмінно справляється з генерацією морів. Однак, як показано на рисунку 3.19 – відсоток поверхні заповненої водою завжди різний. Перша задача, яка була поставлена перед комбінованим алгоритмом – це зовнішній жорсткий контроль над тим, яка частина поверхні буде заповнена водою.

Жорсткого зовнішнього контролю над тим, яка частина поверхні буде заповнена водою можна досягти за допомогою наступного алгоритму:

1. Отримуємо загальний розподіл висот на карті висот за допомогою сортування елементів.

2. Визначаємо «рівень моря» з використанням розподіленої карти і введеного користувачем значення.

3. Интерполюемо карту висот таким чином, щоб частина під «рівнем моря» дійсно перебувала під рівнем моря.

Рисунок 3.19 – Відсоток водної поверхні в ландшафтах згенерованих алгоритмом Diamond-Square

На рисунку 3.20 показаний реальний рівень моря і рівень моря з урахуванням введеного користувачем коефіцієнта.

Рисунок 3.20 – Контроль рівня моря

Однак, при використанні лінійної інтерполяції можуть виникнути дуже пологі моря або навпаки надто круті береги. Для усунення цього ефекту достатньо скористатися простою лінійною функцією: $h_{new} = h_{old}^{x}$, де x – обчислюється виходячи з введених параметрів.

Для прискорення роботи алгоритму Diamond-Square – даним алгоритмом виходять не всі висоти (що також можна контролювати зовнішнім параметром), проміжні значення знаходяться за допомогою лінійної інтерполяції.

Наступним етапом після генерації водних поверхонь буде генерація гір і рівнин. Генерація гір і рівнин виробляється з допомогою модифікації Холмового алгоритму.

Відсоток поверхні, який необхідно заповнити пагорбами задається користувачем. Крім того, пагорби не можуть розташовуватися в морях.

Окремий алгоритм, як показано на рисунку 3.21, шукає всі місця в яких може бути розташований даний (конкретний, так як вони відрізняються за формою і розмірами) пагорб. Потім псевдовипадковим чином вибирається місце розташування цього пагорба з можливих.

Рисунок 3.21 – Пошук можливого розташування пагорба

Даний алгоритм дозволяє створювати реалістичні ландшафти при порівняно невеликих витратах і при цьому, що дуже важливо, забезпечує досить жорсткий контроль над виходідним ландшафтом.

3.5 Порівняльні тести

Вище описані деякі алгоритми генерації тривимірних віртуальних ландшафтів, а також описано їх слабкі і сильні сторони. В даному розділі будуть описані результати порівняльних тестів, що дозволить зробити більш правильні висновки з приводу цих алгоритмів.

Було вироблено три виміру для двох різних розмірів карт: 513x513 і 1025x1025.

В таблиці 3.1 наведено результати вимірів швидкості роботи, а також використованої алгоритмом пам’яті для трьох розглянутих вище алгоритмів для ландшафту розміром 513x513.

Таблиця 3.1 – Результати тестів для ландшафту розміром 513x513

Номір виміру	Холмовий алгоритм		Diamond-Square		Комбінований алгоритм	
	Час викон. (сек.)	RAM (Мб)	Час викон. (сек.)	RAM (Мб)	Час викон. (сек.)	RAM (Мб)
1	0,1389569	0,69	0,08398	0,53	0,214744	1,12
2	0,1353383	0,74	0,087219	0,57	0,180938	0,99
3	0,1280751	0,71	0,084854	0,59	0,199117	1,07
середне значення	0,1341234	0,713333	0,085351	0,563333	0,198266	1,06

В таблиці 3.2 наведено результати вимірів швидкості роботи, а також використованої алгоритмом пам’яті для трьох розглянутих вище алгоритмів для ландшафту розміром 1025x1025. Також виведені середні значення отриманих результатів.

