НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК 004.092

«До захисту допущено»

Завідувач кафедри СПСКС

_____Віталій Романкевич
(підпис) (ініціали, прізвище)

"___"_____2020р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія
(Спеціалізовані комп'ютерні системи)

на тему: ПРОГРАМНІ ЗАСОБИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ЛАНДШАФТУ

Виконав: студент ІІ курсу, групи КВ-93мп

_____Пасічник Максим Юрійович _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник ст.викл. Дробязко Ірина Павлівна

_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю доцент, с.н.с.,к.т.н. Юлія БОЯРІНОВА

_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)
за освітньо-професійною програмою
Спеціальність 123 Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____Віталій Романкевич
(підпис) (ініціали, прізвище)

1 листопада 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студентки
Пасічника Максима Юрійовича

1. Тема дисертації: ПРОГРАМНІ ЗАСОБИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ЛАНДШАФТУ,
науковий керівник дисертації: старший викладач Д.І. Павлівна,
затверджені наказом по університету від "12" листопада 2020 р. № 3298-С.

2. Термін подання студентом дисертації: 17 грудня 2020 р.

3. Об'єкт дослідження: методи та алгоритми процедурної генерації ландшафтів.

4. Предмет дослідження: програмна реалізація розробленого алгоритму процедурної генерації ландшафту.

5. Перелік завдань, які потрібно розробити:
- проаналізувати існуючі системи процедурної генерації ландшафту;
- дослідити методи оптимізації існуючих алгоритмів;
- розробити та описати алгоритм оптимізації;
- реалізувати програмний засіб процедурної генерації ландшафту;
- проаналізувати отримані результати.

6. Перелік ілюстративного матеріалу:
- Презентація;

– Фрагменти програмного коду.

7. Перелік публікацій: «ОПТИМІЗАЦІЯ РОЗРІДЖЕНОГО ВОКСЕЛЬНОГО ОКТОДЕРЕВА З ВИСОКОЮ РОЗДІЛЬНОЮ ЗДАТНІСТЮ», XIII науково-практична конференція магістрантів та аспірантів ПМК-2020; «ПОРІВНЯННЯ АЛГОРИТМІВ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ВОКСЕЛЬНОЇ ГЕОМЕТРІЇ», Міжнародна наукова інтернет-конференція "Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення"(випуск 48).

8. Дата видачі завдання 1 листопада 2019 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Вивчення матеріалів за темою дисертації. Визначення структури магістерської дисертації	15.11.2020	
2	Аналіз існуючих рішень	12.03.2020	
3	Підготовка матеріалів першого розділу магістерської дисертації;	25.07.2020	
4	Підготовка матеріалів другого розділу магістерської дисертації;	1.09.2020	
5	Розробка системи; підготовка матеріалів третього розділу магістерської дисертації;	5.10.2020	
6	Експериментальні дослідження; підготовка матеріалів четвертого розділу магістерської дисертації;	5.11.2020	
7	Завершення роботи над магістерською дисертацією; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2019	15.11.2020	
8	Оформлення документації магістерської дисертації	25.11.2020	
9	Попередній розгляд магістерської дисертації на кафедрі на кафедрі	26.11.2020	

Студент _____ Максим Пасічник

Науковий керівник дисертації _____ Ірина Дробязко

РЕФЕРАТ

Актуальність теми. В сучасному світі, особливої актуальності набуває індустрія розваг. Послуги цієї індустрії здатні задовільнити потреби людини і не рідко сприяють її розвитку. Вагоме місце в індустрії розваг посідають відеоігри.

Основним етапом розробки ігрових програм є створення навколишнього середовища. Для певних ігрових програм це може бути фонове зображення, а для деяких це може бути повноцінний ландшафт. Існує безліч програмних засобів, які дозволяють створювати ландшафти, щоб потім використовувати їх в ігрових програмах. Здебільшого алгоритми генерації ландшафту націлені на вирішення певної задачі, тому не завжди можна отримати бажаний результат. Ще однією, важливою проблемою є залежність від програмної архітектури застосунків, які використовуються, тому досить часто виникає проблема не раціонального використання пам’яті та інших обмежень.

Не менш важливим, є можливість модифікації згенерованого ландшафту, тому можна спостерігати суттєве підвищення інтересу до воксельної графіки. Зважаючи на підвищення інтересу до воксельної графіки, виникає необхідність в розробці ефективних алгоритмів опису воксельних даних.

Об’єктом дослідження є методи та алгоритми процедурної генерації ландшафту.

Предметом дослідження є програмна реалізація розробленого алгоритму процедурної генерації ландшафту.

Мета роботи: підвищення ефективності процедурної генерації ландшафту використовуючи модифіковане розріджене воксельне октодерево

Методи дослідження. В роботі використовуються системного аналізу, графічної візуалізації, оптимізації.

Наукова новизна полягає в запропонованому алгоритмі оптимізації процедурної генерації ландшафту, що дозволяє зменшити вимоги до пам’яті.

Практична цінність отриманих в роботі результатів полягає в тому, що запропонований алгоритм дозволяє зменшити необхідний обсяг пам’яті, який потрібний для опису згенерованого ландшафту. Це дозволить створювати більш складне навколишнє середовище, що значно підвищить візуальне сприйняття.

Структура та обсяг роботи. *Магістерська дисертація складається з вступу, чотирьох розділів, висновків та додатків.*

У вступі представлена загальна характеристика проблеми, обґрунтована необхідність розробки нового алгоритму, сформульована задача роботи.

У першому розділі розглянуто основні методи створення ландшафтів. Проведено порівняльний аналіз методів процедурної генерації ландшафту.

У другому розділі описується воксельна технологія, проаналізовані способи опису воксельних даних.

У третьому розділі описано алгоритми для генерації ландшафту в ігрових програмах; розроблено та описано етапи алгоритму генерації з використанням модифікованого розрідженого воксельного октодерева.

У четвертому розділі реалізовано розроблений алгоритм процедурної генерації ландшафту з використанням модифікованого розрідженого воксельного октодерева; описано структуру розробленого програмного продукту.

У висновках проаналізовано отримані результати.

У додатках наведено презентацію, лістинг розробленого програмного продукту, а також копії публікацій.

Магістерська дисертація виконана на 69 аркушах, містить 3 додатки та посилання на список використаних літературних джерел з 16 найменувань. У роботі наведено 72 рисунків та 1 таблиць.

Ключові слова: полігони, вокселі, процедурна генерація, ландшафт, методи оптимізації, розріджене воксельне октодерево.

ABSTRACT

Actuality of theme. In today's world, the entertainment industry is especially relevant. The services of this industry are able to meet human needs and often contribute to its development. Video games play an important role in the entertainment industry.

The main stage of game development is the creation of the environment. For some game programs it may be a background image, and for some it may be a full-fledged landscape. There are many software tools that allow you to create landscapes and then use them in gaming applications. For the most part, landscape generation algorithms are aimed at solving a specific problem, so it is not always possible to obtain the desired result. Another important problem is the dependence on the software architecture of the applications used, so quite often there is a problem of irrational use of memory and other limitations.

No less important is the ability to modify the generated landscape, so you can see a significant increase in interest in voxel graphics. Due to the growing interest in voxel graphics, there is a need to develop effective algorithms for describing voxel data.

The object of research is the methods and algorithms of procedural generation of the landscape.

The subject of research is the software implementation of the developed algorithm of procedural generation of the landscape.

Purpose: to increase the efficiency of procedural generation of the landscape using a modified sparse voxel octopus

Research methods. The work uses system analysis, graphical visualization, optimization.

The scientific novelty lies in the proposed algorithm for optimizing the procedural generation of the landscape, which reduces memory requirements.

The practical value of the results obtained in this work is that the proposed algorithm reduces the required amount of memory required to describe the generated landscape. This will create a more complex environment that will greatly enhance visual perception.

Structure and scope of work. The master's dissertation consists of an introduction, four chapters, conclusions and appendices.

The introduction presents the general characteristics of the problem, substantiates the need to develop a new algorithm, formulates the task.

The first section discusses the main methods of creating landscapes. The comparative analysis of methods of procedural generation of a landscape is carried out.

The second section describes the voxel technology, analyzes the methods of describing voxel data.

The third section describes algorithms for generating landscape in game programs; the stages of the generation algorithm using a modified sparse voxel octodree are developed and described.

In the fourth section the developed algorithm of procedural generation of a landscape with use of the modified rarefied voxel octodree is realized; describes the structure of the developed software product.

The results analyze the results.

The appendices provide a presentation, listing of the developed software product, as well as copies of publications.

The master's dissertation is made on 69 sheets, contains 3 appendices and links to the list of used literature sources from 16 titles. The paper contains 72 figures and 1 table.

Keywords: polygons, voxels, procedural generation, landscape, optimization methods, sparse voxel octopus.

ЗМІСТ

ДОДАТКИ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Воксель (від англ. Volume та англ. pixel) – аналог піклеля в тривимірному просторі.

Кросплатформеність – властивість програмного забезпечення працювати більш ніж на одній програмній або апаратній платформі.

Симплекс (від лат. simplex — простий) — геометрична фігура, що є багатовимірним узагальненням трикутника і тетраедра.

Чанк (з англ. Chunk – осередок, шматок, уламок) — частина нескінченного процедуно згенерованого ландшафту.

GPU (з англ. Graphics Processing Unit) – графічний процесор, відповідає за візуалізацію.

Октодерево (англ. Octree) – дерево, у якому кожна вершина має вісім дітей.

Розріджене вокслельне октодерево (англ. Sparse voxel octree, SVO) – регулярна ієрархічна структура даних, в основі якої лежить нерегулярна тривимірна сітка.

Спрямований ациклічний граф (англ. Directed Acyclic Graph, DGA) – випадок орієнтованого графа, в якому відсутні орієнтовані цикли.

Рівень деталізації (англ. Level Of Detail) – складність тривимірної геометрії.

ВСТУП

В сучасному світі особливої актуальності набуває індустрія розваг. Послуги цієї індустрії здатні задовільнити потреби людини і не рідко сприяють її розвитку. Вагоме місце в індустрії розваг посідають відеоігри.

Основним етапом розробки ігрових програм є створення навколишнього середовища. Для деяких ігрових програм це може бути фонове зображення, а для деяких це може бути повноцінний ландшафт. Існує багато програмних засобів, які дозволяють створювати ландшафти, щоб потім використовувати їх в ігрових програмах. Здебільшого алгоритми генерації ландшафту націлені на вирішення певної задачі, тому не завжди можна отримати бажаний результат. Ще однією важливою проблемою є залежність від програмної архітектури застосунків, які використовуються, тому досить часто виникає проблема не раціонального використання пам’яті та інших обмежень.

Не менш важливим є якість візуалізації згенерованого ландшафту і можливість модифікації під час ігрового процесу, тому традиційні технології на основі полігонів не підходять. Вони мають багато проблем: складність обчислень для модифікації поверхні; визначення колізій; величезні кодові бази. Альтернативою полігонам, що може вирішити ці проблеми є вокселі. Тому, можна спостерігати суттєве підвищення інтересу до воксельної графіки.

Вокселі є об’ємними за своєю природою, їх можна використовувати для моделювання внутрішньої частини об’єктів. Відеокарти не мають апаратної підтримки вокселей, тому виникає необхідність в розробці ефективних алгоритмів опису воксельних даних.

Метою магістерської дисертації є підвищення ефективності процесу генерації ландшафту за допомогою використання модифікованого розрідженого воксельного октодерева.

1. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ГЕНЕРАЦІЇ ЛАНДШАФТУ

1.1. Аналіз систем процедурної генерації ландшафту

Невід'ємною частиною будь-якої ігрової програми є навколишнє середовище, саме від нього залежить ігрова атмосфера. Останнім часом найбільшу популярність мають реалістичні навколишні середовища, через які користувач може максимально зануритись в ігровий процес. Основою будь-якого реалістичного навколишнього середовища є ландшафт.

Важливу роль у створенні процедурно згенерованого ландшафту відіграють програмні засоби. Ці програми використовуються для процедурної генерації ландшафтів із різними декораціями, елементами місцевості, річками, озерами, травою. Більшість програмних засобів створюють карту висот для опису базової місцевості, використовуючи при цьому псевдовипадкові числові значення у вигляді шуму, фракталів та інших алгоритмів. Це дозволяє з легкістю створювати поверхню, однак для створення скель, печер та різних складних структур потрібно використовувати інші алгоритми для генерації тривимірного шуму.

Наприклад, програма Terragen (рис. 1.1) дозволяє створювати фотореалістичні пейзажі із динамічним освітленням, але її недоліком є дуже повільна обробка і візуалізація даних, оскільки всі обчислення відбуваються на CPU.

Рисунок 1.1 – Ландшафт, згенерований в Terragen

Ще одним програмний засіб, який можна використовувати для

процедурної або ручної генерації ландшафту, є Grome (рис. 1.2). Його особливістю є нова система накладання текстур, що дозволяє використовувати різні методи затінення і процедурної генерації текстури на основі карти висот.

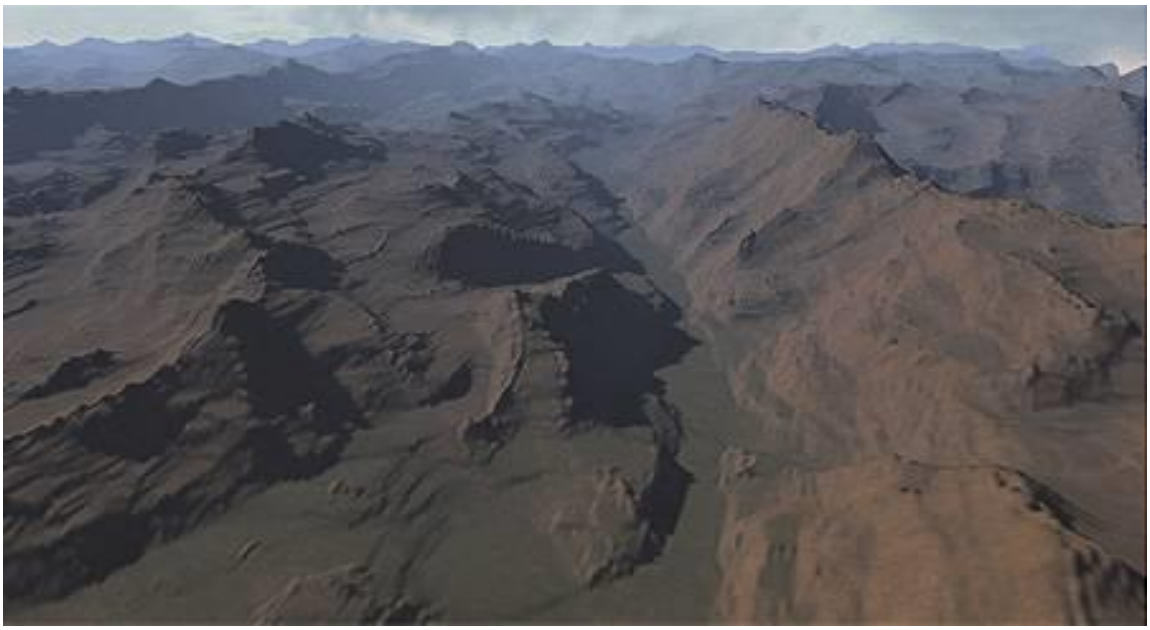


Рисунок 1.2 – Ландшафт, згенерований в Grome

Програма Bryce (рис 1.3) використовує різні фрактальні алгоритми для створення місцевості і також підтримує візуалізацію за допомогою трасування променів.

Рисунок 1.3 – Скеля, згенерована в Bryce

Якщо завдання полягає в генерації і візуалізації процедурно згенерованого ландшафту в реальному часі, можна використати програму World Creator (рис. 1.4), оскільки всі обчислення відбуваються на графічному процесорі (англ. Graphics Processing Unit, GPU).

Рисунок 1.4 – Ландшафт, згенерований в World Creator

Для моделювання ерозії можна використати програму Houdini 17 SideFX (рис. 1.5). Houdini 17 SideFX використовує науково обґрунтовані алгоритми для моделювання ерозії, тому результати є надзвичайно реалістичними.

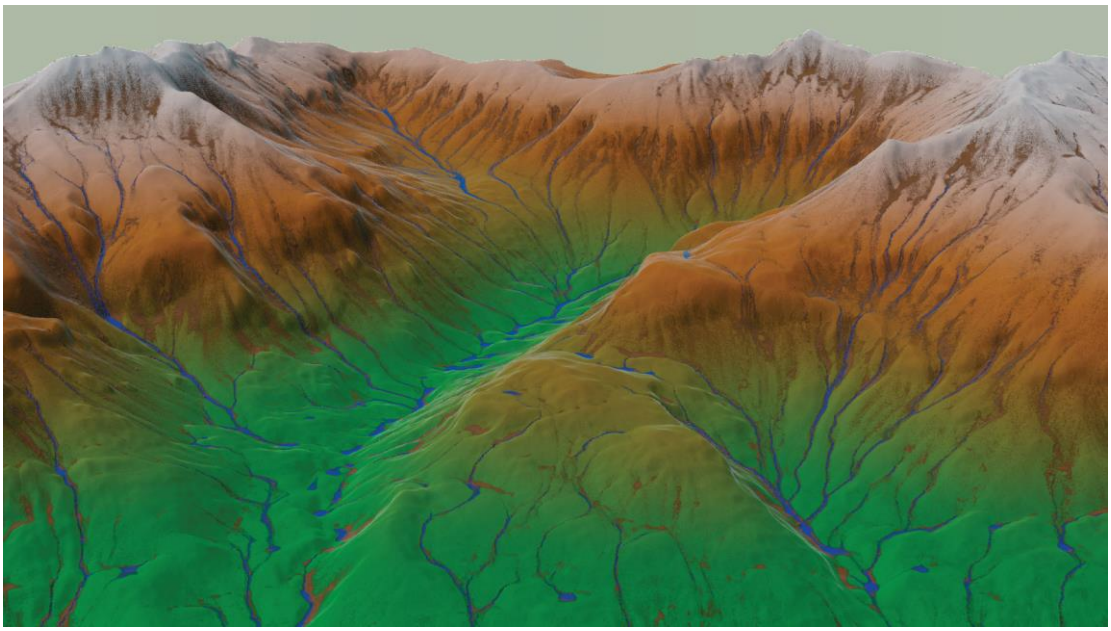


Рисунок 1.5 – Ландшафт, згенерований в Houdini

Основним недоліком використання таких програмних засобів є можливість змінювати параметри ландшафту лише в редакторі, що займає тривалий час і не завжди є зручним.

Альтернативою використання готових програмних засобів, є генерація процедурного ландшафту за допомогою різних алгоритмів. Даний спосіб є більш складним, оскільки потрібно розуміти, яким саме чином працюють алгоритми процедурної генерації. Перевагою такого методу є відсутність додаткових програмних засобів і можливість реалізувати специфічні для проекту можливості.

1.2. Аналіз способів представлення ландшафту

Існує декілька способів представлення процедурно згенерованого ландшафту, кожен з яких має свої переваги і недоліки. Основними вимогами представлення процедурно згенерованого ландшафту є:

- можливість модифікувати дані;
- ефективне використання пам’яті;
- швидкий доступ до даних;
- відсутність артефактів під час візуалізації.

