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 004.021

«До захисту допущено»

Завідувач кафедри СПСКС

_____ В.П.Тарасенко_
(підпис) (ініціали, прізвище)

"___"_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія (Комп'ютерні системи та компоненти)

на тему МЕТОДИ ТА ПРОГРАМНІ ЗАСОБИ ГЕНЕРАЦІЇ ЛАНДШАФТУ В ІГРОВИХ ПРОГРАМАХ

Виконала: студентка ІІ курсу, групи КВ-71мп
(шифр групи)

Шевчук Олександра Юріївна _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник д.т.н., проф. Зайцев В.Г. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

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

Київ – 2018 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)
Спеціальність 123 Комп'ютерна інженерія
Комп'ютерні системи та компоненти

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ _В.П.Тарасенко__
(підпис) (ініціали, прізвище)
«___»_____2018р.

ЗАВДАННЯ
на магістерську дисертацію студентки
Шевчук Олександрі Юріївні

1. Тема дисертації: МЕТОДИ ТА ПРОГРАМНІ ЗАСОБИ ГЕНЕРАЦІЇ ЛАНДШАФТУ В ІГРОВИХ ПРОГРАМАХ,
науковий керівник дисертації: д.т.н., проф. Зайцев В.Г,
затверджені наказом по університету від « 30» жовтня 2018 р. № 4030-с.

2. Термін подання студентом дисертації: 7 грудня 2018 р.

3. Об'єкт дослідження: методи та алгоритми генерації ландшафту в ігрових програмах.

4. Предмет дослідження: програмна реалізація розробленого алгоритму генерації ландшафту в ігрових програмах.

5. Перелік завдань, які потрібно розробити:
– розглянути основні методи генерації ландшафту в ігрових програмах;
– провести порівняльний аналіз методів генерації ландшафту;
– дослідити методи оптимізації існуючих алгоритмів;
– розробити та описати етапи обраного алгоритму генерації ландшафту в ігрових програмах;
– програмно реалізувати розроблений алгоритм генерації ландшафту;
– розглянути результати генерації тривимірного воксельного ландшафту;
– провести експерименти і оцінити роботу оптимізованого алгоритму.

6. Перелік ілюстративного матеріалу:
– Структурна схема архітектури програмної системи;
– Діаграма октодерева;
– Презентація.

7. Перелік публікацій: «Методи та програмні засоби генерації ландшафтів в ігрових програмах», XI наукова конференція молодих вчених «Прикладна математика та комп'ютинг» ПМК-2018-2 (Київ, 14-16 листопада 2018 р.); «Методи та програмні засоби генерації ландшафтів в ігрових програмах», V Міжнародна науково-технічна конференція «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами» (Київ, 22-23 листопада 2018 р.).

8. Дата видачі завдання 5 вересня 2017 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Ознайомлення з предметною галуззю	17.12.2017	
2	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	05.02.2018	
3	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.03.2018	
4	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	27.06.2018	
5	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	12.09.2018	
6	Проведення наукового дослідження; робота над третім розділом магістерської дисертації; підготовка матеріалів доповіді на конференції «Прикладна математика та комп'ютинг» ПМК-2018-2	03.10.2018	
7	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу	16.10.2018	
8	Оформлення текстової і графічної частини магістерської дисертації	10.11.2018	
9	Попередній розгляд магістерської дисертації на кафедрі	26.11.2018	

Студент _____ Шевчук О.Ю.

Науковий керівник дисертації _____ Зайцев В.Г.

РЕФЕРАТ

Актуальність теми. Результатом технологічного розвитку, а також масового розповсюдження різних девайсів, стало розширення ринку розваг, завдяки чому зросла увага до проектів зі створення нових ігор, які застосовуються в багатьох галузях. Одним з етапів створення певних комп'ютерних ігор є генерація (створення) ландшафту. Нові карти можна малювати використовуючи послуги художників в цій галузі. Ще одним варіантом є програмна генерація початкової випадкової карти або ландшафту. Алгоритми генерації ландшафту зазвичай націлені на вирішення лише однієї задачі, тому часто один певний алгоритм не може цілком задовольнити вимоги до вихідного результату. Від якості зовнішньої сторони, а також комфорту ігрового процесу напряму залежить прибутковість гри, її сприймання користувачами, а також подальший розвиток, тому розробка нового, оптимізованого алгоритму генерації ландшафтів в ігрових програмах є актуальною задачею як з наукової, так і з практичної точки зору.

Об'єктом дослідження є методи та алгоритми генерації ландшафту в ігрових програмах.

Предметом дослідження є програмна реалізація розробленого алгоритму генерації ландшафту в ігрових програмах.

Мета роботи: підвищення ефективності процесу генерації ландшафту в ігрових програмах за рахунок використання розрідженого воксельного октодерева в поєднанні з алгоритмом Wire deformation.

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

Наукова новизна полягає в наступному:

1. Запропоновано оптимізований алгоритм генерації ландшафтів в ігрових програмах, який відрізняється від існуючих підвищеною точністю деталей вихідного ландшафту.

2. Запропонований алгоритм дозволяє зменшити вимоги до пам’яті, що використовується.