Таблиця 3.2 – Результати тестів для ландшафту розміром 1025x1025

Номір виміру	Холмовий алгоритм		Diamond-Square		Комбінований алгоритм	
	Час викон. (сек.)	RAM (Мб)	Час викон. (сек.)	RAM (Мб)	Час викон. (сек.)	RAM (Мб)
1	0,5542846	1,72	0,325956	1,44	0,835866	2,19
2	0,5545734	1,63	0,320803	1,49	0,814341	1,97
3	0,5503648	1,77	0,323103	1,39	0,827462	2,01
середне значення	0,5530742	1,706667	0,323287	1,44	0,825889	2,056667

З таблиць видно, що найменьш вимогливим алгоритмом є алгоритм Diamond-Square. Даний алгоритм (або його модифікації) застосовується у багатьох програмах, це обумовлено як порівняно швидкою роботою алгоритму, так і досить хорошими результатами на виході. Комбінований

алгоритм є досить складним, але показав досить гарні результати по швидкості і використовуваної пам'яті.

Проте слід також зазначити, що тестування проводилося хоч і в кілька вимірів – однак, всеж на одному ПК. Безумовно, на різних системах результати будуть відрізнятися, однак найімовірніше, що не значно.

3.6 Висновки до розділу 3

У даному розділі були розглянуті: алгоритми генерації псевдовипадкових чисел та їх особливості; особливості представлення даних о ландшафтах; алгоритми генерації тривимірних віртуальних ландшафтів; постобробка ландшафтів. Також розроблено комбінований метод і програмне забезпечення генерації ландшафтів, на основі модифікацій розглянутих раніше методів. Проведені порівняльні тести.

ВИСНОВКИ

У дипломній роботі були розглянуті основні аспекти проектування та розробки генератора і візуалізатора тривимірних зображень ландшафтів. При цьому отримано наступні результати:

– проведений аналіз предметної області;

– проведений аналіз інструментальних засобів для розробки генератора і візуалізатора тривимірних віртуальних ландшафтів та їх обґрунтований вибір;

– описано принципи побудови (візуалізації) тривимірних зображень;

– розглянуто методи генерації псевдовипадкових чисел;

– розглянуто методи генерації тривимірних віртуальних ландшафтів;

– розроблено програмний інтерфейс для реалізації поставлених завдань;

Реалізований програмний засіб відповідає всім вимогам функціональності, так як виконує всі покладені на нього функції, так і вимогам надійності. Поставлене завдання було виконане.

ПЕРЕЛІК ПОСИЛАНЬ

1. Авдєєва С. М., Куров А. В. Алгоритми тривимірної машинної графіки: Навчальний посібник. – М: Вид-во МДТУ їм. Н.Е. Баумана, 1996. – 60 с.: іл.

2. Роджерс Д. Алгоритмічні основи машинної графіки: пер. з англ.— М.: Світ, 1989.— 512 с.: іл.

3. Еликов Н. Практична модель динамічних атмосферних явищ при візуалізації відкритих просторів в системах візуалізації реального часу. Новосибірськ: Лабораторія програмних систем машинної графіки Іаіе ЗІ РАН, 2004. 8с.

4. Снук Р. 3D ландшафти у реальному часі на C++ і DirectX9. - М.:КУДИЦ-Образ, 2007 - 157 с.

5. Джозеф Хокінг. Unity у дії. - СПб.:Пітер, 2016 - 333 с.