Одним із способів представлення процедурно згенерованого ландшафту є полігональна сітка (рис. 1.6). Полігональна сітка є універсальним способом процедурно згенерованого ландшафту. Вона здатна описати процедурно згенерований ландшафт будь-якої складності у вигляді трикутників.

Полігональні сітки зазвичай використовують для опису статичного ландшафту, оскільки динамічно модифікувати такий ландшафт не дуже практично. Ще одним недоліком є відсутність підтримки рівнів деталізації. Щоб обійти це обмеження, доводиться робити декілька версій одного й того ж ландшафту з різною деталізацією, що негативно впливає на обсяг необхідної пам’яті.

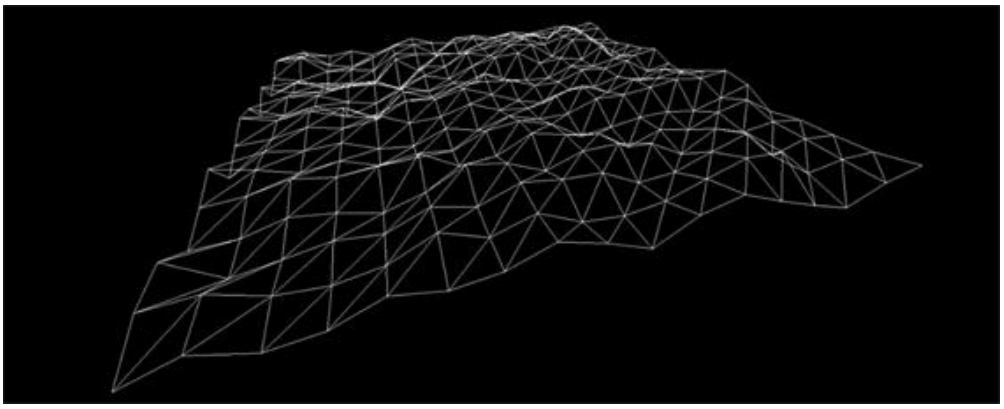


Рисунок 1.6 – Полігональна сітка

Ще одним способом опису процедурно згенерованого ландшафту є трикутна нерегулярна сітка (рис. 1.7). Трикутна нерегулярна сітка потребує значно меншого обсягу пам’яті ніж звичайна полігональна сітка, оскільки поверхня ландшафту описується не окремими трикутниками, а унікальними вершинами і зв’язками між ними. Трикутну нерегулярну сітку можна отримати із полігональної сітки, об’єднавши спільні вершини.

Недоліком такого підходу є складність оптимізації полігональної сітки в трикутну нерегулярну сітку і необхідність значних ресурсів. Тому такий спосіб опису не підходить в ігрових програмах реального часу, де ландшафт можна змінювати динамічно.

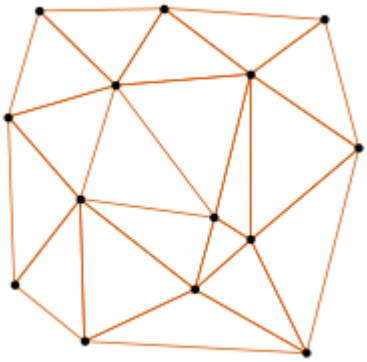


Рисунок 1.7 – Нерегулярна сітка

Для зручності обробки і модифікації процедурно згенерованого ландшафту, потрібно максимально спростити спосіб представлення даних. Одним із таких спрощень, може бути використання карти висот (рис. 1.8).

Карта висот зберігається у вигляді двовимірного масиву, кожний елемент якого описує значення висоти у відповідних позиціях (рис. 1.9).

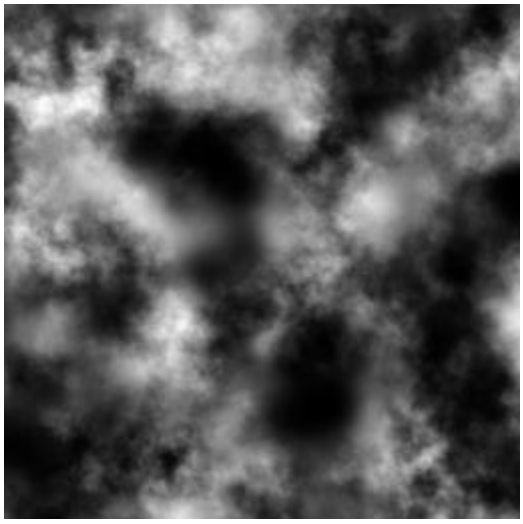


Рисунок 1.8 – Карта висот

Використання карти висот потребує значно меншого обсягу пам’яті ніж трикутні сітки, оскільки зберігає лише значення висоти для кожної позиції. Карти висот дозволяють дуже легко змінювати геометрію ландшафту, але важливим недоліком є відсутність підтримки печер.

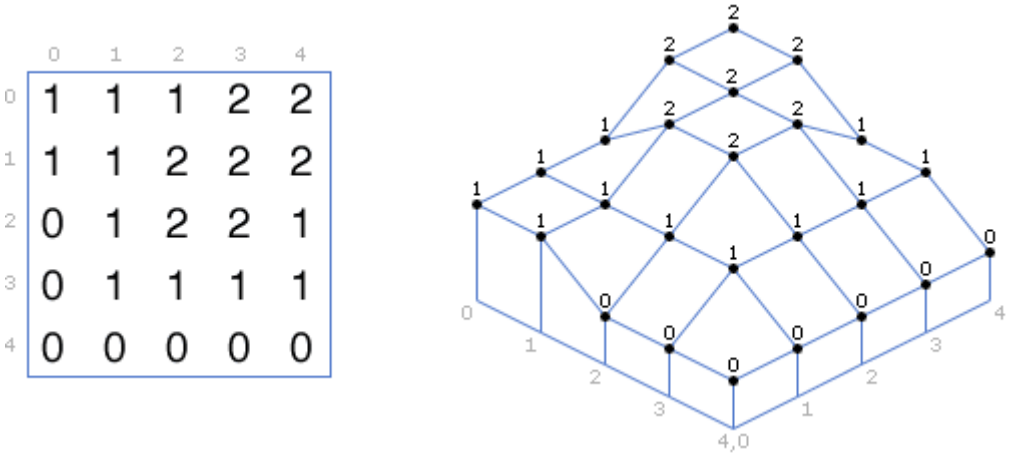


Рисунок 1.9 – Двовимірний масив і отриманий ландшафт

Для покращення швидкодії обробки і візуалізації процедурно згенерованих ландшафтів, можна розділити масив даних на окремі частини (рис. 1.10). Завдяки такому розподілу, можна динамічно завантажувати та вивантажувати потрібні частини процедурно згенерованого ландшафту. Це дозволить оперувати з окремими частинами процедурно згенерованого ландшафту без необхідності тримати в пам’яті весь обсяг даних.

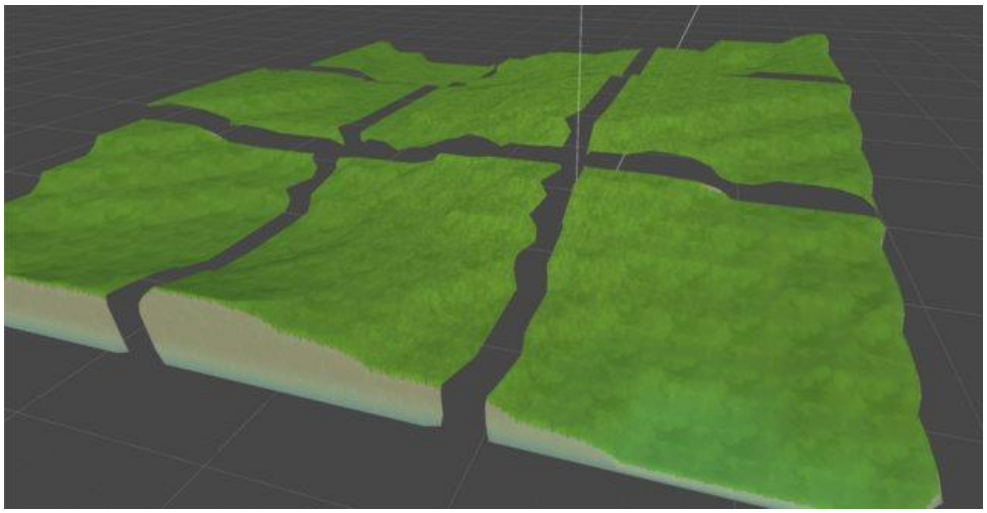


Рисунок 1.10 – Окремі частини згенерованого ландшафту

1.3. Аналіз алгоритмів генерації карти висот

Найпростішим способом заповнення карти висот є генерація псевдовипадкових чисел (рис. 1.11). Основною проблемою використання псевдовипадкових чисел є генерація ландшафту дуже низької якості.

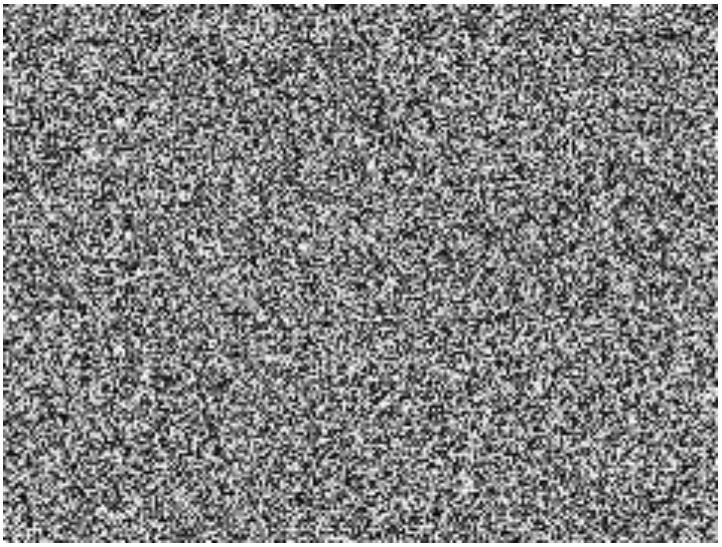


Рисунок 1.11 – Білий шум

Для покращення отриманого результату, можна використовувати різні функції згладжування, але результат все одно буде не дуже втішним.

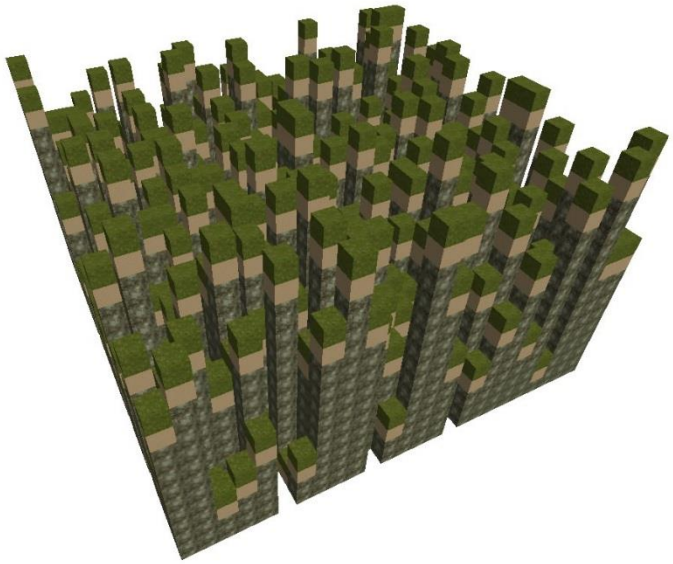


Рисунок 1.12 – Ландшафт, згенерований за допомогою білого шуму

Для генерації карти висот можна також використовувати шум Перлина (рис. 1.13) [6]. Його особливість полягає в тому, що згенеровані точки рівномірно згладжуються градієнтом. Звичайно шум Перлина реалізується як дво-, три-, або чотиривимірна функція, але його можна використовувати для довільної кількості вимірів. Однак складність обрахунків подвоюється на кожному наступному вимірі.

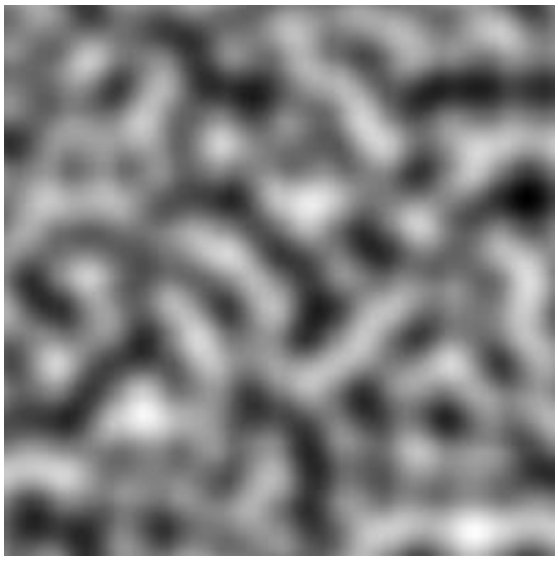


Рисунок 1.13 – Шум Перлина

Реалізація складається із трьох кроків: визначення сітки, обчислення скалярного добутку градієнтних векторів, інтерполяція отриманих результатів (рис. 1.14).

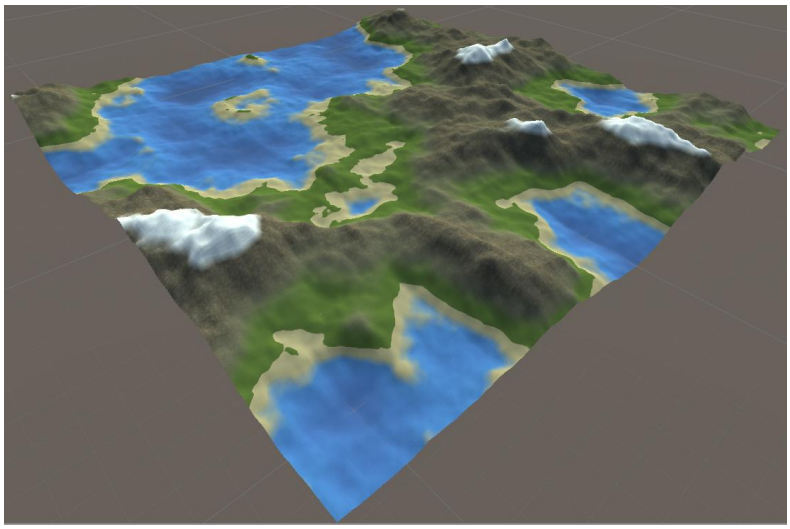


Рисунок 1.14 – Ландшафт, згенерований за допомогою шуму Перлина

Альтернативою шуму Перлина є симплекс-шум (рис. 1.15). Він дозволяє вирішити проблему із зростанням складності обчислень, а також має менше артефактів у згенерованому шумі.

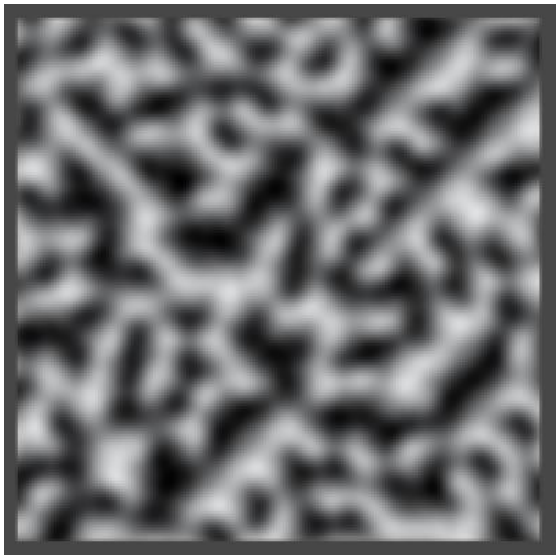


Рисунок 1.15 – Симплекс-шум

Переваги симплекс-шуму над шумом Перлина:

- Має меншу обчислювальну складність і вимагає менше операцій множення;
- За більшої кількості вимірів є продуктивнішим, складність — $O(n^2)$ для n вимірів, в той час як у класичному шуму — $O(2^n)$ [11];
- Не має помітних артефактів.

- Має добре визначений безперервний градієнт, який обчислюється доволі швидко;
- Легко реалізувати на машинному рівні.

Реалізація складається з чотирьох кроків: обрахунок вершин стільникового гіперкуба, поділ на симплекси, вибір градієнта, сумування ядер.

Недоліком симплекс-шуму є те, що згенерований шум на різних рівнях відрізняється. Наприклад, двовимірний шум виглядає інакше, ніж переріз тривимірного шуму. Вигляд погіршується зі збільшенням кількості вимірів.

Реалізація тривимірного і вище симплекс-шуму захищений патентом, якщо алгоритм реалізований з використанням методик, описаних у будь-якому з пунктів патенту.

Ще одним способом генерації карти висот є розбиття метричного простору, що визначається відстанями до заданої дискретної множини ізольованих точок цього простору (рис. 1.16).

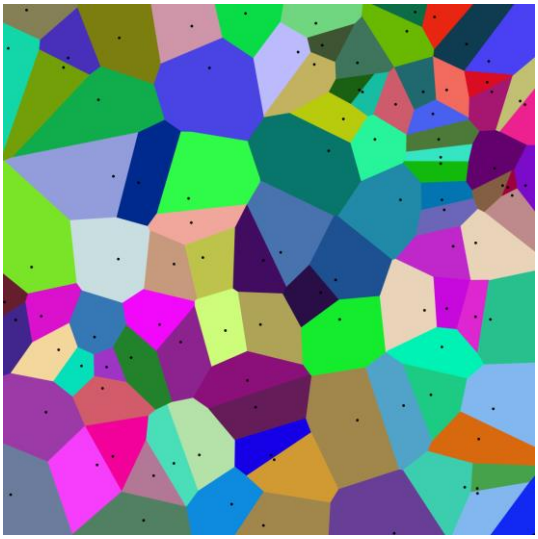


Рисунок 1.16 – Діаграма Вороного

Існує декілька алгоритмів побудови діаграми Вороного:

- Прямий;
- Рекурсивний;
- Алгоритм Форчуна;
- Шляхом перетину напівплощин.

Задача обчислення діаграми Вороного зводиться до проблеми сортування дійсних чисел, тому найбільш оптимальним є алгоритм Форчуна.

Алгоритм Форчуна заснований на застосуванні замітаючої прямої. Замітаюча пряма являє собою вертикальну пряму лінію, яка рухається зліва направо (рис. 1. 17).