Практична цінність отриманих в роботі результатів полягає в тому, що запропонований алгоритм генерації ландшафту в ігрових програмах дозволяє ефективно виконувати необхідну модифікацію місцевості, а також створювати більш складне оточення, наприклад моделюючи печери і тунелі. Також використання розрідженого воксельного октодерева в ігрових програмах дозволяє генерувати ландшафт без потреби його перемальовування в залежності від зміни кута повороту або модифікацій. Результати роботи розробленого алгоритму дозволяють ефективно створювати візуальну частину гри, що призводить до підвищення якості програмного продукту, а отже до його подальшого розвитку і розповсюдження.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на XI науковій конференції молодих вчених «Прикладна математика та комп’ютинг» ПМК-2018-2 (Київ, 14-16 листопада 2018 р.), а також на V Міжнародній науково-технічній конференції «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами» (Київ, 22-23 листопада 2018 р.).

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

У вступі надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету дослідження, показано наукову новизну отриманих результатів і практичну цінність роботи.

У першому розділі розглянуто способи створення ландшафту і основні методи генерації ландшафту в ігрових програмах; наведено теоретичні засади генерації, а також проведено порівняльний аналіз методів генерації ландшафту в ігрових програмах.

У другому розділі проаналізовано воксельну технологію, описано структури воксельних даних на основі дерева, проведено оцінку структури дерева, досліджено методи оптимізації воксельного алгоритму.

У третьому розділі описано застосування алгоритму Marching Cubes, а також використання методу Voxel carving для генерації ландшафту в ігрових програмах; розроблено та описано етапи алгоритму генерації ландшафту в ігрових програмах з використанням розрідженого воксельного октодерева в поєднанні з алгоритмом Wire deformation; представлено варіант згенерованого ландшафту і його деформацію за допомогою алгоритму Wire deformation.

У четвертому розділі з безпекою у приміщенні, в якому проводилась розробка, програмно реалізовано розроблений алгоритм генерації ландшафту в ігрових програмах з використанням розрідженого воксельного октодерева в поєднанні з алгоритмом Wire deformation; описано структуру розробленого програмного продукту; розглянуто результати генерації тривимірного воксельного ландшафту; проведено експерименти з подальшою оцінкою використання розрідженого воксельного октодерева в процесі генерації ландшафту в ігрових програмах.

У висновках проаналізовано отримані результати роботи.

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

Магістерська дисертація виконана на 85 аркушах, містить 3 додатки та посилання на список використаних літературних джерел з 43 найменувань. У роботі наведено 53 рисунків та 5 таблиць.

Ключові слова: воксельний алгоритм, генерація ландшафту, методи оптимізації, візуальна частина гри, розріджене воксельне октодерево.

ЗМІСТ

ВСТУП

Результатом технологічного розвитку, а також масового розповсюдження різних гаджетів, стало розширення ринку розваг, завдяки чому зросла увага до проектів зі створення нових ігор, які застосовуються в багатьох галузях. За 40 років невелика галузь комп'ютерних ігор досягла таких масштабів, що вже багато в чому перевершує найближчих конкурентів: кіноіндустрію, музичну промисловість, шоу-бізнес.

Одним з етапів створення певних ігрових програм є генерація (створення) ландшафту. Нові карти можна малювати використовуючи послуги графічних дизайнерів або художників ігрового оточення. Іншим варіантом, без залучення сторонніх спеціалістів, є програмна генерація початкової випадкової карти або ландшафту. Генератор сценаріїв [6] – це програмне забезпечення, яке використовується для створення зображень ландшафту, 3D-моделей та анімацій. Алгоритми генерації ландшафту зазвичай націлені на вирішення лише однієї задачі, тому часто один певний алгоритм не може цілком задовольнити вимоги до вихідного результату.

Генерація ландшафтів має велике значення для результуючого програмного продукту. Від якості зовнішньої сторони, а також комфорту ігрового процесу напряму залежить прибутковість гри, її сприймання користувачами, а також подальший розвиток, тому створення нових, оптимізованих методів генерації ландшафтів є актуальною практичною задачею.

Метою магістерської дисертації є підвищення ефективності процесу генерації ландшафту в ігрових програмах за рахунок використання розрідженого воксельного октодерева [7] в поєднанні з алгоритмом Wire deformation [3], що дозволить більш ефективно розробляти візуальну частину гри і призведе до підвищення якості програмного продукту, а отже його подальшого розвитку і розповсюдження.

1. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ГЕНЕРАЦІЇ ЛАНДШАФТУ В ІГРОВИХ ПРОГРАМАХ

1.1. Ландшафт в ігрових програмах та існуючі рішення його генерації

В комп'ютерних іграх існує таке поняття, як ігрове оточення, яке є дуже важливою складовою будь-якого проекту. Саме воно створює ігрову атмосферу. За допомогою оточення можна вміло створювати різноманітні ситуації, що занурюють гравця у віртуальний світ. Оточення – це своєрідне тіло гри, в яке поміщені персонажі і яке повинно повністю відповідати ігровій механіці. У сучасному світі найбільшу популярність здобувають ті ігри, чиє оточення виглядає максимально реалістично. Запорукою цієї реалістичності і її фундаментом є ігровий ландшафт.