ДОДАТОК А

UML ДІАГРАМИ

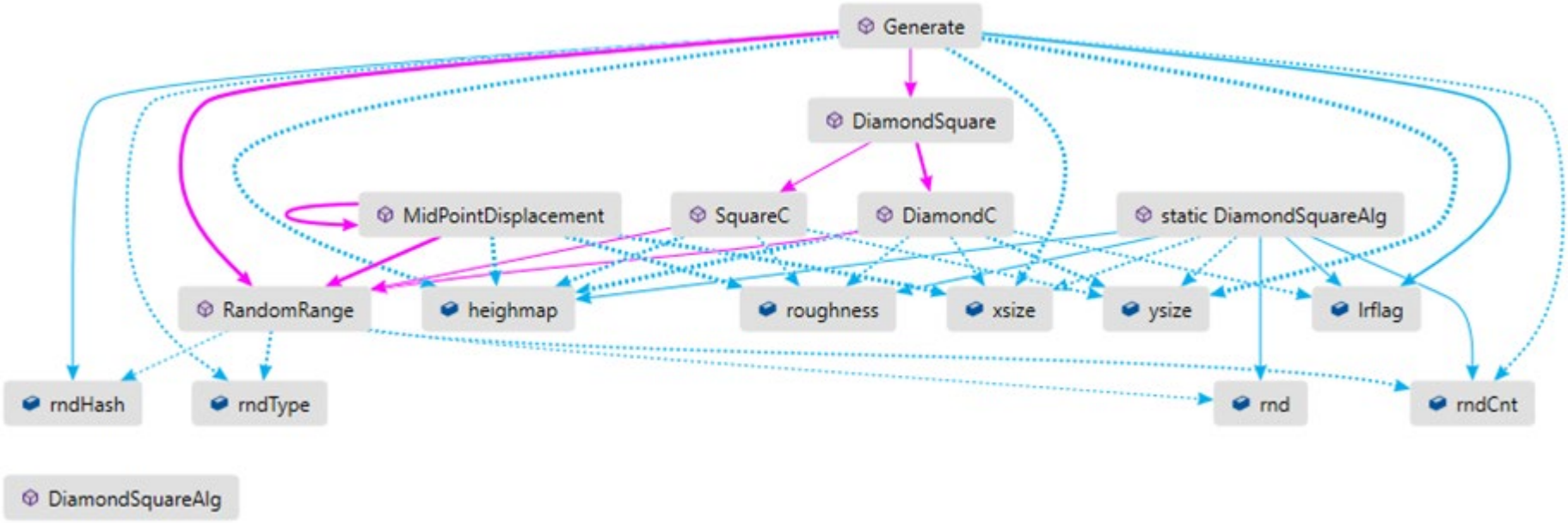

Рисунок А.1 – UML-діаграма класу DiamondSquareAlg