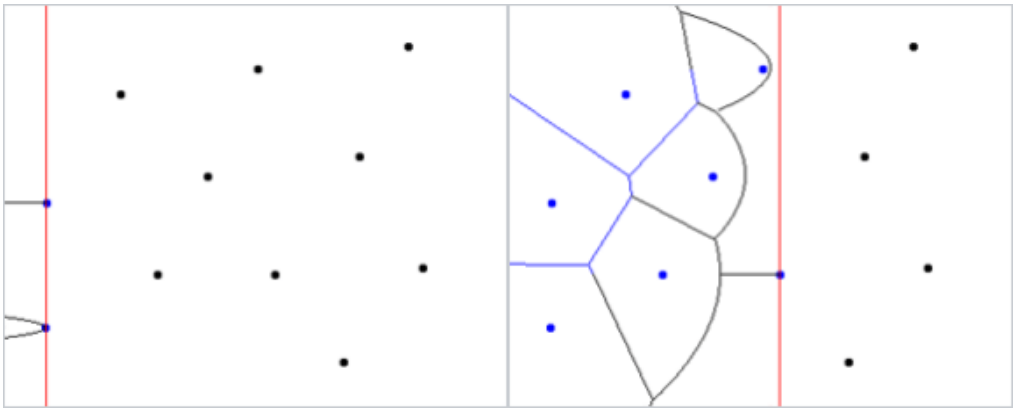


Рисунок 1.17 – Побудова діаграми Вороного за алгоритмом Форчуна

На кожному кроці алгоритму діаграма Вороного побудована для множини, що складається з замітаючої прямої та точок ліворуч від неї. При цьому межа між замітаючою прямою та областями точок складається з відрізків парабол.

Щоразу, коли замітаюча пряма проходить через чергову точку, ця точка додається до вже побудованої ділянки діаграми (рис. 1.18). Обчислювальна складність алгоритму Форчуна дорівнює O(nlog n).

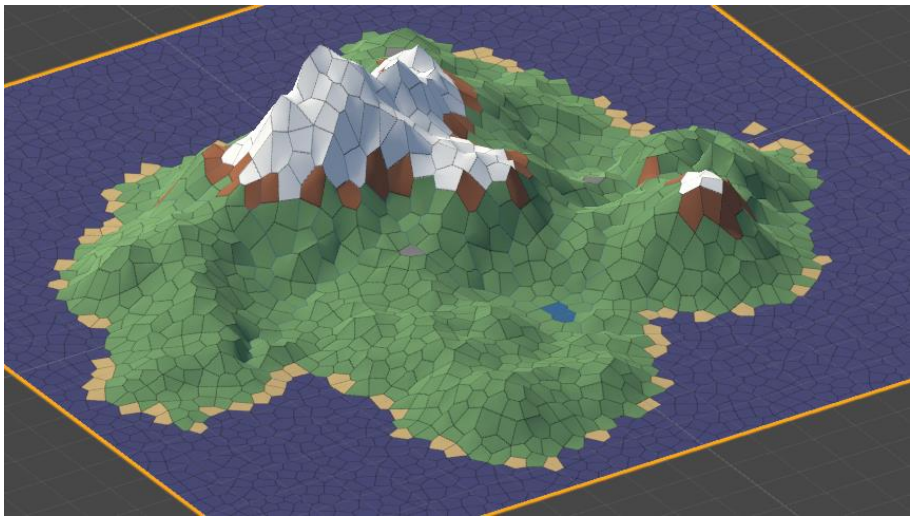


Рисунок 1.18 –Ландшафт, згенерований за допомогою діаграми Вороного

Для генерації фрактальних ландшафтів можна використати алгоритм Diamond Square (рис. 1.19). Він заснований на одновимірному алгоритмі midpoint displacement. Цей алгоритм найкраще підходить для генерації реалістичних ландшафтів.

Суть алгоритму полягає в тому, що спочатку встановлюється початкова висота у чотирьох кутових точках масиву.

Далі для кожного масиву знаходиться середня точка, в яку записується середнє значення чотирьох кутових точок, зміщене на випадкову величину.

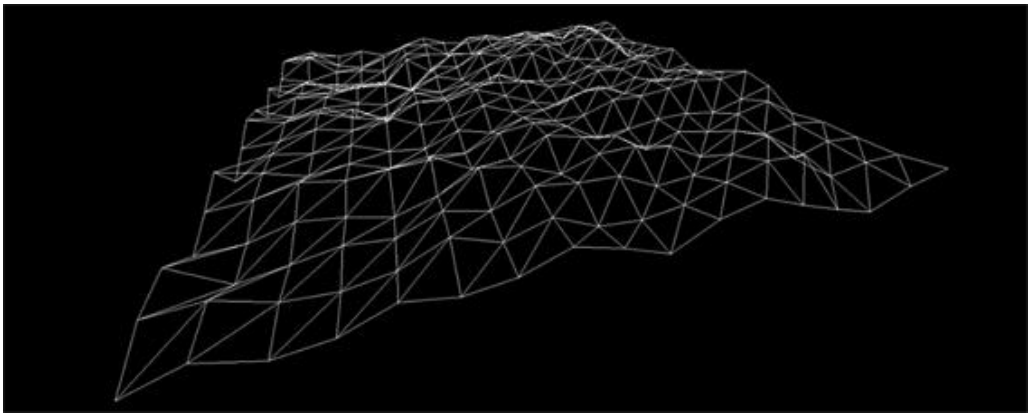


Рисунок 1.19 – Ландшафт, створений за допомогою Diamond-Square

Далі для кожного ромбу в масиві знаходиться середня точка, в яку записується середнє значення чотирьох кутових точок, зміщене на випадкову величину. На кожній ітерації випадкове значення зменшується, кількість ітерацій залежить від бажаного рівня деталізації (рис. 1.20).

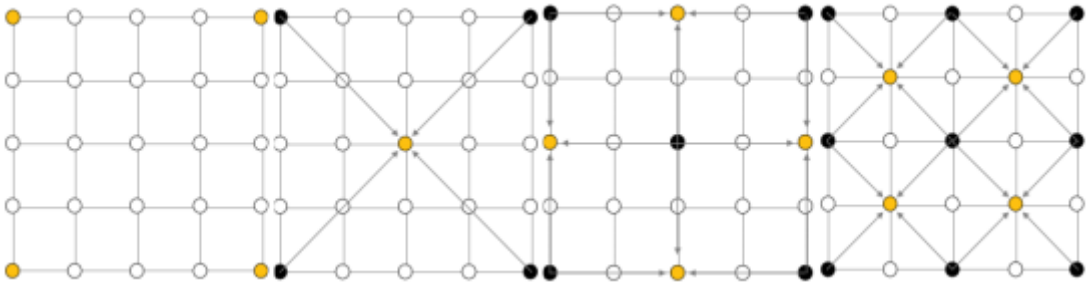


Рисунок 1.20 – Візуалізація алгоритму Diamond-Square

Недоліком такого алгоритму є те, що він визначений лише в початково заданих межах, а тому не підходить для створення нескінченних ландшафтів.

Для реалістичної деформації ландшафту можна використати алгоритм

для моделювання ерозії. Ерозія — один з головних зовнішніх чинників формування рельєфу земної поверхні. Розрізняють схилову й руслову водну ерозію. В результаті водної ерозії утворюються яри, балки, річкові долини (рис. 1.21).

Алгоритм працює таким чином, що спочатку генерується карта висот, використовуючи довільний алгоритм. Далі, в залежності від кількості ітерацій, моделюється зовнішній вплив на ландшафт.

При великій кількості ітерацій досягається реалістичне відображення згенерованого ландшафту.

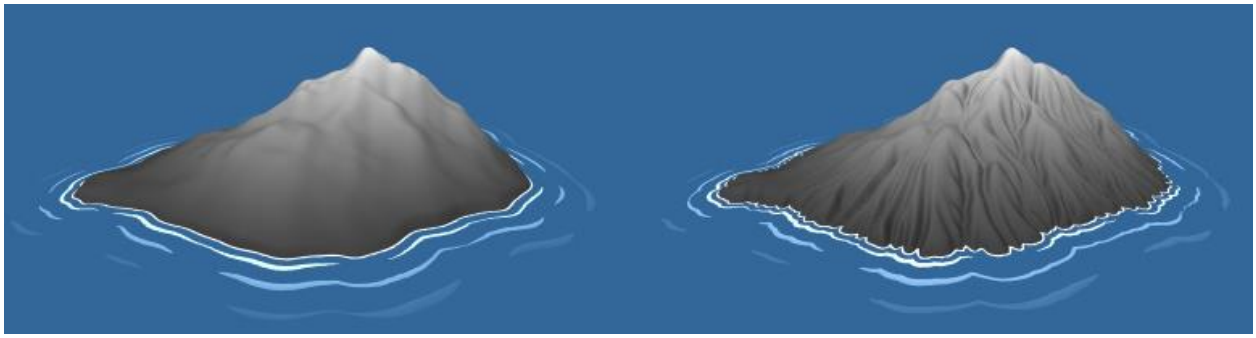


Рисунок 1.21 – Згенерований ландшафт до і після ерозії

1.4. Приклади використання процедурної генерації

Одним із прикладів використання процедурної генерації є гра No Man’s Sky (рис. 1.22). No Man’s Sky – це гра про дослідження всесвіту в нескінченному процедурно-згенерованому всесвіті.

Рисунок 1.22 – Постер гри No Man’s Sky

Окрім ландшафту, алгоритми процедурної генерації використовуються для генерації ігрових об’єктів на основі обумовлених наборів правил. Розробники No Man’s Sky створили систему шаблонів, яка дозволяє спростити алгоритми процедурної генерації. Художники і моделери вручну створюють об’єкти, а потім використовуються алгоритми для урізноманітнення (рис. 1.23).

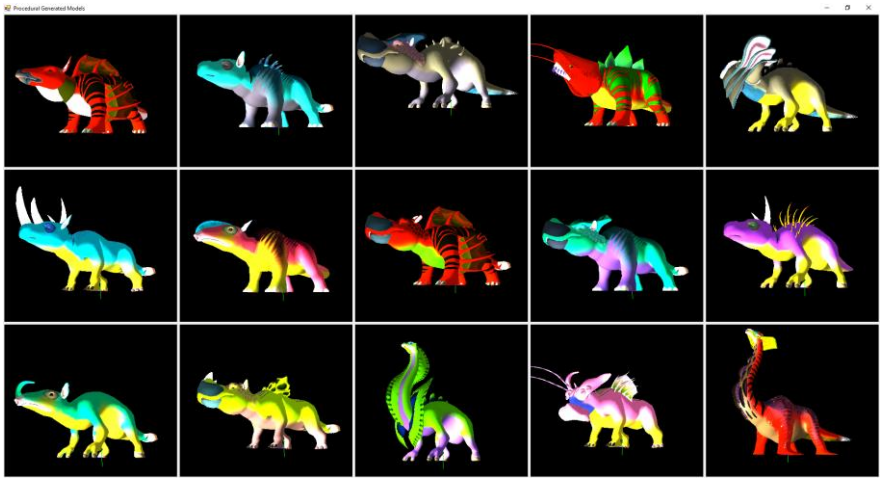


Рисунок 1.23 – Процедурна генерація на основі моделі трицератопса

За аналогією з No Man's Sky, в RymdResa генерується всесвіт. Космічний артхаус RymdResa створює за допомогою алгоритму процедурної генерації безкраї космічні простори тільки для того, щоб підкреслити самотність головного героя.

Рисунок 1.24 – Гемплей RymdResa

Ще одним прикладом використання процедурної генерації є гра The Binding of Isaac: Rebirth в жанрі action-adventure (рис. 1.24).

Рисунок 1.24 – Постер гри The Binding of Isaac: Rebirth

В даній грі, алгоритми процедурної генерації використовуються для створення ігрових кімнат (рис. 1.25), предметів, які гравець може підібрати, ворогів тощо.

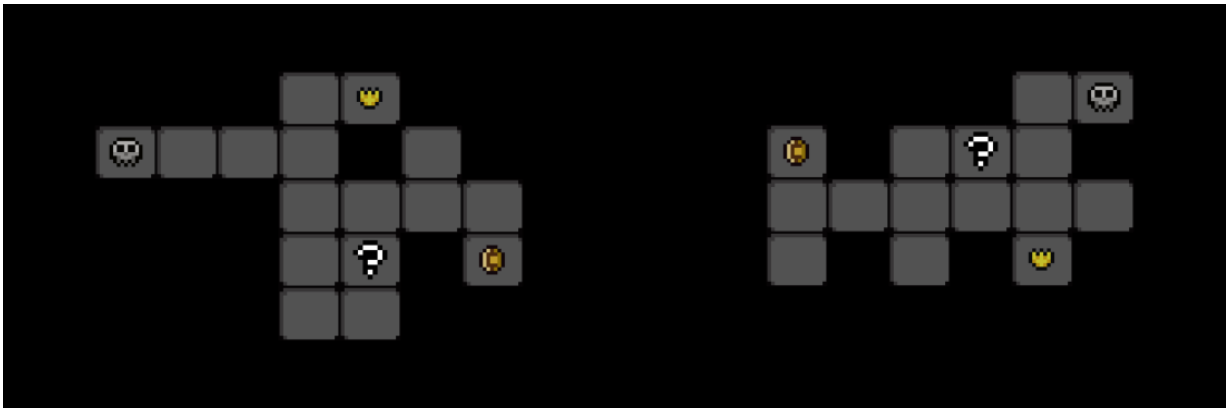


Рисунок 1.25 – Процедурно згенеровані ігрові кімнати

Алгоритми процедурної генерації використовуються для створення ігрового світу із різноманітністю і динамікою. Тоді у гравця буде виникати ілюзія нескінченної ігри.

Ще одна гра з елементами rouguelike, яка використовує алгоритми процедурної генерації, – це Dwarf Fortress (рис. 1.26). Dwarf Fortress відрізняється від інших продуктів свого жанру не складністю концепції, а тим, як окремі прості елементи гри нашаровуються один на одного. Завдяки

цьому кожна ігрова сесія відкриває для користувача унікальний, багатовимірний і абсолютно непередбачуваний процедурно згенерований світ гномів.

Генерація світу в Dwarf Fortress відбувається в декілька етапів. Спочатку вибирається полюс (наприклад, захід або схід). Далі генератор випадкових чисел визначає базові значення полів карти (висота, опади, температура, вулканічна активність), потім програма заповнює їх за допомогою фрактальних алгоритмів.

Далі, в залежності від висоти і температури, генерується рослинність, визначаються кліматичні зони, згладжується згенерована карта.

Далі, застосовуючи алгоритми ерозії, генеруються річки. Маленькі океани висушуються, далі визначаються висоти гір, в яких можна створити початок річок. Занадто великі висоти згладжуються, щоб вся карта не перетворилася в каньйони.

Рисунок 1.26 – Поселення гномів

В точках, де перетинаються багато річок, утворюються озера. На основі згенерованих даних відбувається коригування опадів, температури. Далі відбувається фінальна генерація рослинності, враховуючи оновлені дані. Додаються підземні шари і проводиться верифікація згенерованого світу. Потім для кожного регіону визначається початкова популяція тваринного

світу і визначаються параметри погоди.

Після генерації світу, на карті генеруються поселення гномів і за допомогою симуляції визначаються взаємовідносини між гномами, родовід, історія подій. Це доволі великий обсяг роботи, який потребує значних обчислювальних можливостей. Проте дозволяє згенерувати унікальну сюжетну історію для кожного гравця.

Ще одним прикладом процедурної генерації є гра Minecraft. Minecraft надає можливість гравцю досліджувати процедурно згенерований світ.

Рисунок 1.27 – Постер гри Minecraft

Генерація світу в Minecraft відбувається в декілька етапів. Спочатку визначаються кліматичні зони. Далі генератор випадкових чисел визначає значення температури і опадів, враховуючи кліматичні зони світу.

Враховуючи наявні дані, генератор визначає позиції майбутніх структур (поселень, замків, скарбниць тощо).

Далі генератор випадкових чисел визначає значення висот, і, використовуючи наявні дані, створюється початковий світ (рис. 1.28).

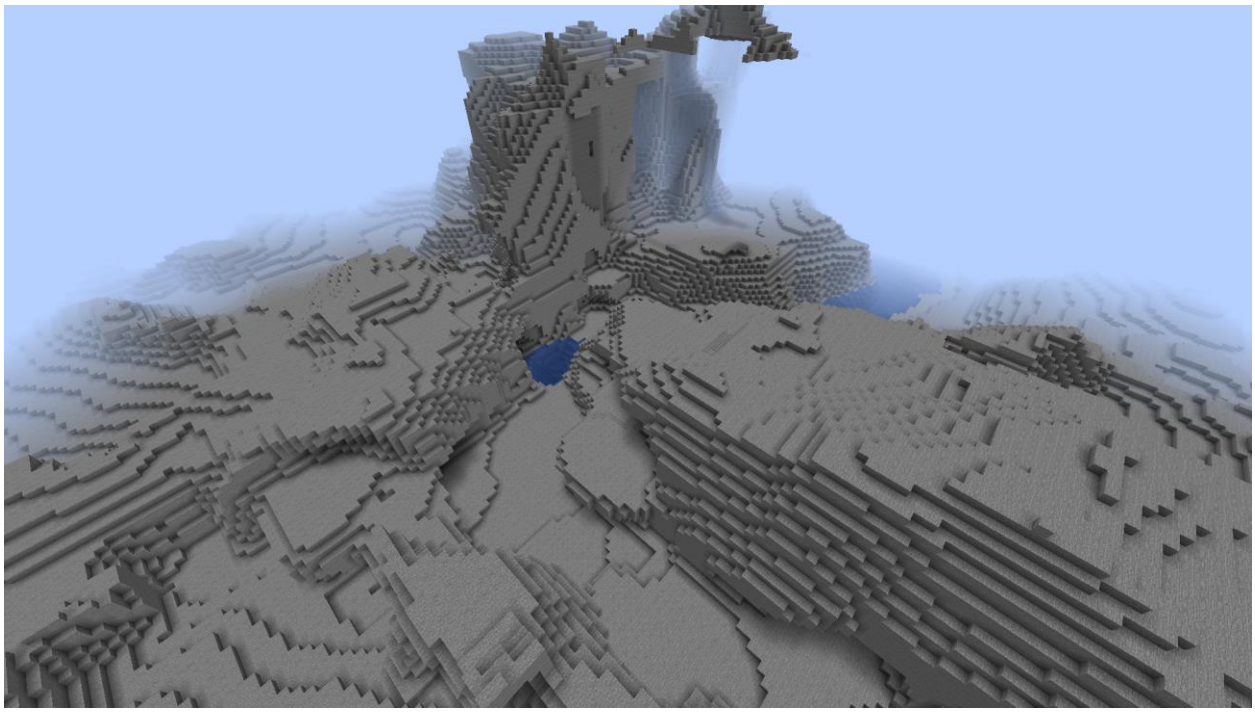


Рисунок 1.28 – Початкова генерація світу

Потім, в залежності від висоти, опадів, температури і кліматичних зон, замінюється верхній шар світу (рис. 1.29).

Рисунок 1.29 – Замінений верхній шар світу

Далі генератор випадкових чисел визначає маску, по якій видаляються певні вокелі, або замінюються на інші (рис. 1.30), генеруються печери.

Рисунок 1.30 – Згенеровані печери

Далі відбувається генерація рослинності і структур на основі позицій, які були визначенні раніше.

Рисунок 1.31 – Згенерована рослинність і поселення

Потім для кожного регіону визначається початкова популяція тваринного світу.

Процедурна генерація часто використовується у фільмах для створення

навколишнього середовища. Це дозволить зменшити бюджет фільму, або використати його для вдосконалення візуальних ефектів.