Ландшафт – це відображення ділянки землі, яке розглядається з урахуванням її природних особливостей, військових переваг тощо. Ландшафт або рельєф (також топографічний рельєф) передбачає вертикальні та горизонтальні розміри поверхні суші.

Важливу роль відіграє генератор сценаріїв – програмне забезпечення, яке використовується для створення зображень ландшафту, 3D-моделей та анімацій. Ці програми часто використовують процедурне генерування для створення пейзажів. Основними елементами ландшафтів, створених декораційними генераторами, є місцевість, вода, листя та хмари.

Більшість декоративних генераторів здатні створювати базові висоти діапазонів, щоб імітувати зміни висоти в базовій місцевості. Загальні методи включають в себе використання шуму, фрактали [8] або алгоритм діамантів [9], які здатні генерувати 2D висоту кадрів. Однак, скелі та печери, завдяки своїй тривимірній природі, не можуть бути створені за допомогою основних методів вимірювання висоти, і створюються різноманітними альтернативними способами. Органи води мають свою власну чітку поведінку і утворюють озера, річки та океани. Вони або створюються окремо як унікальний аспект декорацій, або обчислюються з існуючих висот у карті висоти. Рослинність також може бути змодельована на вершині

сформованої місцевості з метою досягнення природного ландшафту. Такі функції, як дерева або чагарники, зазвичай створюються за допомогою L-систем [10] або фракталів, завдяки унікальній структурі. Хмари та інші атмосферні ефекти рідше виявляються у генераторах декорацій, і створюються за допомогою різних застосувань фракталів та шуму.

Окрім процедурних методів, генератори декорації, такі як Grome [11], також можуть дозволити користувачам вручну редагувати властивості місцевості, часто за певними правилами. Невелика кількість програмних пакетів, які не генерують місцевість, часто спеціалізуються на створенні конкретних аспектів. Наприклад, Speedtree [12] використовується для створення тільки рослинності, тоді як CityEngine [13] спеціалізується на створенні реалістичних будівель та інфраструктури.

За способом представлення ігрові ландшафти можна розділити на двовимірні (2D) і тривимірні (3D), які використовуються у 2D і 3D іграх відповідно.

Існує декілька варіантів створення ландшафтів. Перший варіант – це використання графічних редакторів, таких як Zbrush [14], Autodesk 3ds Max [15], Blender [16], World machine [17], Terragen [18] та інші. Одним із недоліків використання таких редакторів є те, що змінювати параметри створених об’єктів можна лише в самому редакторі, що незручно і часто займає зайвий час. Другим варіантом є процес створення ландшафту вручну, без використання додаткових редакторів. Даний спосіб є більш складним, так як для його реалізації необхідно розуміти принцип роботи алгоритмів генерації, що потребує також знань у певних, досить складних, розділах математики. Перевагою такого методу є можливість швидко змінювати необхідні параметри ландшафту без потреби використовувати додаткові програмні застосунки.

Одним із найстаріших і найпростіших способів створення 2D ландшафтів є тайлова карта (tilemap) [19]. Тайли – невеликі зображення однакових розмірів, які служать фрагментами великої картини. Зазвичай

тайлів для розробки ігрового світу роблять порядку декількох сотень. Матриця клітин складається з номерів тайлів. Таким чином можна будувати багато двомірних просторів, які витрачають зовсім небагато пам'яті. Всі клітини діляться на прохідні і непрохідні. Поняття висоти тут немає, гірські плато нічим не відрізняються від низин. Деякі ігри зображають тайлову карту у вигляді не квадратів, а шести- або восьмикутників. Розвитком цього став ізометричний метод, який ще називають 2.5D [20] при якому квадрат карти розвертається на 30-45°. Ці квадрати можуть перекривати один одного по висоті для зображення височин, дерев, будівель тощо. В деяких ізометричних іграх є реальне поняття висоти, що обмежує дії гравця в переміщенні, будівництві або інших діях.

Під час процесу створення 3D ландшафтів у роботі [2] пропонується розглянути два види генерації тривимірних зображень. Перший – це процес генерації зображень в реальному часі (Real-time), де застосовується система умовних припущень, що дозволяє застосовувати більш прості алгоритми і тим самим значно зменшити обсяг обчислень, хоча якість вихідного зображення в результаті погіршується; другий – процес генерації високо реалістичних тривимірних зображень (High realistic), який потребують дуже великих обсягів обчислень.

Збільшення швидкості побудови Real-time ландшафтів досягається за допомогою апаратної реалізації частини алгоритмів, такі як алгоритми побудови та текстурування трикутників, видалення ступінчастості тощо. Подібна реалізація досягається з допомогою використання різних бібліотек, таких як, наприклад, OpenGL [21] і Direct3D [22]. Переваги їх використання полягають у тому, що відбувається розвантаження CPU, який в певних ситуаціях бере на себе додаткові функції. Також їх використання дозволяє програмісту не вникати в деталі вже відомих алгоритмів, а направити свою діяльність на їх оптимізацію або створення нових. На даний момент майже всі системи використовують одну або обидві ці бібліотеки, що доводить їх користь в роботі з Real-time ландшафтами.