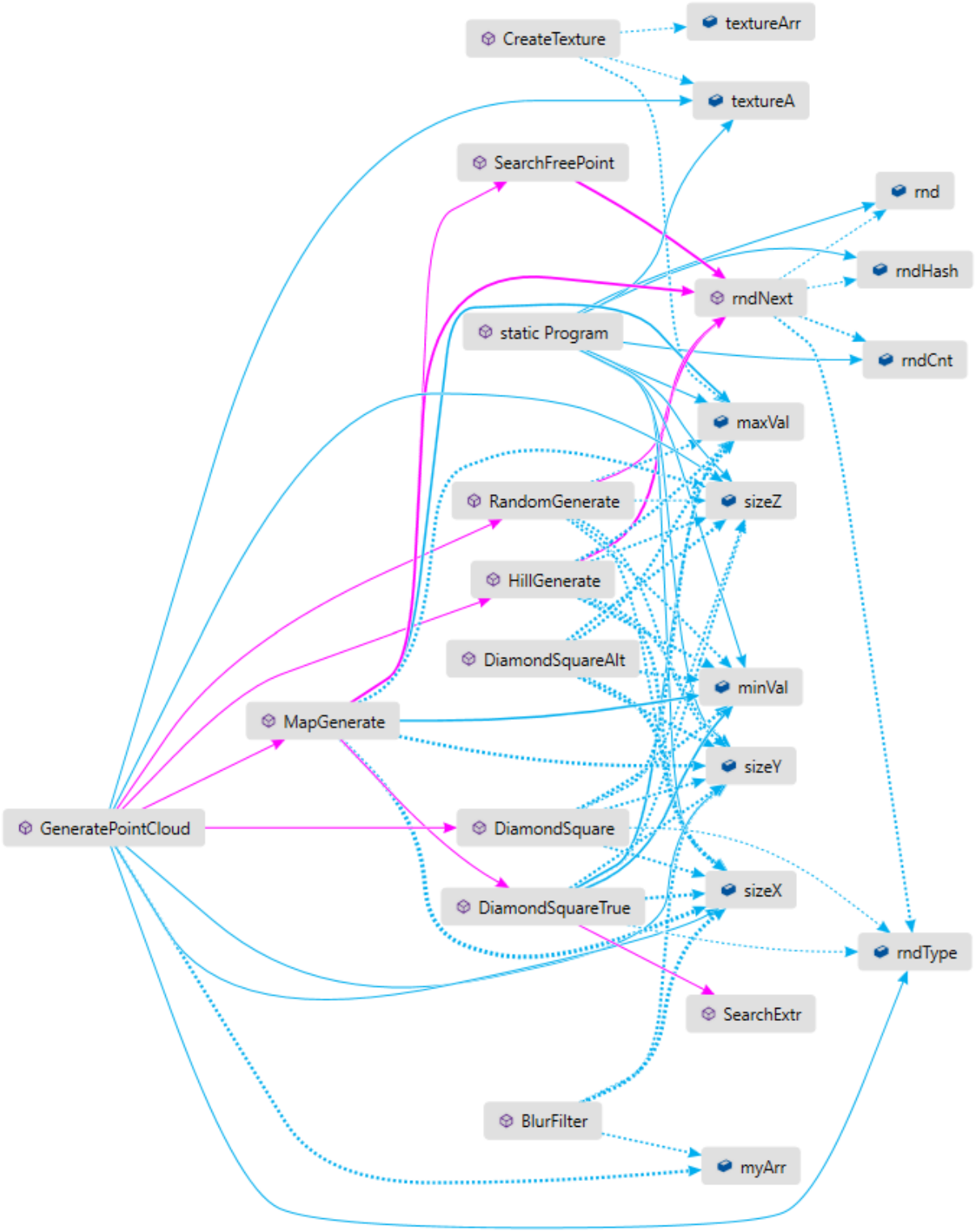

Рисунок А.2 – UML-діаграма головного класу

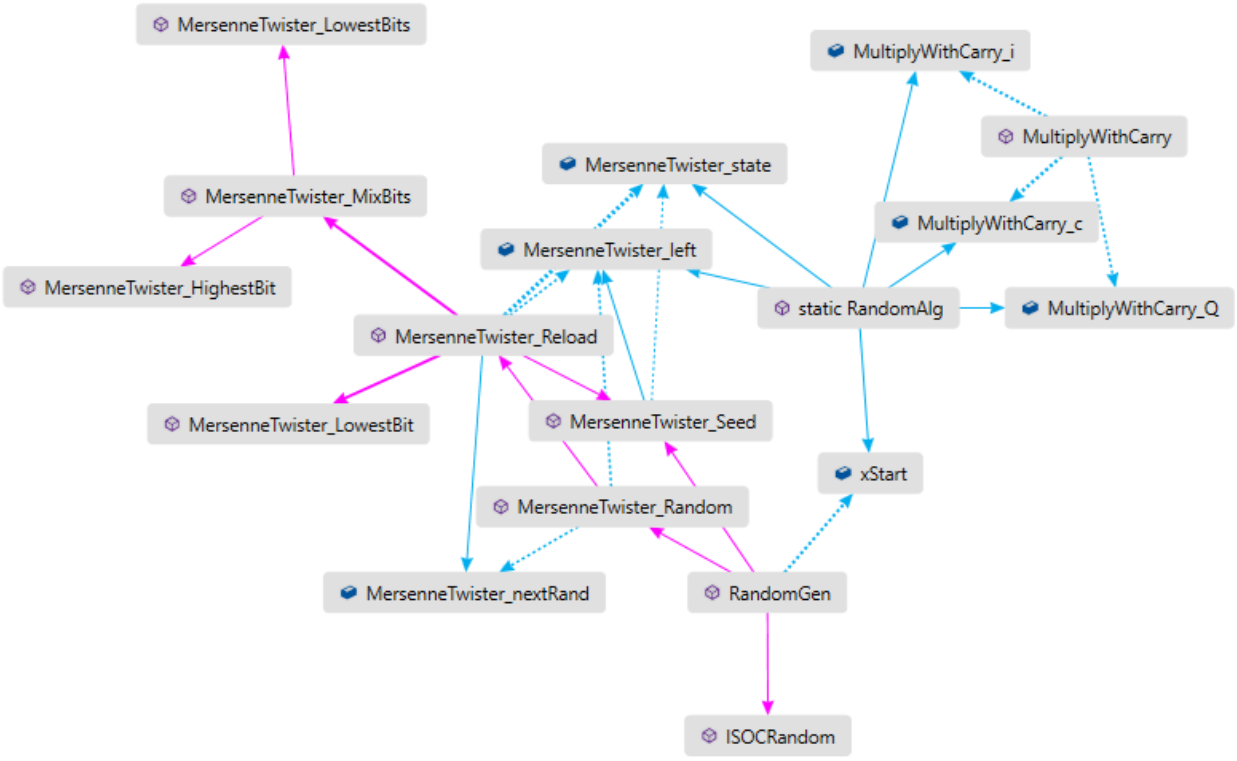