Висновки до розділу 1

Проаналізувавши засоби і алгоритми процедурної генерації, можна виділити основні проблеми алгоритмів процедурної генерації:

- Неможливо забезпечити необхідний контроль якості згенерованого контенту;
- Алгоритми процедурної генерації можуть бути складними при розробці;
- Процес процедурної генерації може негативно впливати на швидкодію гри;
- Складно керувати результатом, необхідно вносити зміни в алгоритми процедурної генерації;
- Необхідно забезпечити достатній рівень різноманітності згенерованого контенту. Низький рівень різноманітності створює відчуття повторюваності.

Цих проблем можна уникнути при правильному плануванні розробки, розумінні роботи алгоритмів процедурної генерації.

Важливу роль у створенні процедурного ландшафту відіграють програмні засоби, які дозволять редагувати декорації, елементи місцевості, річки, озера, траву і інші елементи декору.

Більшість програмних засобів для процедурної генерації ландшафтів створюють діапазон висот для опису базової місцевості, використовуючи при цьому псевдовипадкові значення у вигляді шуму, фракталів, та інших алгоритмів.

Для побудови карти висот можна використовувати такі алгоритми:

- псевдовипадкові значення;
- шум Перліна;
- симплекс-шум;
- моделювання ерозії;
- алгоритм Diamond Square;

- діаграма Вороного.

Але такий спосіб опису має декілька проблем, зокрема не можна описувати скелі, печери.

Виявлення цих обмежень визначило необхідність розробки принципово нових способів процедурної генерації ландшафтів.

2. ВОКСЕЛЬНА ТЕХНОЛОГІЯ

2.1. Способи візуалізації воксельних даних

На сьогодні основним методом візуалізації тривимірної геометрії є полігони. Технології на основі полігонів мають багато проблем: складність обчислень, обчислення колізій, складність модифікації тривимірної геометрії тощо [1].

Альтернативою полігонам є вокселі. Воксель є аналогом пікселя у тривимірному просторі (рис. 2.1). Використання вокселів може вирішити проблеми, з якими стикаються технології на основі полігонів. Вокселі є об’ємними за своєю природою, тому не потребують додаткової інформації для моделювання внутрішньої частини об’єктів.

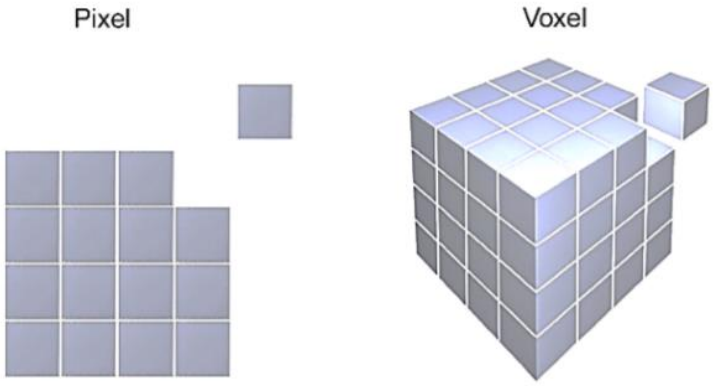


Рисунок 2.1 – Піксель і воксель

Існує багато алгоритмів для відображення воксельних даних, серед яких:

- Marching cubes;
- Marching tetrahedra;
- Surface Net;
- Dual Contouring;
- Raymarching.

Особливістю Marching cubes алгоритму є те, що для створення геометрії існує 256 можливих конфігурацій, з яких унікальними є лише 15 (рис. 2.2), а всі інші конфігурації можна отримати за допомогою повороту і

відображення [7].

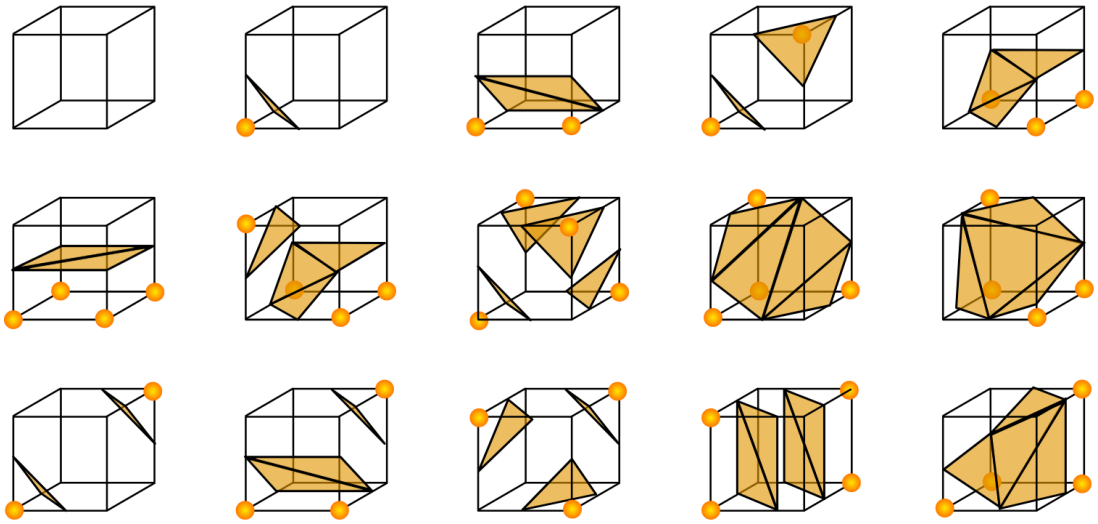


Рисунок 2.2 – 15 унікальних конфігурацій куба

Marching cubes є простим в реалізації, проте має багато проблем:

- Складність. Необхідно розглядати багато різних випадків;
- Невизначеність. Не узгоджена послідовність вибору конфігурації може створювати пустоти в геометрії (рис. 2.3).
- Marching cubes не може створювати гострі кути. Адаптивність теж не допоможе, Marching cubes завжди створює прямі відрізки всередиді будь-якого елементу (рис. 2.4).

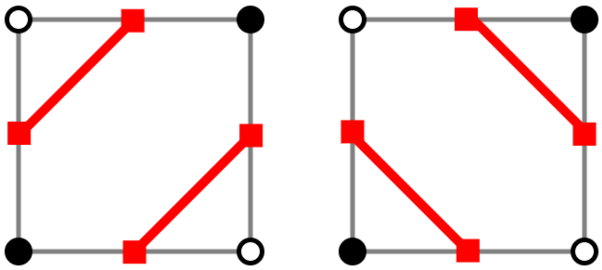


Рисунок 2.3 – Невизначеність конфігурації куба

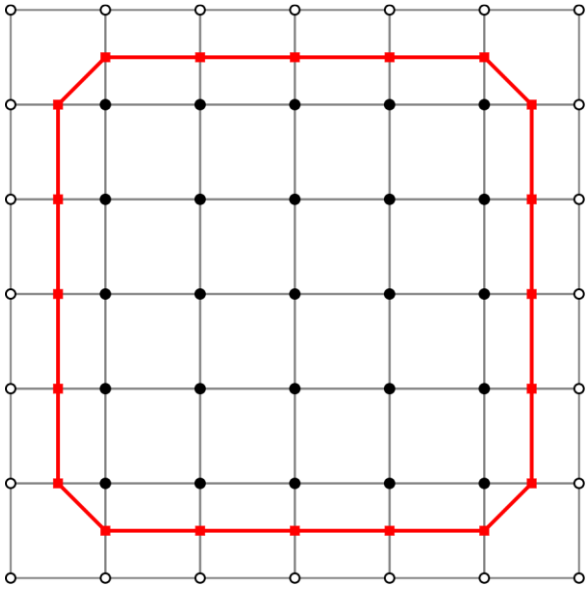


Рисунок 2.4 – Апроксимований квадрат з допомогою Marching cubes

Альтернативою Marching cubes є алгоритм Marching tetrahedra. Він працює із тетраедрами, а не з кубами [8], а тому кількість можливих конфігурацій в нього лише 8 (рис. 2.5).

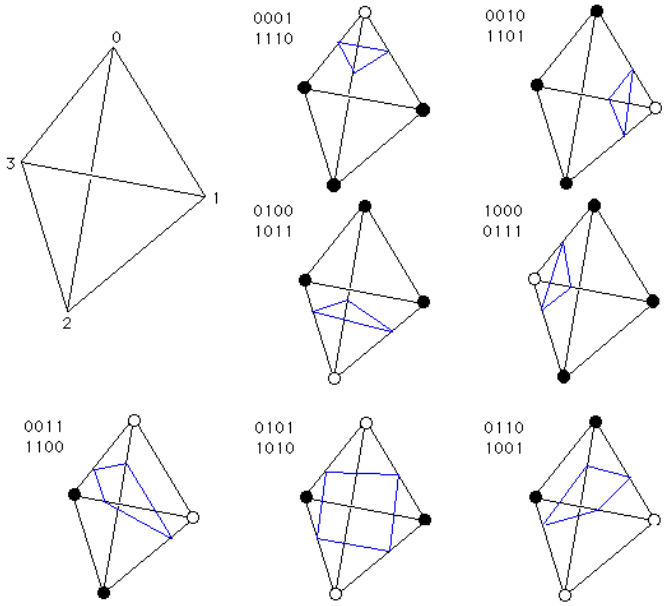


Рисунок 2.5 – 8 можливих комбінацій конфігурацій тетраедра

Порівнянно з Marhing cubes, Marching tetrahedra має декілька переваг:

- Алгоритм може працювати як на неструктурованих сітках, так і на структурованих сітках, що робить Marching tetrahedra універсальним рішенням для створення поверхні на всіх типах сітки.

- Дозволяє уникнути певних неоднозначних випадків в алгоритмі Marching cubes.

Хоча підхід Marching tetrahedra може подолати неоднозначність, через різні розподіли куба на тетраедри можуть утворюватися різні грані поверхні. Зокрема, є дві схеми поділу куба на п’ять тетраедрів, і кожна схема призводить до різної поверхні [9].

Для того, щоб отримувати більш згладжені поверхні порівняно із кубічними підходами, можна використати алгоритм Surface Net. Він є простим в реалізації, але його недоліком є значна обчислювальна складність.

Для створення більш природних форм, можна використати алгоритм Dual Contouring, який розширює можливості Surface Net і Marching cubes [10].

Його недоліком є необхідність мати більше інформації про поверхню, а це суттєво збільшує використання пам’яті. Однак, збільшення інформації про поверхню дозволяє покращити адаптивність порівняно із Marching Cubes. Ще одна проблема Dual Contouring полягає в тому, що можуть створюватися поверхні, що перетинаються. Хоча поверхня, що отримана завдяки Dual Contouring, є герметичною, вона може бути не завжди добре описаною. Через це можуть виникати проблеми в окремих алгоритмах текстурування. Проблеми виникають, коли декілька об’єктів майже торкаються один одного.

Одним із найцікавіших способів відображення воксельної графіки є Raymarching. Даний алгоритм дозволяє отримувати згладжені поверхні будь-якої форми у реальному часі, відображати фото-реалістичні сцени та об’єкти.

Він є швидким і простим в реалізації. Існують три основні способи візуалізації за допомогою Raymarching:

- Raymarching plane intersections;
- Sphere assisted raymarching;
- Cube assisted raymarching.

.

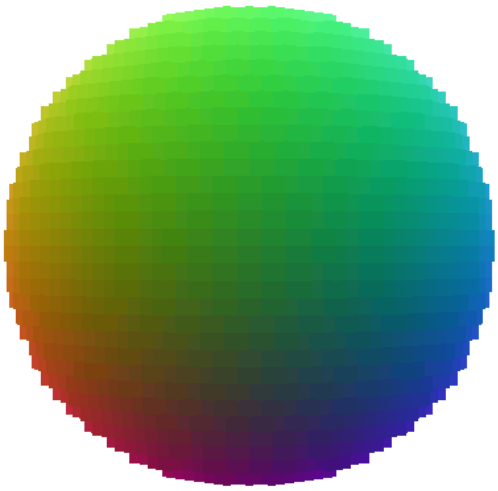


Рисунок 2.6 – Воксельна сфера

Перший спосіб – це Raymarching plane intersections. Набір вокселів NxNxN можна розділити, використовуючи площини N + 1 вздовж кожної вісі. Основна ідея Raymarching plane intersections це взяти промінь для кожного фрагмента і пройтись вздовж кожного перетину площини, поки не буде знайдено заповнений воксель.

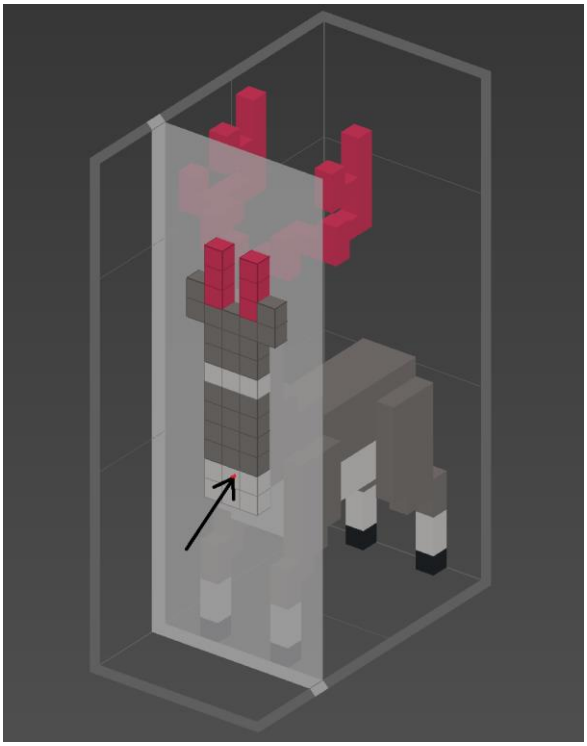


Рисунок 2.7 – Перетин площини із вокселями

Sphere assisted raymarching зменшує кількість ітерацій для пошуку вокселів, хоча це не завжди дає виграш у продуктивності. Основна ідея – зберігати інформацію про фігуру окремо від інформації про колір.

Замість того, щоб зберігати інформацію про фігуру безпосередньо, кожен відсутній воксель зберігає радіус порожнього простору навколо себе – представляючи найбільшу сферу, яка могла б заповнити порожній простір, центрований поточним вокселем. Це дозволяє робити великі кроки через порожній простір.

Cube assisted raymarching подібний до попереднього способу, за винятком використання кубів замість сфер.

Недоліком використання Raymarching є те, що незначні похибки в обчисленнях можуть призводити до артефактів.

Відображення вокселів є не дуже простим завданням і є окремим предметом досліджень для пошуку найбільш ефективних алгоритмів візуалізації.

2.2. Способи опису воксельних даних

Використання вокселей має багато переваг, основними з яких є:

- Проста модифікація згенерованого ландшафту;
- Підтримка ландшафтів будь-якої складності;
- Безперевні тривимірні дані, які дозволяють описувати внутрішнє представлення.

Одним із способів представлення воксельних даних є тривимірна сітка. Перевагами використання тривимірної сітки для опису воксельних даних є:

- Швидкість доступу до кожного вокселя;
- Швидкість модифікації;
- Простота реалізації.

Але такий спосіб опису має декілька недоліків, зокрема низьку швидкодію візуалізації і значний обсяг пам’яті для опису воксельних даних.

Ще одним недоліком є відсутність підтримки рівнів деталізації, що унеможливлює використання воксельної технології для складної і деталізованої тривимірної геометрії.

Тому на заміну звичайній тривимірній сітці було запропоновано використовувати деревовидну структуру даних – октодерево [2].

Октодерево – дерево октанів, у якому вершина має вісім нащадків. Найчастіше вісімкові дерева використовуються для того, щоб рекурсивно поділити тривимірний простір на октанти (рис. 2.9).

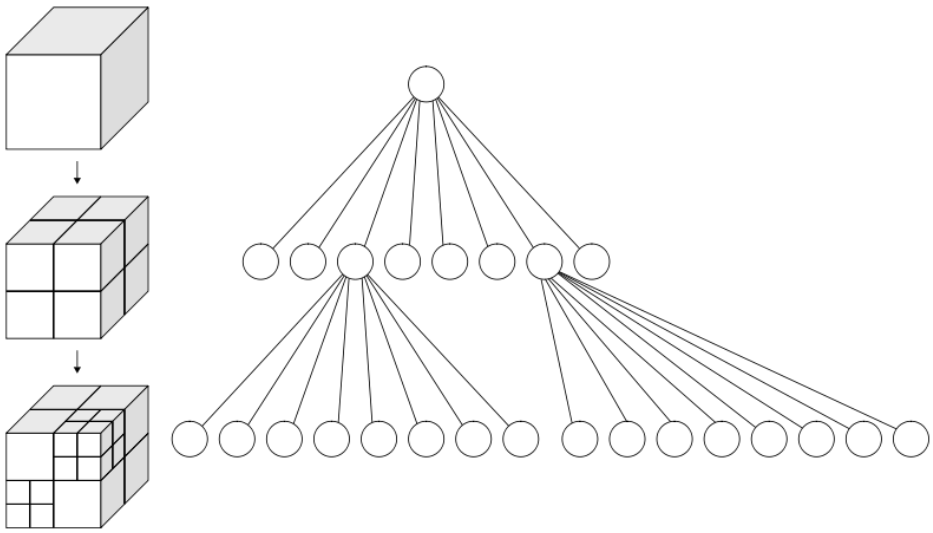


Рисунок 2.9 – Рекурсивний поділ куба на октанти. Справа: відповідне дерево октантів

Октодерева є тривимірним аналогом квадродерев (рис. 2.10).

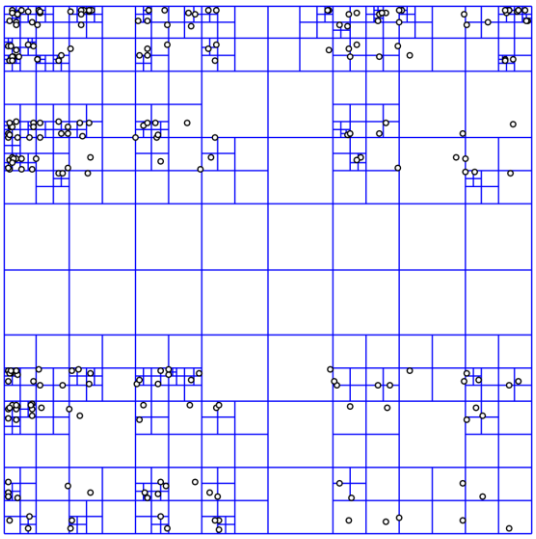


Рисунок 2.10 – Квадродерево

Кожен вузол в дереві октантів ділить простір на вісім нових октантів. У регіональній точці (англ. Point Region - PR) октодерева вузол зберігає явну тривимірну точку, яка є «центром» поділу простору для цього вузла. Дана точка визначає один з кутів кожного з восьми дочірніх просторів. В октодереві на основі матриці (MX-октодерево) точка поділу є неявним центром простору, яке представляє вузол (рис. 2.11).