1.2 Класифікація ландшафтів і способи їх представлення

Створення High realistic ландшафтів відбувається за допомогою алгоритмів перетворення вхідних даних. У роботі [1] демонструється класифікація даних про ландшафт залежно від їх представлення. Наприклад, можна подати дані у вигляді регулярної сітки висот або карти висот (HeightMap), нерегулярної сітки вершин і зав'язків, а також зберігання карти ландшафту, але в даному випадку зберігаються не конкретні висоти, а інформація про використаний блок [40]. У цьому випадку, згідно з джерелом створюється деяка кількість заздалегідь побудованих сегментів, а на карті вказуються тільки індекси цих сегментів.

Карти висот – це двомірні карти, які використовуються для зберігання висот ландшафту. Карта висот ландшафту зазвичай зберігається в 8-розрядному зображенні, її значення висот варіюються від 0 до 255, де 0 (чорний колір) являє найнижчу висоту вершини, а 255 (білий колір) представляє максимально можливу висоту вершини. Користувач може розширити цей інтервал, використовуючи коефіцієнт масштабування, який множиться на задане значення висоти, збільшуючи його діапазон. Це забезпечує більший інтервал висот, але з меншою точністю між значеннями.

Для створення 3D ландшафтів з використанням висотних діапазонів використовують масив 2D значень. Кожне значення в масиві являє собою висоту рельєфу у позиції цього значення. Наприклад, якщо комірка у (2, 3) має значення 5, то місцевість містить точку (2, 3, 5). Для того, щоб зробити карту висоти у трьох вимірах, створюється сітка, повторюється через два індекси масиву і встановлюється висота в кожній вершині до значення висоти карти в цій точці. (рис.1.1). Як правило, осі X і Y використовуються для двох індексів, а вісь Z для висоти місцевості, але будь-яка інша орієнтація також можлива.

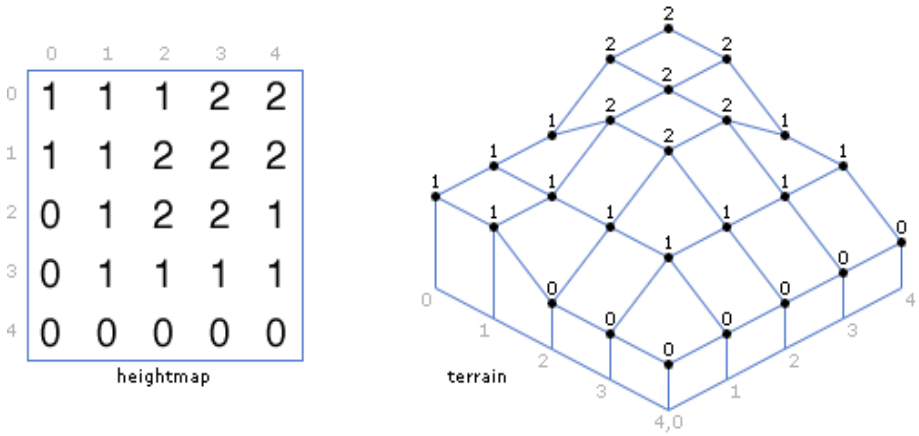


Рисунок 1.1 – Двомірний масив і результуюча карта висот

Інший спосіб зображення карти висоти – це зображення у градаціях сірого (рис.1.2), де яскравість кожного пікселя відповідає висоті рельєфу в цій точці. Таким чином, темні області на зображенні представляють долини, а світлі області – вершини.

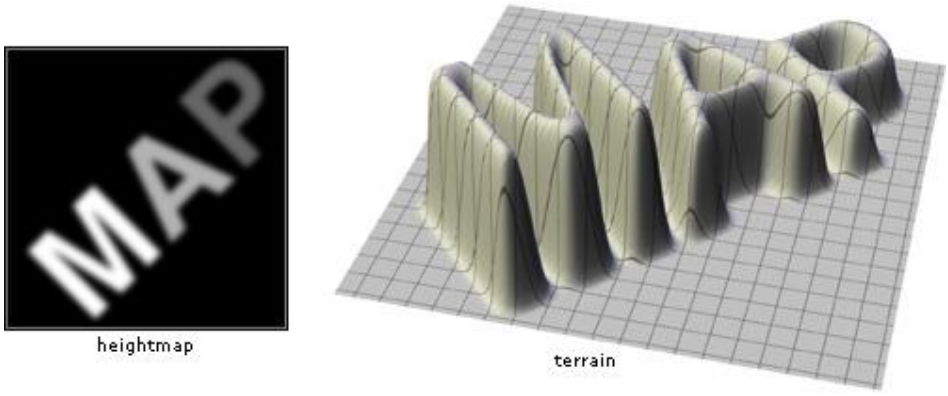


Рисунок 1.2 – Карта висоти у градаціях сірого

Карти висот зберігаються у файлі з форматом RAW, який є простим для зчитування, так як не містить заголовків з інформацією про розмір або тип зображення. Такий файл є двійковим і містить лише дані про висоти ландшафту. Дві координати є положенням конкретного пікселя на зображенні, а третя координата відображає колір. Чим яскравіший колір, тим більша висота для цієї точки. Найчастіше такі картинки є монохромними і містять 256 градацій висоти, але іноді використовуються ще певні кольори, за рахунок чого градація також збільшується.