Рисунок А.3 – UML-діаграма класу RandomAlg

ДОДАТОК Б

ІНТЕРФЕЙС ПРОГРАМНОГО ЗАСОБУ

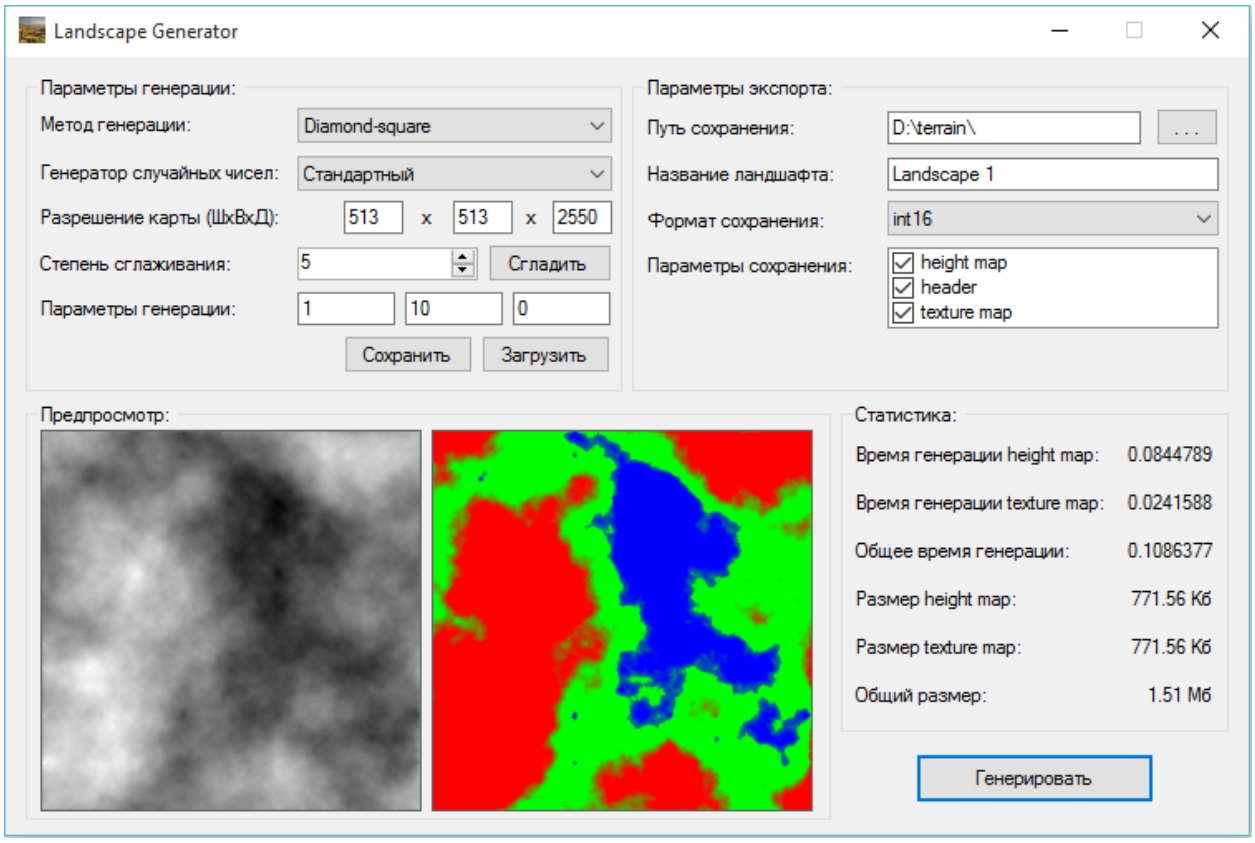