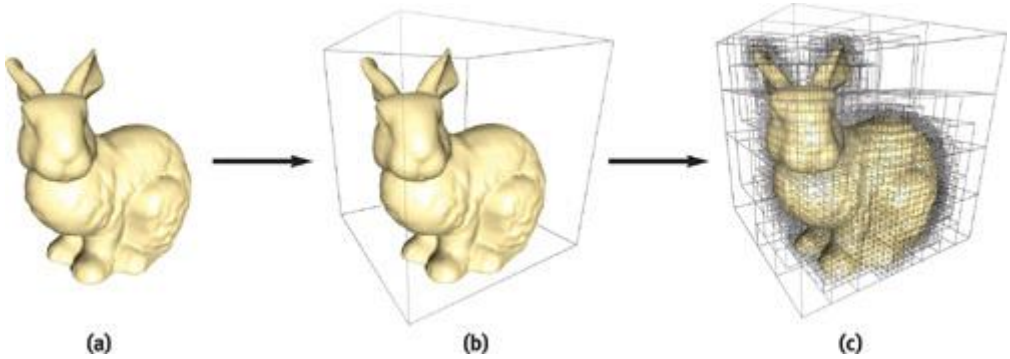


Рисунок 2.11 – Побудова воксельного октодерева

Кореневий вузол PR-октодерева може представляти нескінченний простір. Кореневий вузол MX-октодерева повинен представляти обмежену область простору, так щоб неявні центри були чітко визначеними.

Октодерева не можуть вважатися k-мірними деревами, оскільки k-мірні дерева поділяються уздовж розмірності, а октодерева поділяються навколо точки. Крім того, k-мірні дерева завжди є двійковими, що невірно для октодерев.

Вузли октодерева завжди поділяються по центру (рис. 2.12).

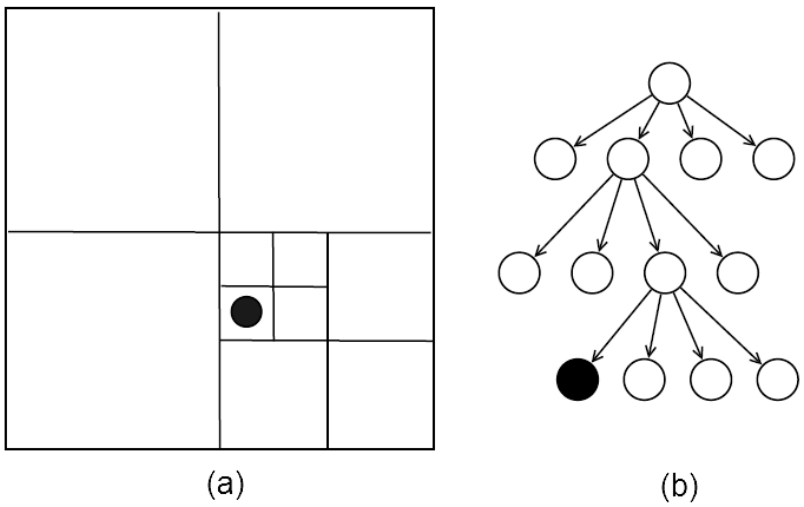


Рисунок 2.12 – Octree

Вузли Point-region octree не завжди поділяються по центру, що дозволяє використовувати елементи різного розміру (рис. 2.13).

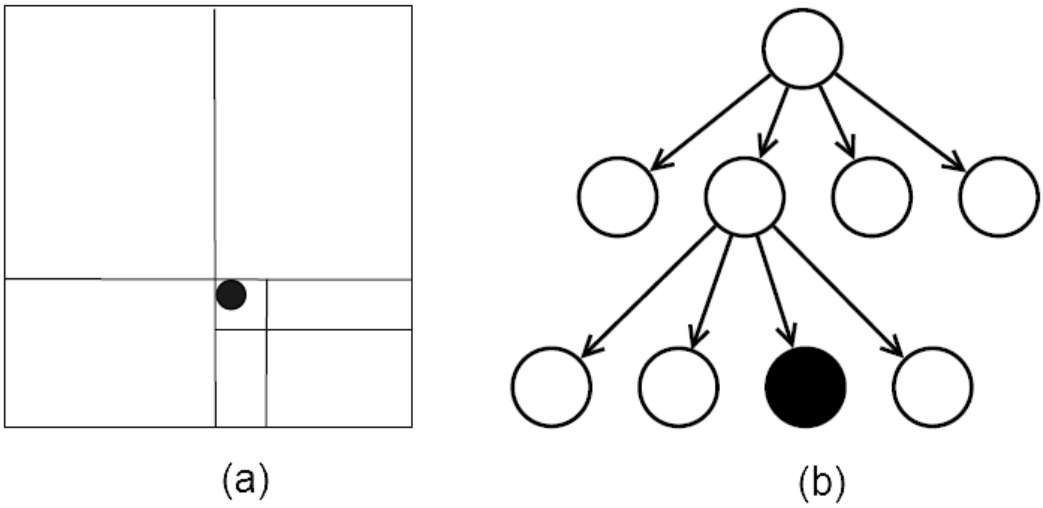


Рисунок 2.13 – Point-region octree

Kd-tree являє собою двійкове дерево (рис. 2.14), яке ділить обмежуючий об’єм на два підлеглих вузла по вісі згладжування. Кожен внутрішній вузол розбивається по певній вісі в заданому положенні. Оскільки в Kd-tree кожен вузол має лише два нащадки, він вимагає менше пам’яті ніж октодерева. Однак, Kd-tree вимагає більшої кількості вузлів, а тому обхід такого дерева може бути значно довшим.

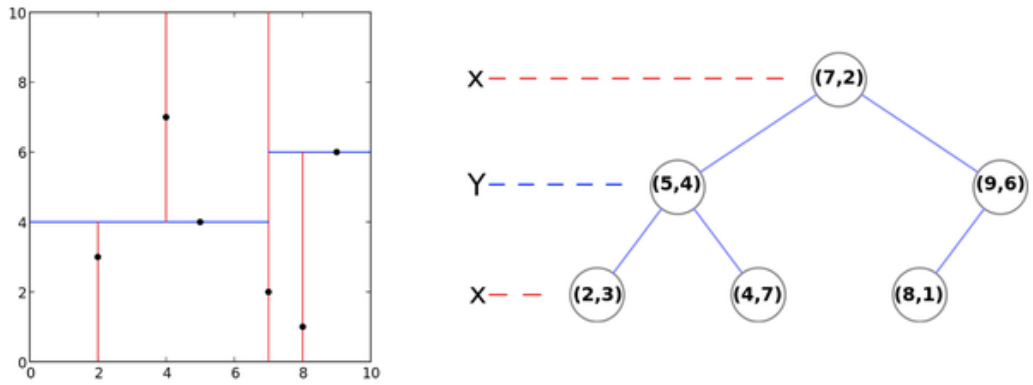


Рисунок 2.14 – Kd-tree

Особливість древовидних структур для опису тривимірної геометрії полягає в тому, що вони автоматично реалізують рівні деталізації. Рівень деталізації тривимірної геометрії визначається глибиною октодерева (рис. 2.15).

Рівень деталізації визначає складність тривимірної геометрії, яку потрібно візуалізувати.

Рисунок 2.15 – Рівні деталізації

Рівень деталізації, як метод, підвищує ефективність візуалізації шляхом зменшення навантаження на етапах графічного конвеєра. Зменшення якості тривимірної об’єкту звичайно залишається непоміченим, якщо об’єкти знаходиться на великій відстані або швидко рухаються.

Існують два підходи для реалізації підтримки рівнів деталізації. Перший з них заснований на розбитті простору на кінцеву кількість областей (рис. 2.16) з визначеним рівнем деталізації для кожної області. Результатом буде дискретна кількість рівнів деталізації.

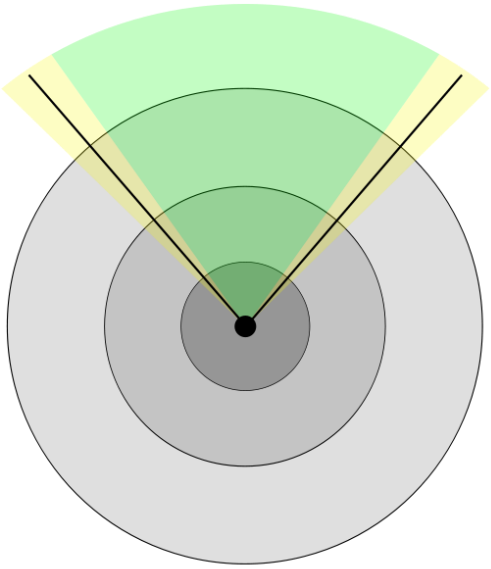


Рисунок 2.16 – Діапазони дискретних рівнів деталізації

Основна концепція дискретного рівня деталізації полягає в тому, щоб для представлення одного й того ж об’єкту мати різні моделі (табл. 2.12). Отримання таких моделей вимагає зовнішнього алгоритму, який звичайно є

нетривіальним та використовує різні методи спрощення.

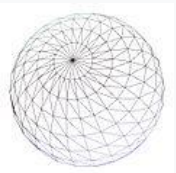
Дискретний рівень деталізації найчастіше використовується у високопродуктивних ігрових додатках, які використовують не великі набори даних.

Іншим підходом є ієрархічний рівень деталізації. Оскільки апаратні засоби орієнтовані на великі обсяги даних, візуалізація великої кількості тривимірних об’єктів низької якості може бути дуже не оптимальною. Ієрархічний рівень деталізації уникає цієї проблеми завдяки групуванню різних об’єктів, це забезпечує більшу ефективність.

Підтримка рівнів деталізації є дуже важливим для ігрової програми, тому використання древовидних структур даних є необхідним.

Проте октодерева мають один дуже суттєвий недолік. Октодерево може мати багато вузлів, які не містять інформації про вокселі. Тому такий спосіб опису тривимірної геометрії потребує достатньо багато пам’яті.

Таблиця 2.1 – Рівні деталізації сфери

Порівняння та вимірювання візуального впливу					
Зображення					
Кількість вершин	~5500	~2880	~1580	~670	140
Коментар	Максимальна деталізація.	Поступове зменшення деталізації.			Мінімальна деталізація.

Для вирішення цієї проблеми можна використати розріджене воксельне

октодерево [3]. Особливістю розрідженого воксельного октодерева є відсутність вузлів, які не містять інформації про вокселі в певній ділянці простору (рис. 2.17). Це дозволяє зменшити обсяг необхідної пам’яті і більш ефективно візуалізувати воксельні дані.

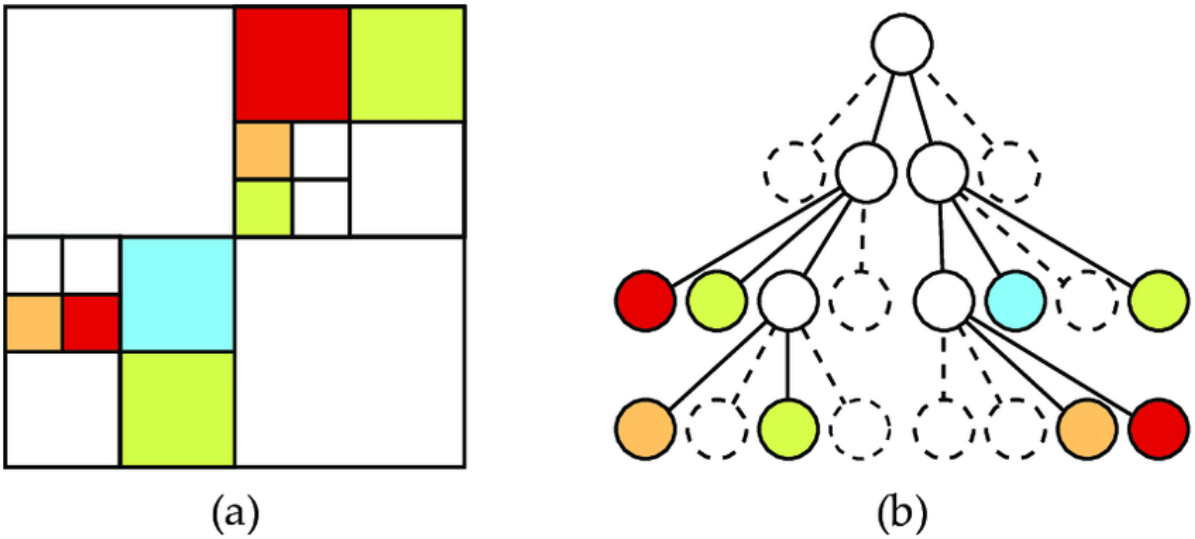


Рисунок 2.17 – Розріджене воксельне октодерево

Зазвичай, будь-яка тривимірна геометрія буде мати багато однакових елементів. Якщо представити ці дані у вигляді розрідженого воксельного октодерева, можна помітити, що таке дерево, буде мати багато однакових вузлів (рис. 2.14).

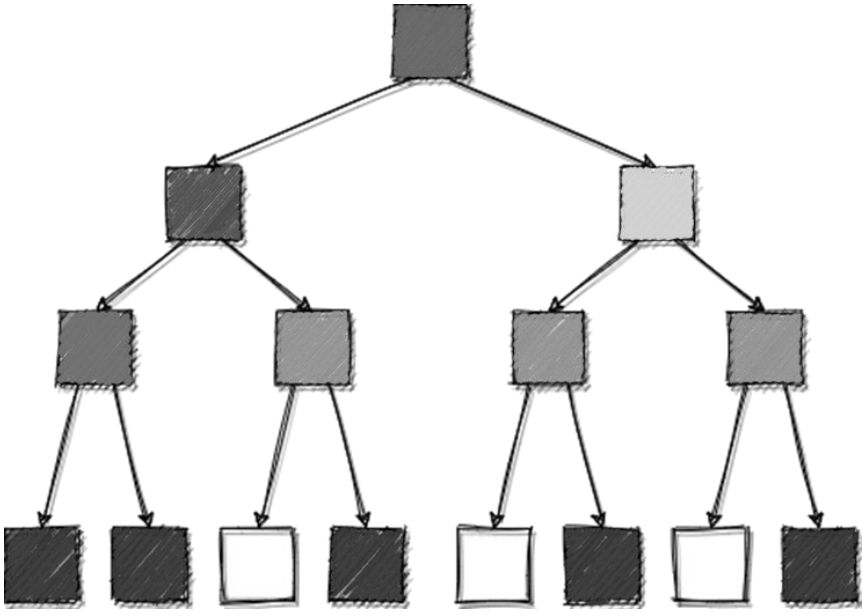


Рисунок 2.18 – Розріджене воксельне октодерево, з однаковими вузлами.

Тому, представлення тривимірного об’єкту у вигляді розрідженого воксельного октодерева не завжди є оптимальним.

Висновки до розділу 2

Воксельна технологія задає новий рівень вимог, яким традиційні технології на основі полігонів вже не здатні задовольнити.

Ключовим є вибір структури опису воксельних даних і способу візуалізації цих даних.

Проаналізувавши різні алгоритми візуалізації, можна зробити висновок, шо кожен з них підходить для вирішення конкретного завдання, а універсального алгоритму не існує. Пошук ефективних методів візуалізації воксельних даних є предметом постійних досліджень.

Аналогічно із способом опису воксельних даних. Для невеликих тривимірних об’єктів потрібно використовувати звичайну тривимірну сітку, оскільки час побудови октодерев може нівелювати результат від зменшення кількості вузлів.

Якщо це тривимірна геометрія, яка майже повністю заповнює заданий простір, потрібно використовувати звичайне октодерево, оскільки побудова розрідженого воксельного октодерева може не дати ніякого покращення.

Якщо тривимірний об’єкт має багато порожніх ділянок простору, потрібно використовувати розріджене воксельне октодерево. Це допоможе суттєво зменшити кількість вузлів в октодереві і відповідно зменшити обсяг потрібної пам’яті.

Якщо тривимірний об’єкт має багато однакових елементів, тоді використання розрідженого воксельного октодерева буде неефективним, оскільки багато вузлів міститимуть однакову інформацію про вокселі.

Звідси виникає потреба в розробці ефективних алгоритмів опису воксельних даних.

В якості методу візуалізації було обрано алгоритм Marching cubes, оскільки він простий в реалізації та ідеально підходить для прототипу.

В якості способу опису воксельних даних обрано розріджене воксельне октодерево із запропонованою оптимізацією, що дозволить зменшити

кількість вузлів в октодереві.

3. РЕАЛІЗАЦІЯ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ЛАНДШАФТУ

3.1. Процедурна генерація ландшафту

Для процедурної генерації ландшафту було обрано алгоритм шуму Перлина, оскільки він є простим в реалізації і підходить для обрахунку тривимірного шуму.

Для обрахунку значення шуму Перлина в довільній позиції спочатку потрібно комірку, в якій вона розміщена (рис. 3.1).

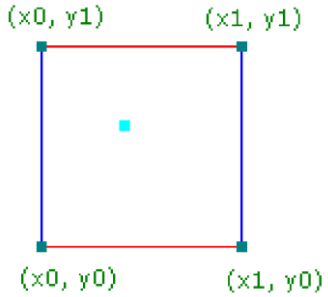


Рисунок 3.1 – Визначена комірка за заданої точки

Далі, для кожного вузла комірки визначається випадковий напрямок градієнту (рис. 3.2). Щоб зменшити кількість обчислень, можна використовувати вектори з центру комірки до ребер.

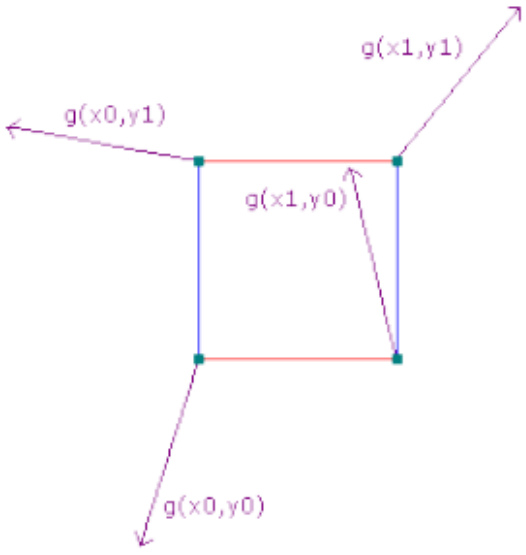


Рисунок 3.2 – Напрямок градієнту в вузлах

Для кожного вузла потрібно визначити вектор між заданою точкою і вузлами (рис. 3.3).

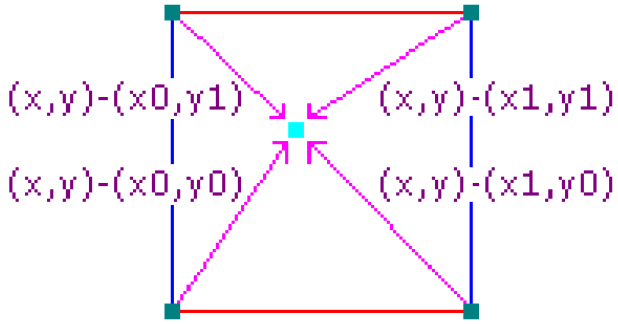


Рисунок 3.3 – Вектори від вузлів до заданої точки

Тоді для кожного вузла визначається скалярний добуток вектора і градієнту (рис. 3.4).