До переваг цього способу можна віднести простоту видозміни ландшафту, адже на даний момент існує багато стандартних редакторів

растрової графіки. Наступною перевагою є те, що існує можливість зберігання додаткової інформації стосовно певних компонентів ландшафту і об'єктів на ньому, таких як розташування рослинності, будівель тощо. Недоліками даного способу є те, що кількість описів для кожної точки є занадто великою, тобто присутня непотрібна надлишковість даних.

Наступним способом згідно з матеріалом, наведеним у роботі [1], представлення ландшафту є використання іррегулярної сітки вершин і їх зав'язків (тріангуляційна сітка) (рис.1.3). Відображається у вигляді тривимірної моделі і зазвичай використовується в спеціалізованих пакетах для роботи з тривимірною графікою (3Dmax, Maya та інші [15]). В зв'язку з форматом відображення потребує менше інформації для побудови ландшафту, так як необхідно зберігати тільки значення висоти кожної вершини і інформацію про зв'язки між ними. Завдяки цьому швидкість передачі масивів збільшується, що є перевагою даного способу.

Недоліком є те, що оптимізація алгоритмів побудови ландшафтів є досить складною процедурою і потребує значних ресурсів і зусиль. Також складність при динамічному освітленні, так як вершини розташовані один від одного досить далеко і нерівномірно. Перегляд і модифікація такого ландшафту теж є досить складною задачею, з якою не впораються стандартні графічні редактори і знадобиться встановлення спеціальних «важких» програм.

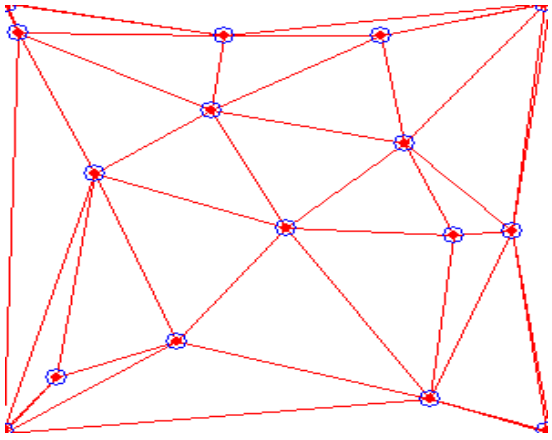


Рисунок 1.3 – Іррегулярна сітка

Третій спосіб представлення ландшафтів, описаний у роботі [1], називається посегментною картою висот і полягає у тому, що так само створюється карта висот, де тепер зберігаються індекси ландшафтних сегментів, які можуть бути як регулярними, так і іррегулярними. Використовувати останні можливо одночасно. Перевагами такого способу є можливість представлення більших ландшафтів, ніж у двох попередніх випадках. Також окрім самих ландшафтів в таких блоках можна зберігати і інформацію про різноманітні об’єкти або специфічні ландшафтні деталі (наприклад, печери або скелі); Доступна можливість створення декількох варіантів одного і того ж сегмента, але при різному ступені деталізації [40]. Залежно від швидкості або завантаженості комп'ютера можна вибирати більш-менш деталізовані варіанти (так звані LOD ландшафти) [23]. Серед недоліків цього способу варто виділити такі: важкість у стикуванні різних сегментів; така велика структура є досить складною для сприйняття людиною, тому досить важко уявити кінцевий варіант ландшафту; проблема модифікації, так як при використанні різних сегментів і різних редакторів до кожного з них, для посегментної карти висот необхідний універсальний редактор, який скоріше за все потрібно буде писати самому.

1.3 Алгоритми генерації карт висот

Розглянемо алгоритми для побудови карти висот. Перший варіант – це використання вбудованої функції Random для генерації псевдовипадкових рівномірно розподілених величин для завдання висоти в кожній точці. Як видно з рис. 1.4, результуюча карта висот є хаотичним набором точок у просторі. Інший можливий варіант – це використання цієї функції з різними вагами. Наприклад, якщо точка знаходиться ближче до центру сцени, то вага менша. Чим далі від центру – тим вага більше і випадкових значень також більше, що не сильно змінює попередній результат.

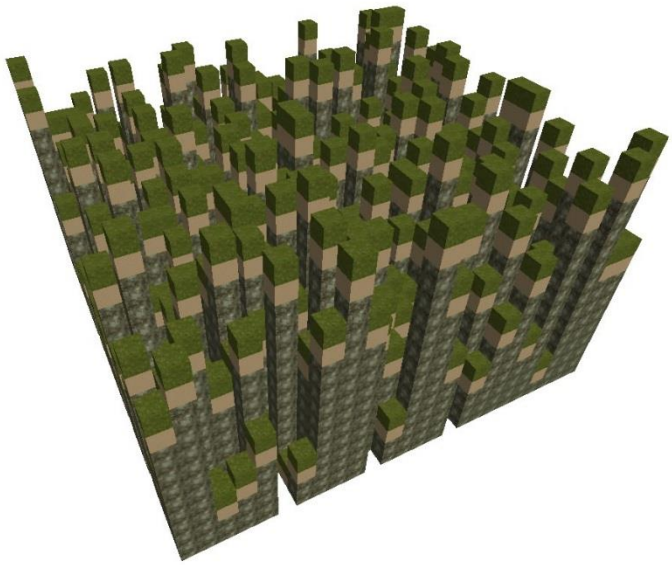


Рисунок 1.4 – Результат використання функції Random

Наступний варіант створення карти висот – це використання шуму Перліна (Perlin noise) – процедурного текстурного примітиву, згенерованого псевдо-випадковим методом, який являє собою тип градієнтного шуму [24].