Рисунок Б.1 – Головна форма генератора ландшафтів

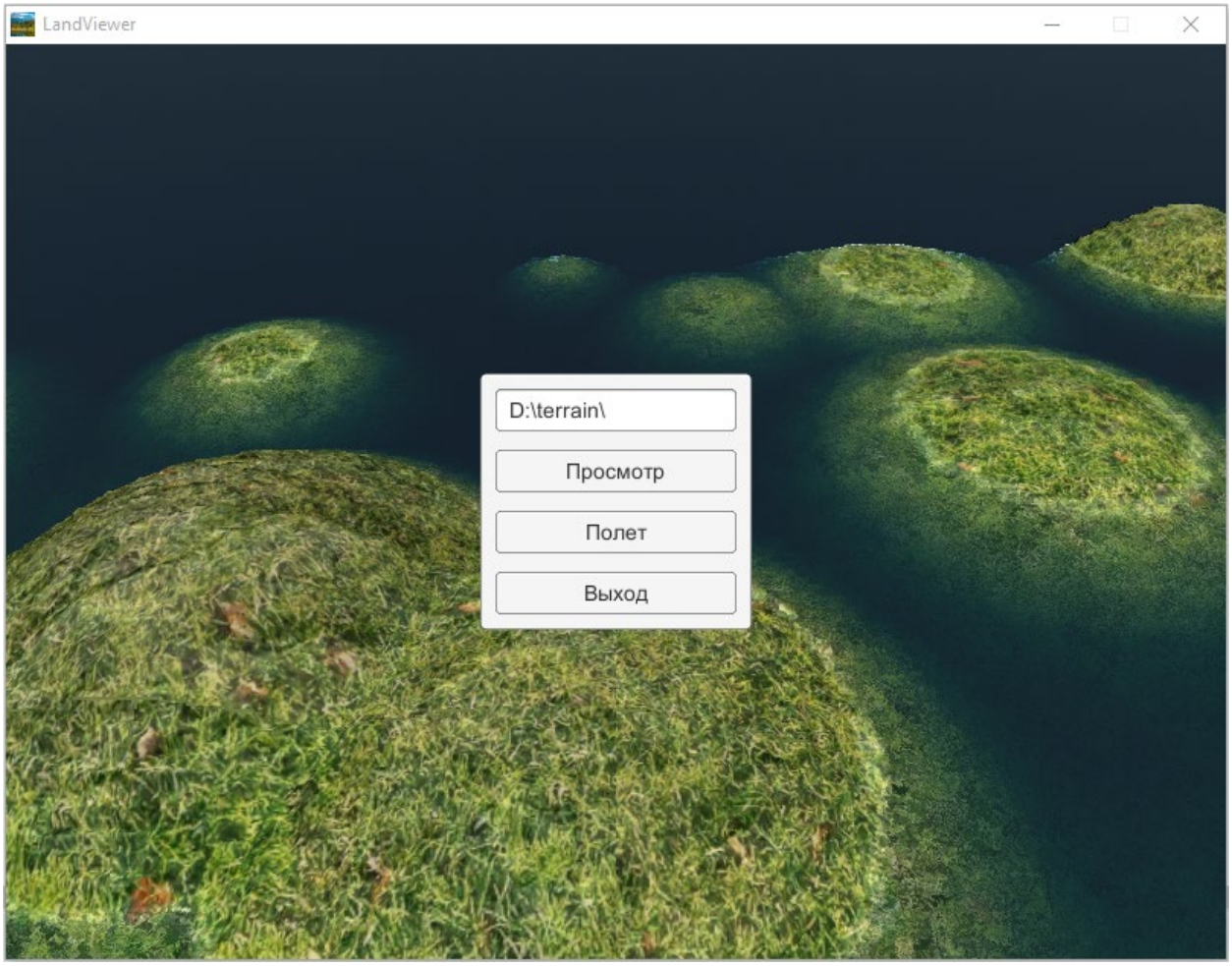