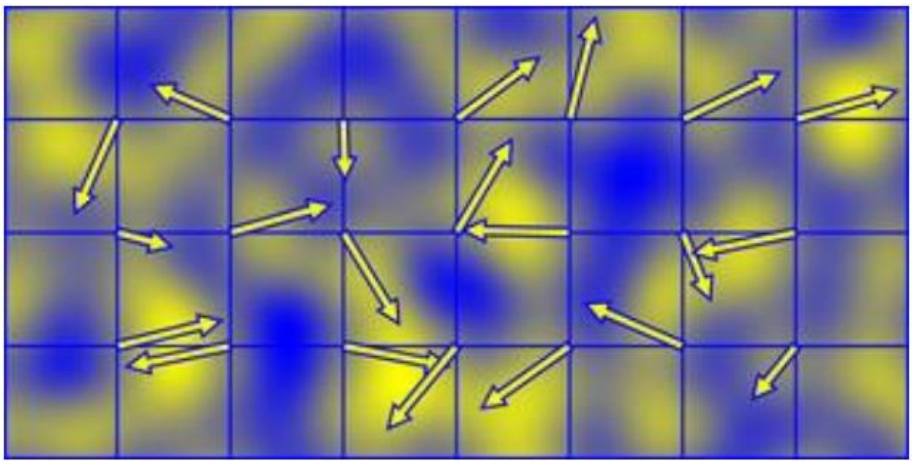


Рисунок 3.4 – Вплив градієнтів на шум Перлина

Далі потрібно виконати інтерполяції скалярних добутків, обчислених для кожного вузла за формулою:

$$f(x) = a_0(1 - x) + a_1 x$$

Це лінійна функція для кінцевих точок зі значеннями a_0 і a_1 і вагою інтерполяції x.

Для визначення ваги інтерполяції можна використати поліноміальну криву другого порядку, або різницю між вузлом із заданою точкою.

Для покращення згенерованого шуму, можна використати фрактальні алгоритми, наприклад фрактальний броунівський рух (англ. Fractional Brownian Motion). Фрактальний броунівський рух являє собою узагальнення броунівського руху. Броунівський рух - це рух, при якому положення об'єкта

з плином часу змінюється з випадковими інкрементами.

На відміну від класичного броунівського руху, інкременти не є повністю випадковими, вони зберігають тенденцію попередніх значень. Тому, зміна знечень інкрементів буде плавнішою, ніж при звачайному броунівському руху.

Фрактальний броунівський рух - це схожий процес, коли інкременти не повністю незалежні від одного, а в цьому процесі існує якась пам'ять. Якщо пам'ять має позитивну кореляцію, зміни в заданому напрямку будуть мати тенденцію до майбутніх змін у тому, що зроблено, і путь при цьому буде плавнішим, що зазвичай при ВМ.

Такий спосіб дозволяє підвищити правдоподібність згенерованого ландшафту (рис. 3.5).

Рисунок 3.5 – Згенерований ландшафт з використанням фрактального броунівського руху

Для того, щоб мати можливість генерації безкінечного ландшафту, потрібно розбити його на менші частини, які описують 16x16x16 вокселів. Це дозволить зменшити кількість обчислень і відображати лише ті частини ландшафту, які гравець може побачити.

3.2. Представлення процедурно згенерованого ландшафту

Для представлення процедурно згенерованого ландшафту, було реалізавано відразу декілька способів:

- Тривимірний масив;
- Октодерево;
- Розріджене воксельне октодерево.

Це дозволить порівняти всі способи і оцінити скільки пам’яті використовує кожний спосіб (рис. 3.6).

Оскільки тривимірний масив, це звичайна сітка розмірністю 16x16x16, тоді кожний чанк буде використовувати (4096 * S) байт пам’яті, де S – розмір вокселя в байтах.

Якщо замість тривимірного масиву використовувати октодерево, тоді для кожного чанку потрібно (N * S) байт пам’яті, де N – кількість вузлів в октодереві, S – розмір вузла октодерева в байтах.

Максимальна кількість пам’яті яку може використовувати октодерево буде (8 * D * S) байт пам’яті, де D – глибина октодерева, S – розмір вузла октодерева в байтах.

Якщо замість октодерева використовувати розріджене воксельне октодерево, тоді для кожного чанку потрібно (N * S) байт пам’яті, де N – кількість вузлів в розрідженому воксельному октодереві, S – розмір вузла розрідженого воксельного октодерева в байтах. Кількість вузлів в розрідженому воксельному октодереві не є лінійною, оскільки кожний вузол може бути поділеним на різну кількість нащадків.

Максимальна кількість пам’яті яку може використовувати розріджене воксельне октодерево буде (8 * D * S) байт пам’яті, де D – глибина розрідженого воксельного октодерева, S – розмір вузла розрідженого воксельного октодерева в байтах.

Серед усіх способів представлення процедурно згенерованого ландшафту тривимірний масив використовує найбільше пам’яті (рис. 3.6).

Октодерево використовує менше пам’яті. Це зумовлено відсутністю

вузлів, які не містять інформацію про вокселі.

Розріджене воксельне октодерево потребує майже таку ж кількість пам’яті, як і звичайне октодерево. Це зумовлено тим, що згенерований ландшафт не має великого перепаду висот і тому кількість вузлів майже співпадає.

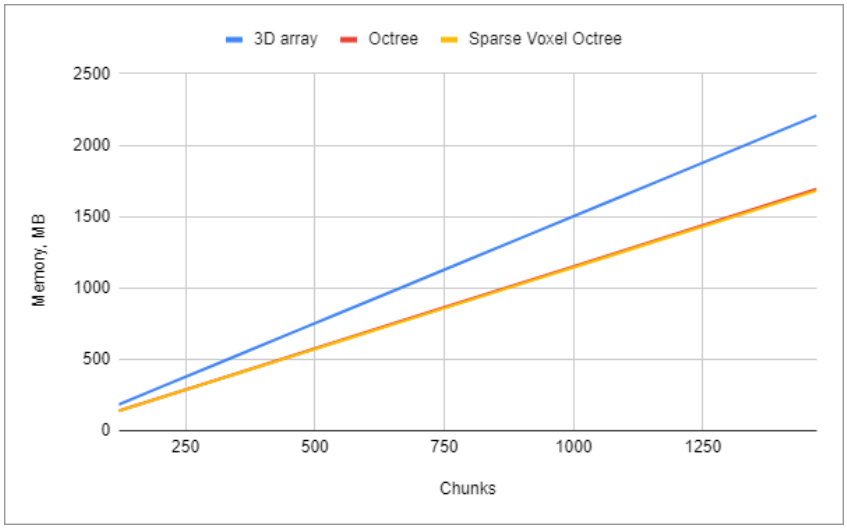


Рисунок 3.6 – Використання пам’яті

Основним способом опису воксельних даних було обрано розріджене воксельне октодерево для подальшої оптимізації.

3.3. Оптимізація розрідженого воксельного октодерева

Оптимізації розрідженого воксельного октодерева передбачає пошук і злиття у дереві загальних піддерев та унікальних вузлів (рис. 3.7). Для перетворення розрідженого воксельного октодерева (рис. 3.7a) в орієнтований ациклічний граф, обхід вузлів октодерева потрібно почати із найнижчого рівня.

Першим кроком перетворення розрідженого воксельного октодерева у спрямований ациклічних граф є злиття однакових вузлів на найнижчому рівні (рис. 3.7b). Переходячи до рівня вище, потрібно знайти всі вузли з однаковими дочірніми масками та однаковими нащадками. Такі вузли є коренями однакових піддерев, тому їх можна об’єднати (рис. 3.7c).

Для отримання кінцевого результату, операцію злиття слід

повторювати на кожному наступному рівні. Коли вичерпані можливості для злиття вузлів (рис. 3.7d), отримуємо кінцевий спрямований ациклічний граф.

Як результат виконання всіх кроків оптимізації воксельного октодерева, отримаємо орієнтований ациклічний граф з мінімальною кількістю вузлів (рис. 3.7d). Зменшення кількості вузлів відповідно зменшує обсяг пам’яті, який необхідний для опису чанка.

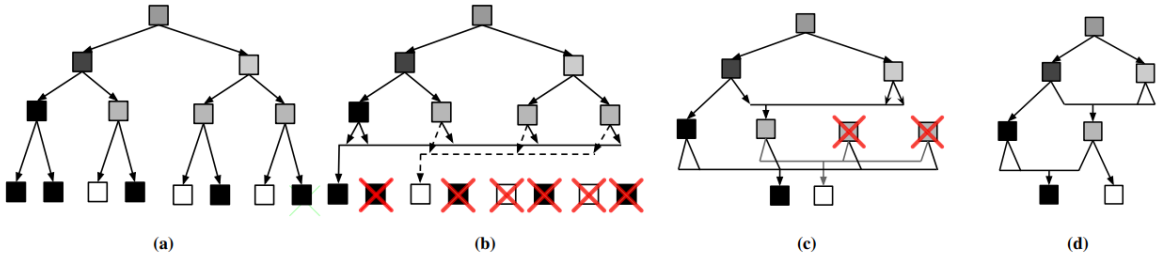


Рис. 3.7 – Оптимізація октодерева:

a) оригінальне дерево; b) видалення вузлів;

c) видалення вузлів на рівень вище; d) кінцевий граф.

Оптимізація розрідженого воксельного октодерева дозволила зменшити обсяг необхідної пам’яті (рис. 3.8).

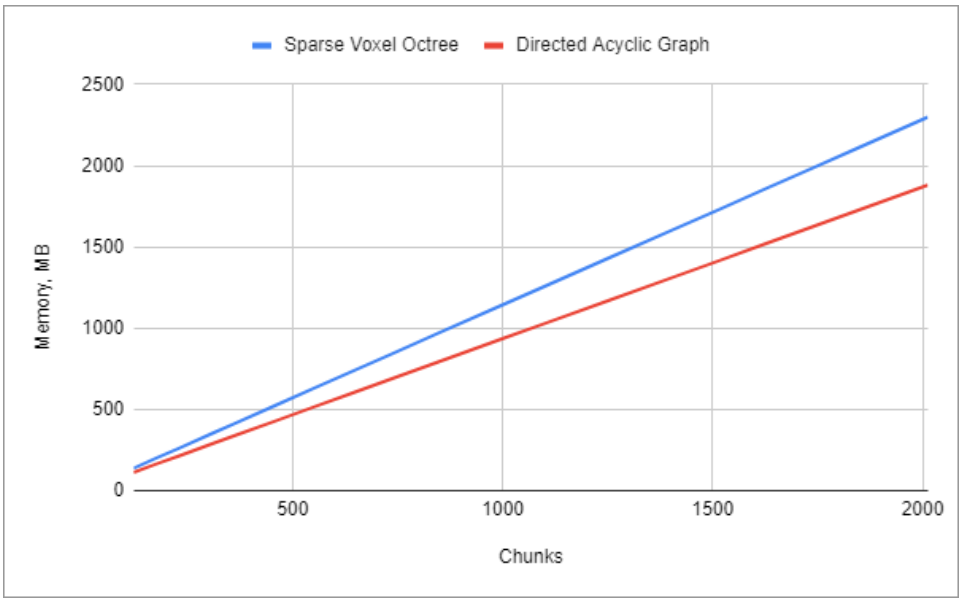


Рисунок 3.8 – Використання пам’яті

3.4. Візуалізація процедурно згенерованого ландшафту

Найбільш простим алгоритмом візуалізації воксельних даних є

Marching Cubes. Точніше, цей алгоритм дає змогу представити воксельні дані у вигляді полігональної сітки, оскільки сучасне обладнання для обробки графічних даних оптимізовано лише для візуалізації полігонів.

Алгоритм Marhing Cubes повністю детермінований, він дозволяє аналізувати вокселі і генерувати дані не залежно від інших вокселів. Завдяки цьому, алгоритм Marching Cubes дуже легко піддається оптимізації. Його можна виконувати з використанням декількох потоків або повністю виконувати на GPU.

Звичайно Marching Cubes і всі його похідні алгоритми працюють з даними, які описують щільність в кожній позиції простору (рис. 3.9). Оцінка щільності дозволяє підібрати необхідну топологію вершин, яка забезпечить нерозривність отриманої поверхні.

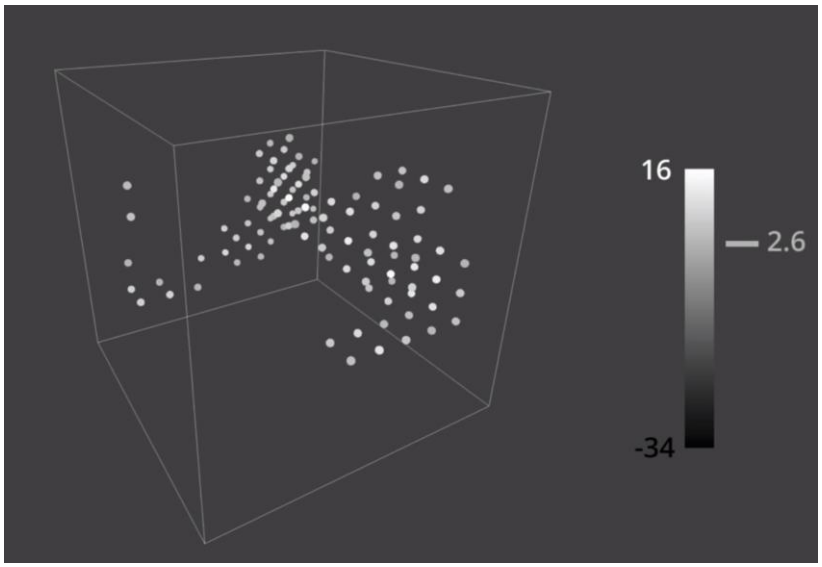


Рисунок 3.9 – Щільність в кожному вокселі

Обрання правильної топології ґрунтується на тому, які саме вершини повинні брати участь у формуванні поверхні (рис. 3.10). Таким чином визначається індекс попередньо згенерованої таблиці, що містить інформацію про відповідні вершини і полігони.

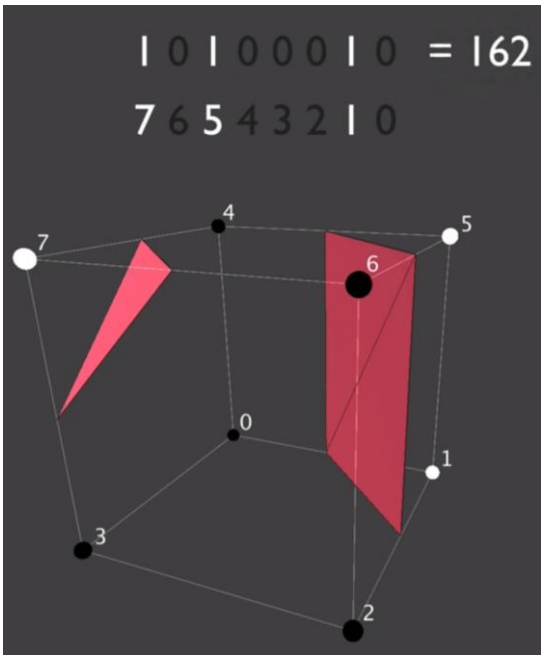


Рисунок 3.10 – Визначення топології

Можна помітити, що отримана поверхня за допомогою Marching Cubes не є згладженою (рис. 3.11). Для вирішення цієї проблеми потрібно використати алгоритм Dual Contouring.

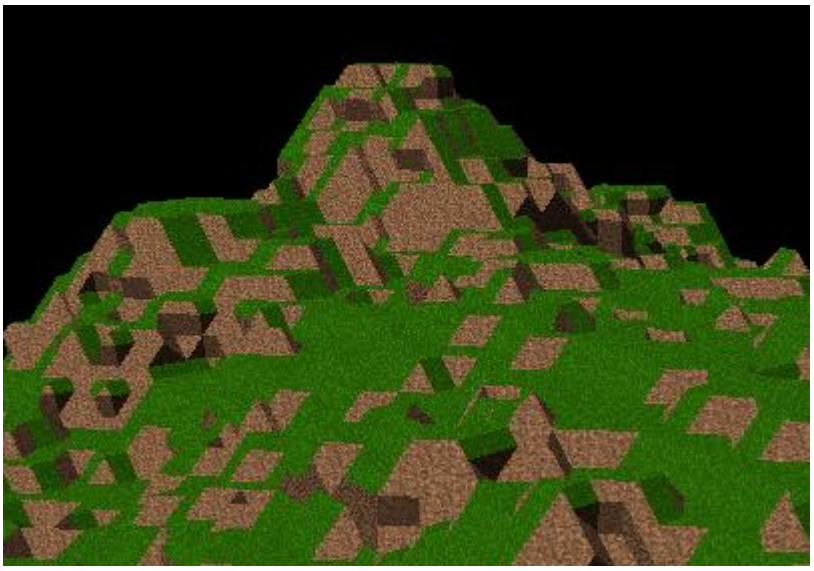


Рисунок 3.11 – Згенерований ландшафт

Для цього, потрібно лише врахувати значення щільності в сусідніх вокселях на етапі визначення позицій вершин (рис. 3.12).

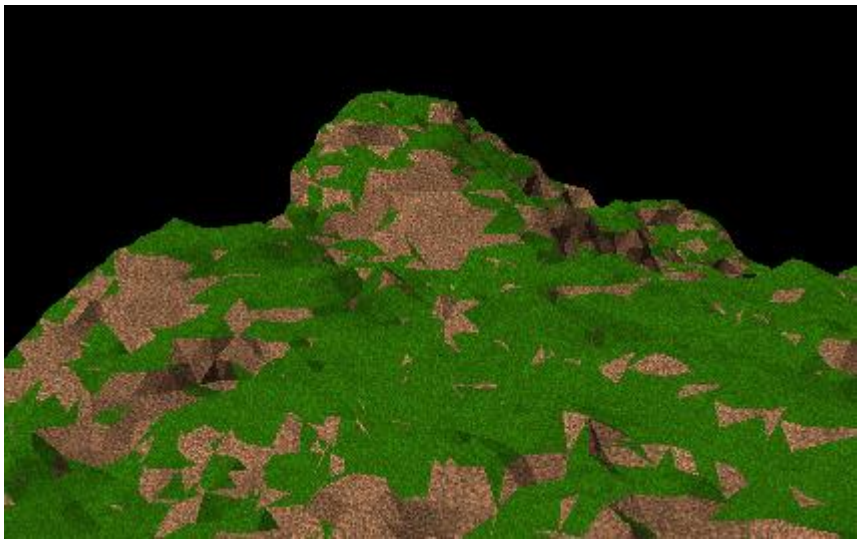


Рисунок 3.12 – Ландшафт після використання Dual Contouring

Висновки до розділу 3

Реалізація різних способів опису воксельних даних дала можливість порівняти їх і оцінити скільки пам’яті використовує кожен спосіб.

Найбільше пам’яті використовує тривимірний масив, оскільки він описує всі вокселі.

Октодерево і розріджене воксельне октодерево показали схожі результати. Це зумовлено тим, що згенерований ланшдшафт немає великих перепадів висот, тому кількість вузлів майже однакова.

Для подальшої оптимізації було обрано розріджене воксельне октодерево.

Представлений алгоритм оптимізації розрідженого воксельного октодерева, передбачає знаходження та злиття однакових вузлів. Запропонована оптимізація дозволяє зменшити кількість вузлів октодерева, що відповідно зменшить обсяг даних, які потрібно відправляти в GPU. Це, у свою чергу, дозволить значно прискорити процес обробки і візуалізації воксельних даних.

4. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСОБУ

4.1. Середовище програмування

Важливим критерієм є швидкодія реалізованої програми, підтримка векторизації, наявність вільних бібліотек. Тому було розглянуто декілька мов програмування:

- C – Низькорівнева, імперативна мова програмування загального призначення;
- C++ – Високорівнева мова програмування з підтримкою об'єктно-орієнтованої, узагальненої та процедурної парадигм програмувань;
- C# - Високорівнева мова програмування з безпечною типізацією для платформи .Net і високою швидкості розробки.

C# як основна мова програмування не задовольняє з декількох причин:

- Механізм пакування та розпакування. Це механізм перетворення скалярних типів у вказівникові та навпаки за допомогою використання фундаментального класу object, що дуже сильно впливає на швидкодію;
- Збирач сміття (англ. garbage collection). Це один із методів керування пам’яттю під час виконання програми. Недоліком даного методу є відсутність прямого керування пам’яттю та значний вплив на швидкодію програми;
- Немає можливості використовувати всі переваги векторизації.
- Орієнтованість на ОС Windows.

Основним недоліком мови програмування C є дуже низький рівень безпеки, що значно підвищує час розробки та пошуку можливих несправностей.

Особливістю мови програмування C++ є можливість обчислення виразів на етапі компіляції, що дозволяє значно зменшити час виконання програми.

Мова програмування C++ дозволяє знайти баланс між безпекою, часом розробки та швидкістю виконання отриманої програми.

Тому, зважаючи на всі перераховані особливості, C++ було обрано як основну мову програмування.

Оскільки цільовою платформою програмування є Linux, на вибір є декілька середовищ програмування:

- Code::Blocks;
- CLion;
- CodeLite IDE;
- Netbeans;
- Eclipse;
- Qt Creator.

Eclipse – безкоштовний кросплатформений додаток із відкритим кодом (рис. 4.1). Особливостями Eclipse є:

- Підтримка рефакторингу коду;
- Підтримка автодоповнення коду;
- Підтримка багатьох компіляторів;
- Наявність засобів декомпіляції.

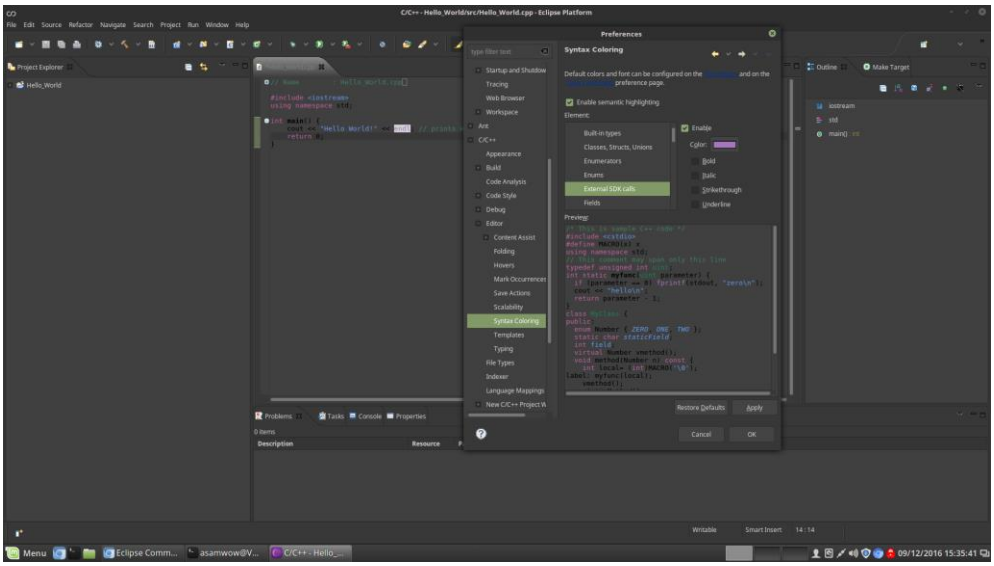


Рисунок 4.1 – Eclipse

Netbeans – безкоштовний кросплатформений додаток з відкритим

кодом (рис. 4.2). Особливостями Netbeans є:

- Підтримка налагодження програми;
- Підтримка автодоповнення коду;
- Підтримка тестів;
- Автоматична упаковка скомпільованого додатку в архів.

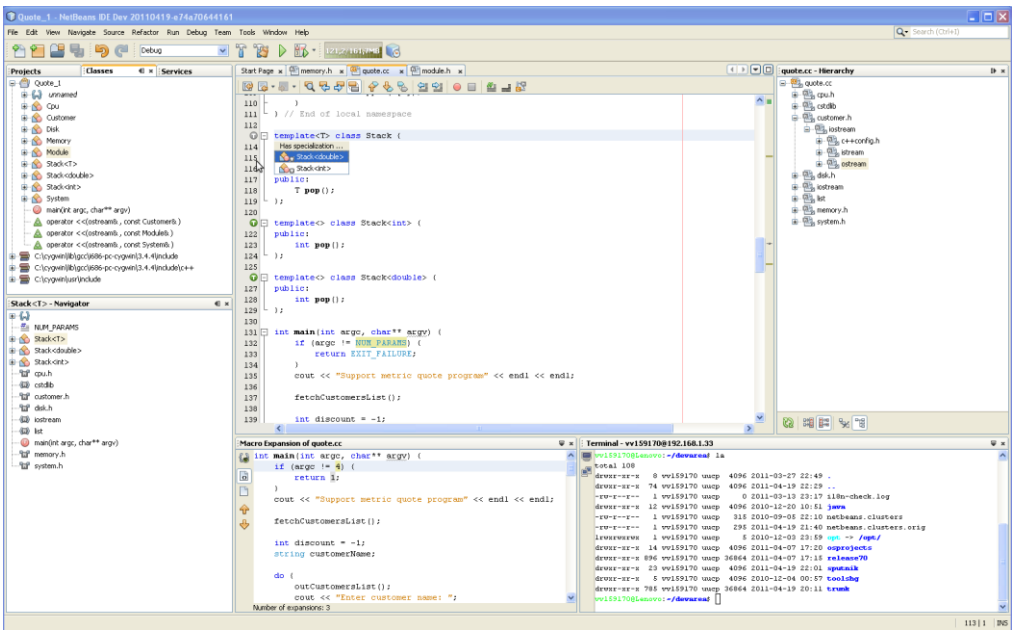


Рисунок 4.2 – Netbeans

CodeLite IDE – майже безкоштовний кросплатформний додаток із відкритим кодом (рис. 4.3). Особливостями CodeLite IDE є:

- Підтримка автодоповнення коду;
- Підтримка рефакторингу коду;
- Підтримка налагодження програми;
- Підтримка засобів контролю версій;
- Підтримка багатьох компіляторів;
- Відображення помилок в коді.

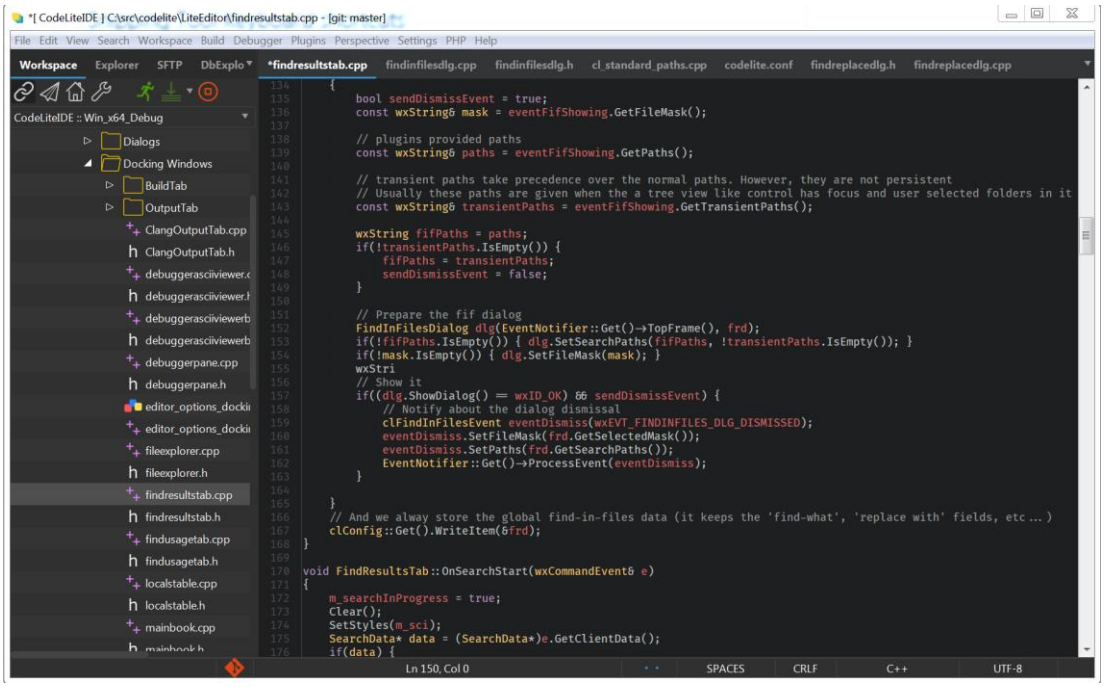


Рисунок 4.3 – CodeLite IDE

Code::Blocks – це безкоштовний кросплатформенний додаток із падтримкою розширень (рис. 4.4). Особливостями Code::Blocks є:

- Підтримка багатьох компіляторів;
- Висока швидкість компіляції, використовуючи інтегрований компілятор;
- Підтримка налагодження програми.

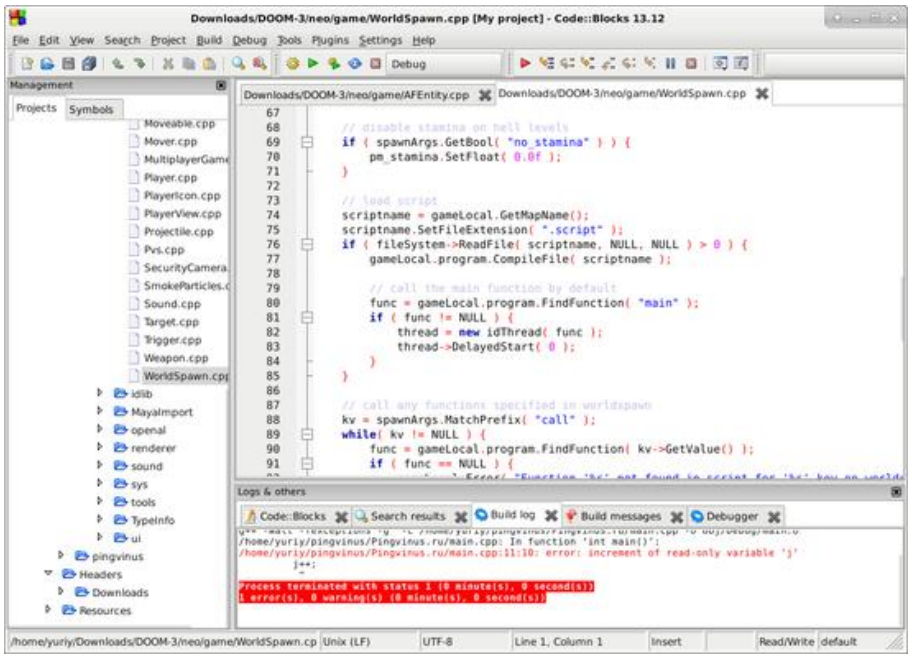


Рисунок 4.4 – Code::Blocks

CLion – кросплатформенний додаток із підтримкою розширень (рис. 4.5). Особливостями CLion є:

- Аналіз коду на льоту;
- Підтримка автогенерації коду;
- Підтримка рефакторингу коду;
- Підтримка налагодження програми;
- Легка навігація в коді;
- Підтримка засобів контролю версії Git, Subversion, Mercurial, CVS, Perforce, TFS;
- Підтримка тестів.

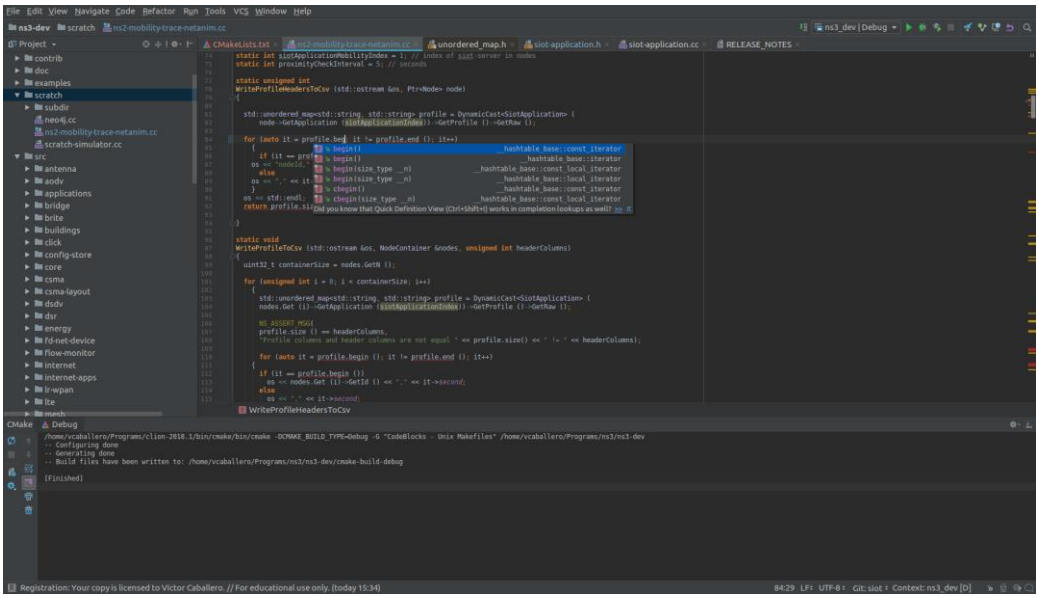


Рисунок 4.5 – CLion

Qt Creator – інтегроване середовище розробки з підтримкою розширень, призначене для створення кросплатформних програмних засобів з використанням бібліотеки Qt (рис. 4.6). Особливостями Qt Creator є:

- Створений спеціально для розробки на Qt;
- Вбудований редактор форм;
- Контекстно-залежна система допомоги;
- Можливість налаштовувати етапи складання проекту;
- Є графічний фронтенд для GDB.

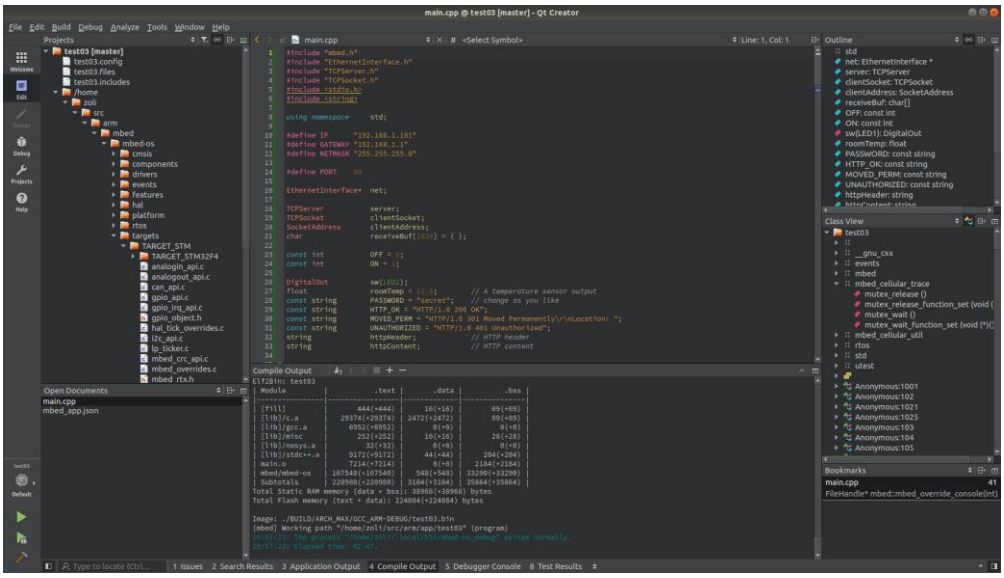


Рисунок 4.6 – Qt Creator

Важливим критерієм при виборі середовища програмування було швидкість і зручність програмування, можливість налаштувати редактор коду під власні потреби, наявність підтримки контролю версії Git, можливість налагодження програми, автодоповнення та рефакторингу коду. Тому обрано середовище програмування CLion.

4.2. Структура розробленого програмного засобу

Для демонстрації процесу процедурної генерації ландшафту за допомогою вокселів з використонням модифікованого розрідженого воксельного октодерева розроблено і реалізовано програмний засіб. Програма складається із чотирьох частин.

Перша частина відповідає за візуалізацію. Вона реалізована за допомогою технології Vulkan (рис. 4.7). Vulkan – багатоплатформний прикладний програмний інтерфейс для 3D графіки і супровідних обчислень. Основною метою створення Vulkan є раціональніше використання можливостей GPU і CPU в порівнянні з OpenGL. Він впроваджує прямий контроль над роботою GPU і, в свою чергу, зменшує навантаження на CPU.

Рисунок 4.7 – Логотип Vulkan

Друга частина відповідає за генерацію ландшафту (рис. 4.8).