Функція шуму по суті є вибіркою функції Random і приймає ціле як параметр, а в якості результату на виході маємо випадкове число на основі цього параметра. Якщо передати той же параметр два рази, вона видає те саме число двічі.

На рис.1.5 продемонстровано графік, що показує приклад функції шуму. Випадкове значення від 0 до 1 присвоюється кожній точці на осі X.

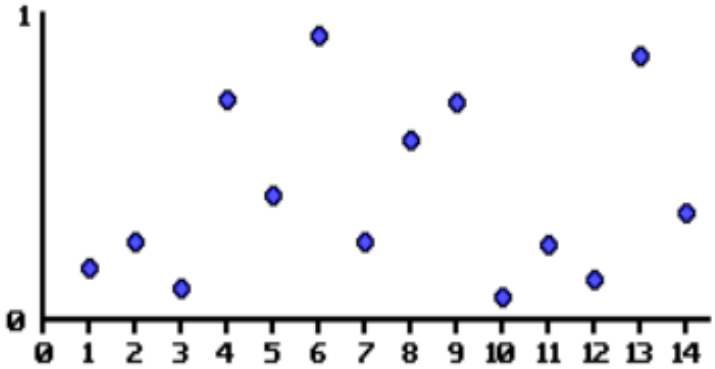


Рисунок 1.5 – Приклад функції шуму

З допомогою плавної інтерполяції між значеннями, можна визначити безперервну функцію (рис.1.6).

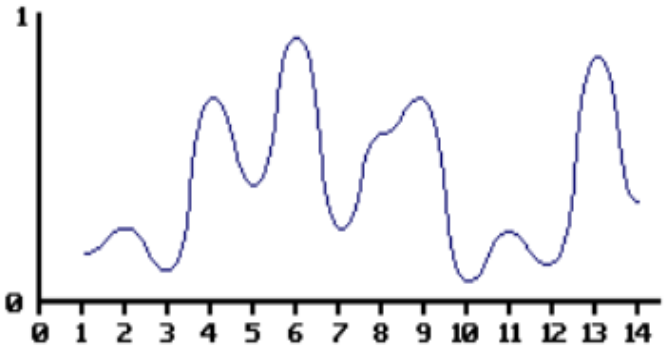


Рисунок 1.6 – Приклад безперервної функції

Таким чином, якщо взяти багато таких гладких функцій, з різною частотою і амплітудою і додати їх всі разом, можна створити добре зашумлену функцію. Це функція шуму Перліна.

Взявши наступні функції шуму (рис. 1.7) і зіставивши їх разом (рис. 1.8) отримуємо функцію шуму Перліна, яка має великі, середні і невеликі варіації.