Рисунок Б.2 – Головне меню візуалізатора

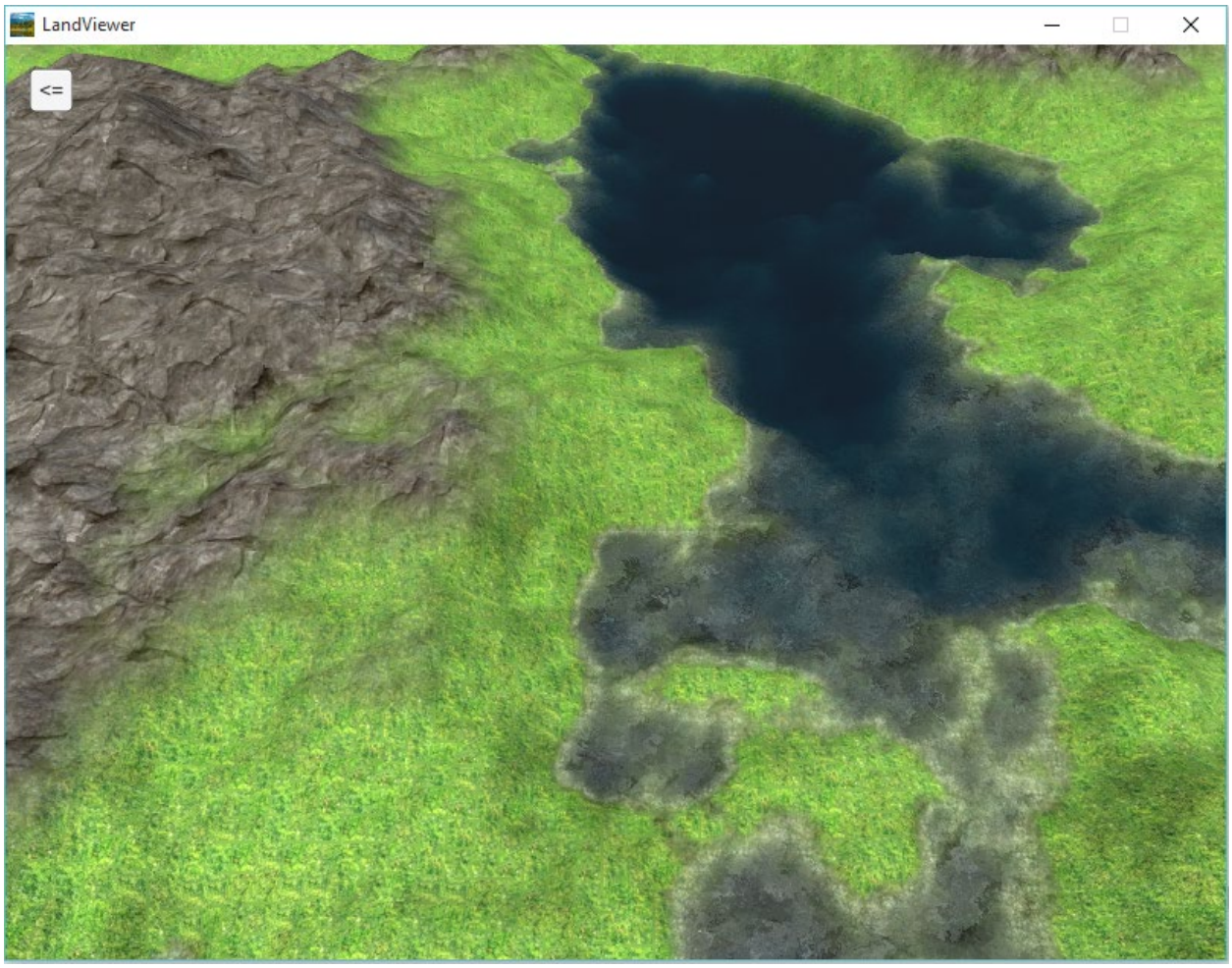

Рисунок Б.3 – Візуалізація ландшафту