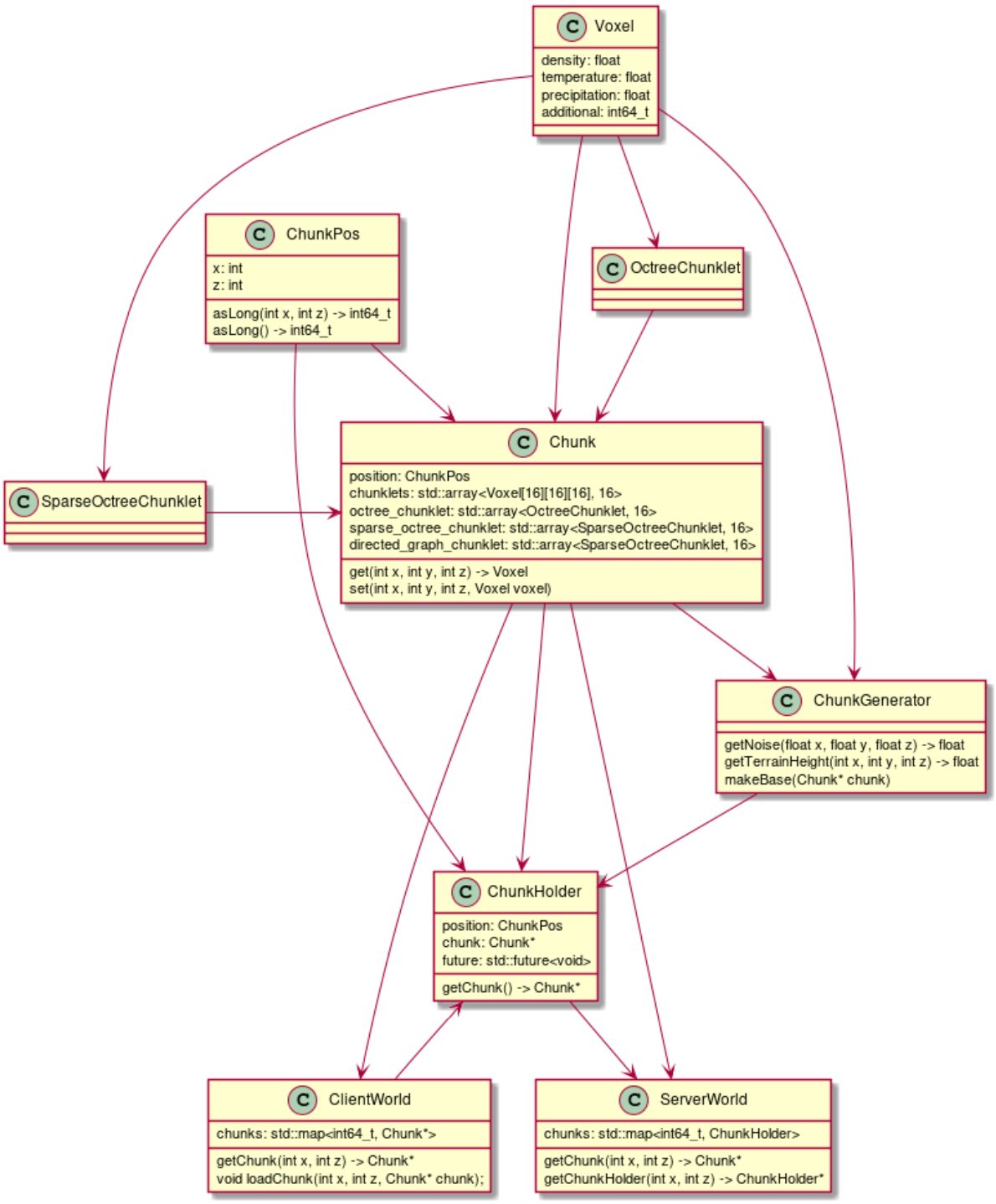


Рисунок 4.8 – Структура класів

Клас Chunk зберігає позицію чанка в світі та інформацію про вокселі у різному вигляді: тривимірний масив; октодерево; розріджене воксельне октодерево; оптимізоване розріджене воксельне октодерево. Метод set дає змогу зберегти воксель в заданій позиції. Метод get дає змогу отримати воксель в заданій позиції.

Клас ChunkHolder зберігає позицію чанка в світі, чанк для якого потрібно згенерувати вокселі, і завдання для генерації вокселей, яка відбувається в іншому потоці виконання програми. Метод getChunk дає змогу отримати чанк для якого генеруються вокселі.

Клас ChunkGenerator відповідає за генерацію вокселей для заданого чанку. Метод getNoise обчислює шум перлина в заданій позиції використовуючи фрактальний броунівський рух. Метод getTerrainHeight визначає щільність вокселя в заданій позиції. Метод makeBase заповнює чанк згенерованими вокселями.

Клас ServerWorld зберігає чанки які уже згенеровані, або тільки в процесі генерації. Метод getChunk дає змогу отримати чанк в заданій позиції. Метод getChunkHolder дає змогу отримати ChunkHolder для заданої позиції, якщо його не існує, створюється новий чанк і створюється завдання для генерації вокселей для заданого чанка.

Клас ClientWorld зберігає повністю згенеровані чанки. Метод getChunk дає отримати чанк в заданій позиції. Метод loadChunk використовується для того, щоб зберегти повнювністю згенерований чанк в заданій позиції.

Третя частина відповідає за підготовку даних для візуалізації (рис. 4.9).

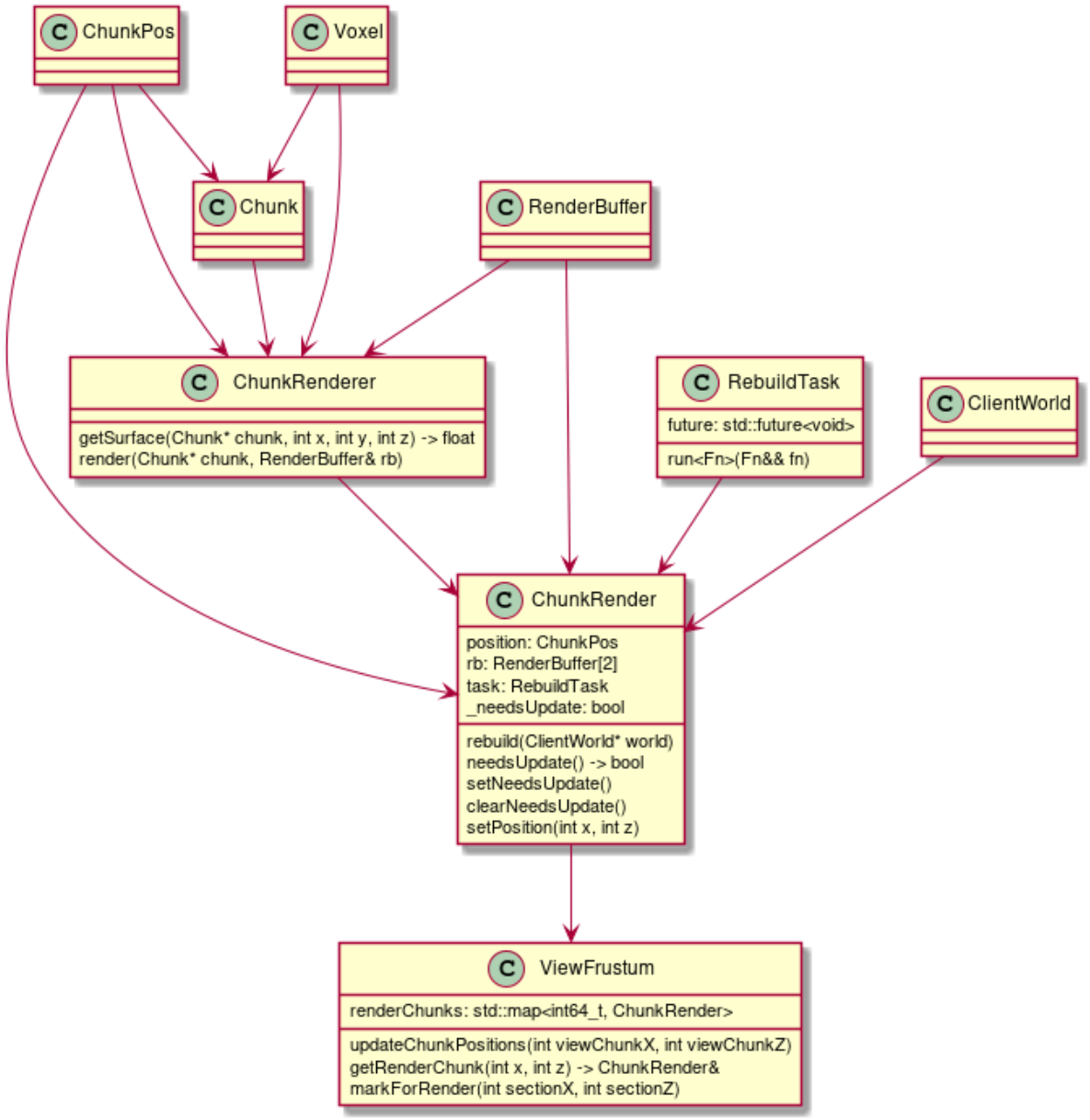


Рисунок 4.9 - Структура класів

Клас RenderBuffer – дані для візуалізації в Vulkan.

Клас ChunkRenderer відповідає за підготовку даних для візуалізації. Метод getSurface дає змогу отримати щільність вокселя для заданої позиції і чанку. Метод render підготовує дані для візуалізації в Vulkan.

Клас RebuildTask виконує потрібне завдання в іншому потоці виконання програми.

Клас ChunkRender зберігає позицію чанку, для якого відбується візуалізація, дані для візуалізації в Vulkan, завдання для підготовки даних, і флаг, який вказує чи потрібно оновити дані. Метод rebuild створює завдання для підготовки даних для візуалізації в Vulkan. Метод needUpdate перевіряє

чи потрібно оновити дані. Метод setNeedsUpdate позначає чанк, що дані потрібно оновити. Метод clearNeedsUpdate позначає чанк, що дані не потрібно оновлювати.

Метод setPosition визначає позицію чанка, для якого потрібно підготовити дані для візуалізації.

Клас ViewFrustum зберігає дані для візуалізації чанків. Метод updateChunkPositions оновлює позиції для чанків в заданому центрі. Метод getChunkRender дає змогу отримати дані для візуалізації чанка. Метод markForRender позначає чанк і його сусідів, що дані потрібно оновити.

Четверта, основна частина відповідає за взаємодію всіх інших частин (рис. 4.10).

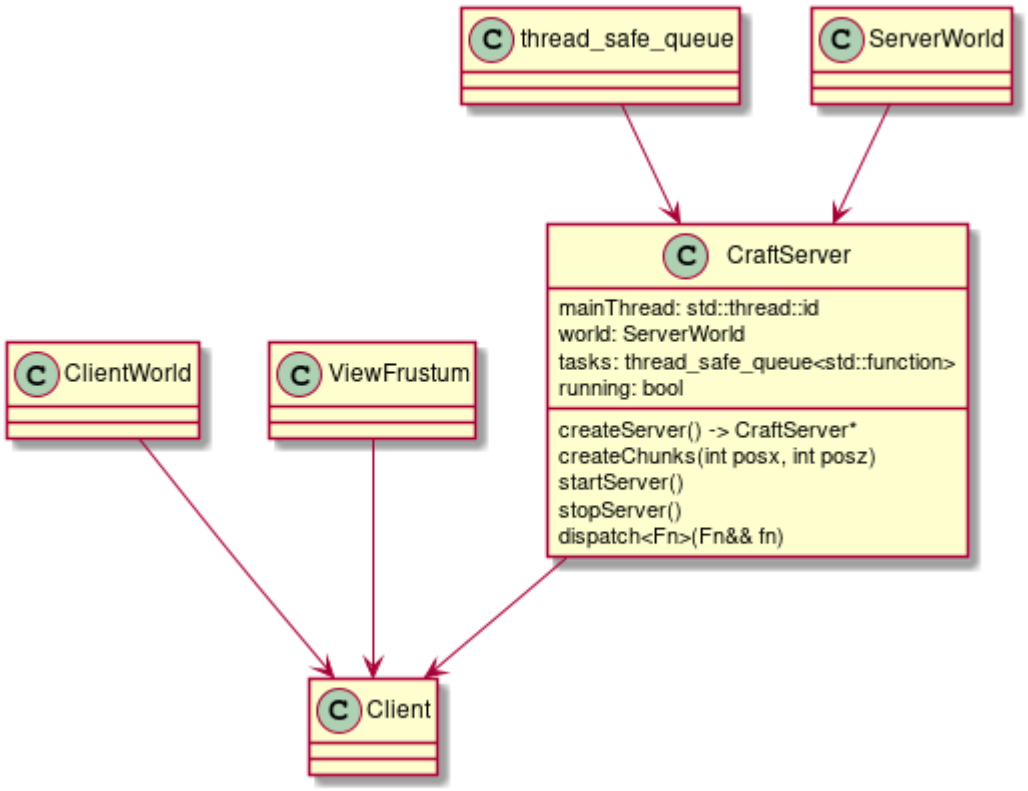


Рисунок 4.10 – Структура класів

Клас CraftServer збрегіє потік, в якому він був створений, світ для якого генеруються чанки і флаг, який вказує чи CraftServer запущений. Метод createServer створює новий сервер в окремому потоці. Метод startServer відповідає за виконання завдань в окремому потоці виконання програми. Метод stopServer зупиняє CraftServer. Метод dispatch добавляє нове завдання,

яке потрібно виконати в потоці виконання завдань для CraftServer. Метод createChunks створює чанки в заданій позиції в певному радіусі.

4.3. Оцінка отриманих результатів

Реалізація декілька способів опису воксельних даних дала змогу порівняти та оцінити скільки пам’яті потребує кожний спосіб (рис. 4.11).

Тривимірний масив використовує найбільше пам’яті, оскільки він описує всі можливі вокселі.

Звичайне октодерево використовує менше пам’яті ніж тривимірний масив. Це зумовлено відсутністю в октодереві вузлів для тих ділянок ландшафту, де відсутні вокселі.

Розріджене воксельне октодерево використовує майже таку ж кількість пам’яті, як і звичайне октодерево. Це зумовлено тим, що згенерований ландшафт не має великого перепаду висот, тому кількість вузлів майже співпадає.

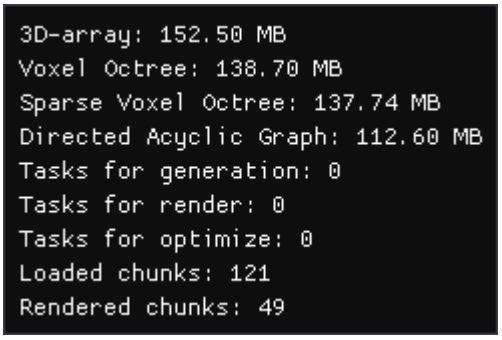


Рисунок 4.11 – Використання пам’яті

Проаналізувавши швидкість зростання необхідного обсягу пам’яті (рис. 4.12), помітно, що для тривимірного масиву вони найбільші, для октодерева і розрідженого воксельного октодерева майже однакові, а для оптимізованого розрідженого воксельного октодерева, найнижчі.

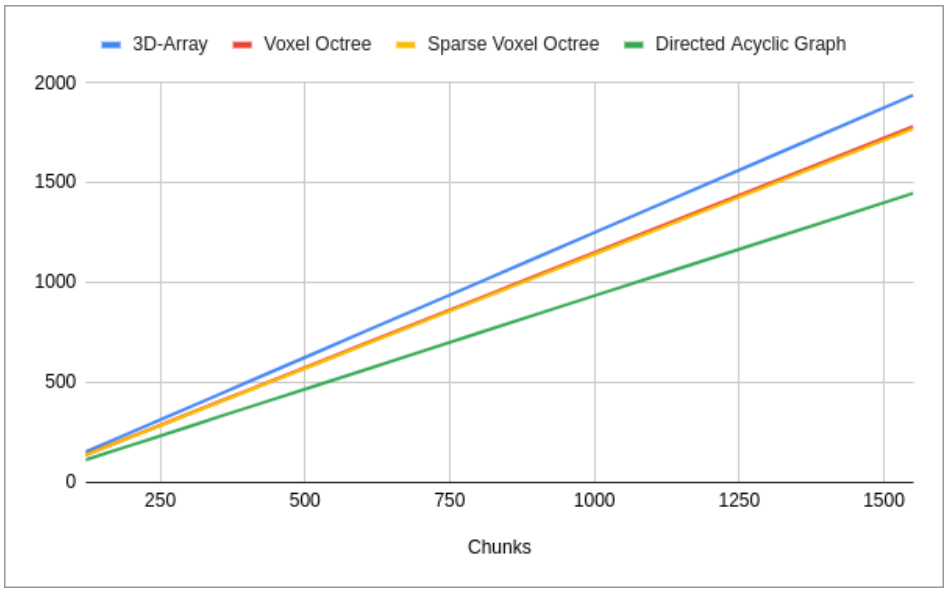


Рисунок 4.12 – Використання пам’яті

Висновки до розділу 4

Визначено загальну структуру програмного засобу, що використовує запропонований алгоритм оптимізації розрідженого воксельного октодерева, та здійснено програмну реалізацію.

Програма демонструє процес процедурної генерації ландшафту за допомогою вокселів з використанням оптимізованого розрідженого воксельного октодерева.

В роботі представлено структуру програми, а також опис елементів програми та їх взаємодії.

Програма генерує ландшафт, використовуючи декілька способів представлення воксельних даних:

- Тривимірний масив;
- Октодерево;
- Розріджене воксельне октодерево;
- Оптимізоване розріджене воксельне октодерево.

Проаналізовано та порівняно результати генерації ландшафту з використанням різних способів опису воксельних даних. Результати доводять ефективність оптимізації розрідженого воксельного октодерева.

ВИСНОВКИ

Метою даної магістерської дисертації було підвищання ефективності процедурної генерації ландшафту використовуючи розріджене воксольне октодерево для опису воксельних даних із запропонованою оптимізацією.

В ході виконання магістерської дисертації було проаналізовано ряд алгоритмів та програмних засобів для процедурної генерації ландшафтів.

Було проаналізовано і реалізовано декілька способів опису воксельних даних.

Тривимірний масив найбільше підходить для рівномірно розподілених вокселей. Октодерево і розріджене воксельне октодерево показують схожі результати, якщо в процедурно згенерованому ландшафті немає перепаду висот.

Розріджене воксельне октодерево найбільше підходить для опису не рівномірно розподілених вокселей.

Таким чином, можна виділити основні моменти:

- Програмні засоби для процедурної генерації ландшафтів не універсальні;
- Більшість алгоритмів процедурної генерації є простими в реалізації;
- Спосіб опису воксельних даних потрібно обирати зважаючи на те, які дані потрібно описати;
- Пошук ефективних методів візуалізації воксельних даних є предметом постійних досліджень.

Практична цінність отриманих результатів полягає в тому, що запропонована оптимізація розрідженого воксельного октодерева дає змогу зменшити обсяг необхідної пам’яті. Це дозволить зменшити об’єм даних, які потрібно візуалізувати.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Kirill Tokarev (May 2018). How Voxels Became “The Next Big Thing”. Режим доступу до ресурсу: https://medium.com/@EightyLevel/how-voxels-became-the-next-big-thing-4eb9665cd13a.
2. Meagher, Donald (October 1980). “Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer”. Rensselaer Polytechnic Institute (Technical Report IPL-TR-80-111).
3. Laine, Samuli; Tero Karras (February 2010). “Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation” (PDF). NVIDIA Corporation. Retrieved June 11, 2010. – Режим доступу до ресурсу: https://research.nvidia.com/sites/default/files/pubs/2010-02_Efficient-Sparse- Voxel/laine2010tr1_paper.pdf
4. Thulasiraman, K.; Swamy, M. N. S. (1992), “5.7 Acyclic Directed Graphs”, Graphs: Theory and Algorithms, John Wiley and Son, p. 118, ISBN 978-0-471-51356-8.
5. Jon Olick. SIGGRAPH 2008: Current and Next Generation Parallelism. – Режим доступу до ресурсу: https://www.webcitation.org/66bQncVWe?url=http://s08.idav.ucdavis.edu/olick-current-and-next-generation-parallelism-in-games.pdf
6. A Survey of Procedural Noise Functions, Computer Graphics Forum, Volume 29 (2010), Number 8, pp 2379—2600.
7. Lorensen, W. E.; Cline, Harvey E. (1987). "Marching cubes: A high resolution 3d surface construction algorithm". ACM Computer Graphics.
8. Akio Doi, Akio Koide. "An Efficient Method of Triangulating Equi-Valued Surfaces by Using Tetrahedral Cells." IEICE Transactions of Information and Systems, Vol.E74-D No. 1, 1991.

9. Zhou Y, Chen W, Tang Z. An elaborate ambiguity detection method for constructing isosurfaces within tetrahedral meshes. Computers Graphics 1995;19(3):355–64.
10. Tao Ju, Frank Losasso, Scott Schaefer, Joe Warren: Dual Contouring of Hermite Data.
11. Ken Perlin, Making noise. Based on a talk presented at GDCHardcore (Dec 9, 1999).

ДОДАТКИ