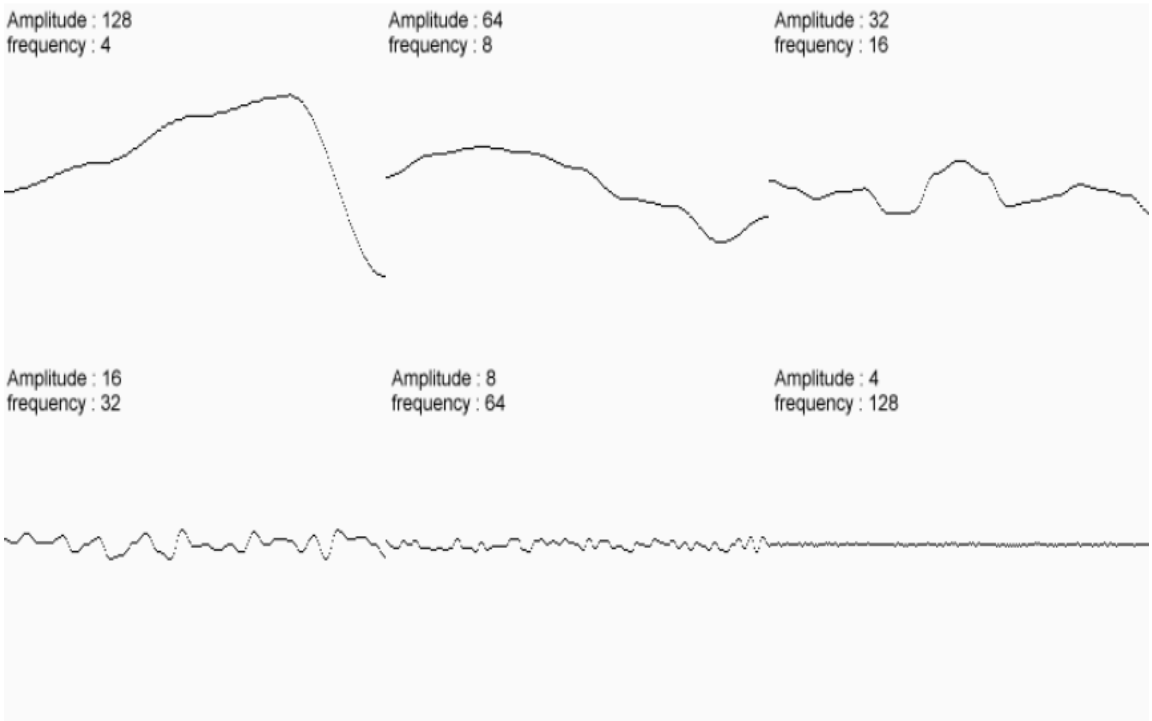


Рисунок 1.7 – Функції шуму

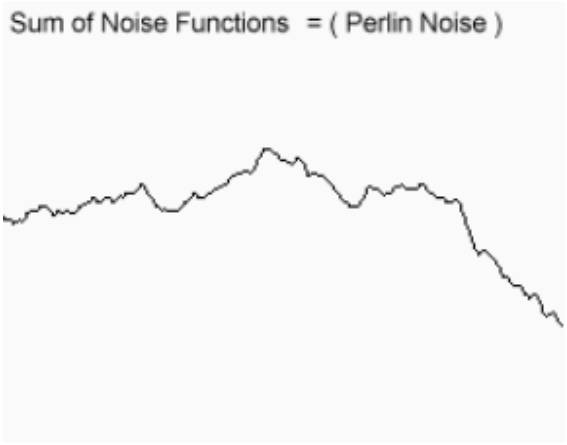


Рисунок 1.8 – Сумарна функція шуму (шум Перліна)

Можна виконати те саме для двох вимірів. Наприклад, на рис.1.9 представлені деякі функції шуму в 2D, а також результуючий шум, який як раз є згенерованою картою висот у вигляді зображення.

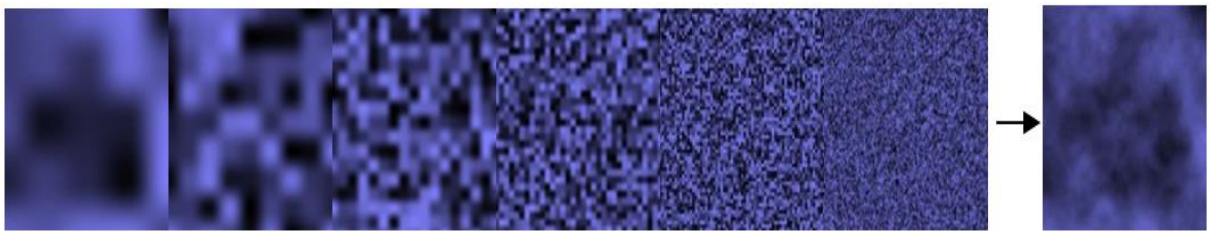


Рисунок 1.9 – Функції 2D шуму і результуючий сумарний шум

Наступним варіантом генерації ландшафту за допомогою карти висот є використання пагорбового алгоритму (Hill Algorithm) [25]. Цей ітераційний алгоритм містить декілька параметрів, зміна яких призводить до зміни характеристик місцевості. Алгоритм складається з таких кроків:

- Спочатку проводиться робота з плоскою місцевістю. Всі значення висоти ініціалізуються нулями;
- Далі вибирається випадкова точка на місцевості або біля неї і випадковий радіус між деяким заданим мінімумом і максимумом, які необхідно підбирати особливо ретельно, так як в залежності від цього можна отримати різну поверхню, наприклад більш грубу або гладку;
- Піднімається пагорб на місцевості з центром в точці з заданим радіусом;

- Далі відбувається повернення до кроку 2, який повторюється стільки разів, скільки це необхідно. Кількість обраних ітерацій впливає на зовнішній вигляд місцевості;
- Нормалізується ландшафт;
- Згладжуються долини.

Необхідно конкретизувати крок підняття пагорба на місцевості з центром в точці з заданим радіусом. В даному випадку пагорб – це куля і чим більше її радіус, тим вище висота пагорба. Математично це схоже на параболу, що обертається навколо центральної точки, яка досягає значення нуля в радіусі. Конкретно, враховуючи центральну точку (x_1, y_1) і радіус r, висота пагорба в точці (x_2, y_2), згідно з джерелом [40], еквівалентна формулі:

$$\boldsymbol{z = r^2 - ((x_2 - x_1)^2 + (y_2 - y_1)^2)}$$

На рис.1.10 зображено результат, представлений у вигляді місцевості з одним пагорбом посередині. Радіус можна розглядати як круг, де пагорб зустрічає землю.

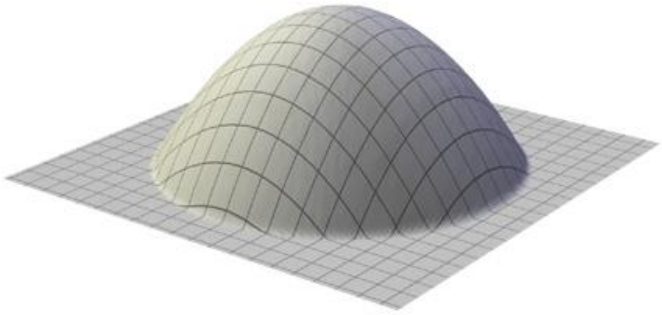


Рисунок 1.10 – Ізольований пагорб

Щоб створити цілісний ландшафт, потрібно багато разів додавати по декілька пагорбів. В процесі потрібно пам'ятати про дві речі. По-перше, треба ігнорувати негативні значення висоти. Рівняння, показане вище, дасть негативний результат, якщо точка, що обчислюється, знаходиться далі від центру, ніж радіус. Коли це відбувається, значення ігнорується. По-друге, кращим варіантом буде додавати отримане значення висоти до уже існуючого. Такий варіант дозволяє отримати більш правдоподібне, реальне

зображення, а не ідеально круглі пагорби (рис.1.11). Більша кількість ітерацій дає більш реалістичний результат, так як деякі рівні пагорбів не будуть відображені на менших ітераціях.

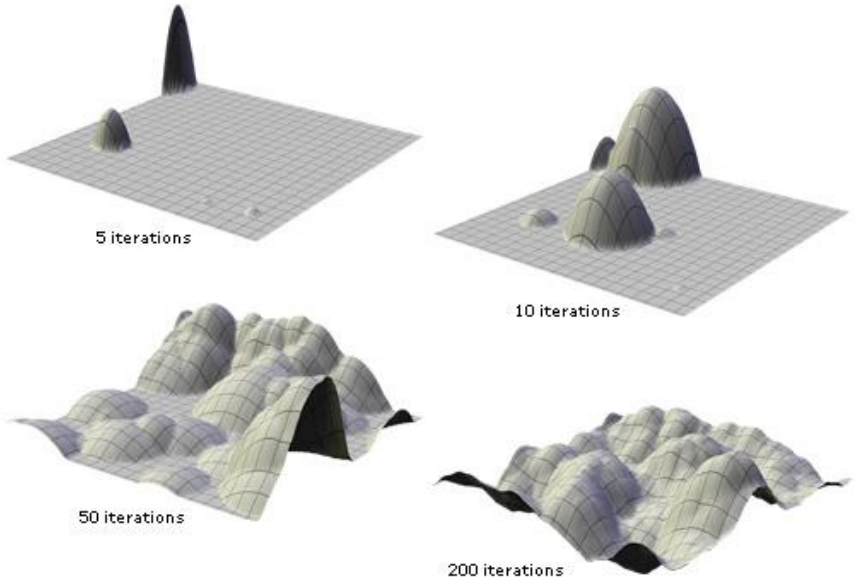


Рисунок 1.11 – Генерація на певних рівнях ітерації

Також одним із варіантів побудови карти висот є моделювання ерозії – моделювання фізичної поведінки рельєфу в часі [26]. Генерується довільна карта висот. Далі за допомогою великої кількості ітерацій моделюється зношування поверхні відповідно до часу (рис.1.12). Недоліком цього способу є проблема продуктивності.

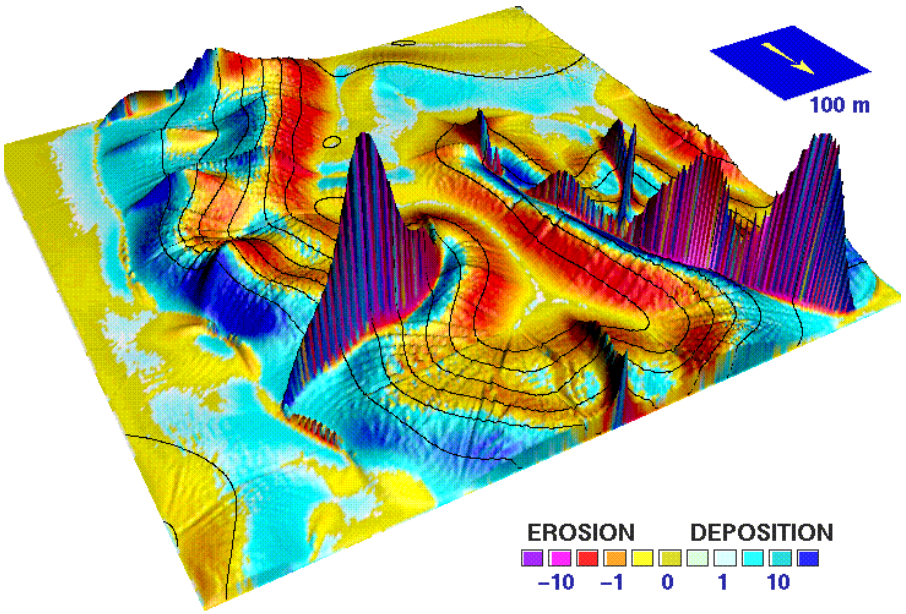


Рисунок 1.12 – Моделювання ерозії

Останнім розглянутим алгоритмом є Diamond Square. Він генерує карту висот для регулярної сітки, тоді як попередні алгоритми дозволяють робити це для нерегулярної сітки. Основна ідея алгоритму полягає в проходженні по сітці квадратами, що зменшуються і обчисленні висот в центрі квадрата та на серединах його сторін за значеннями в кутових точках з додаванням випадкового значення. Алгоритм складається з двох кроків «square» і «diamond» (рис.1.13).

Перший крок «square» полягає в наступному:

- Задаються висоти в кутових точках;
- Обчислюються значення в центральній точці квадрата шляхом усереднення значень кутових точок і додавання випадкового відхилення.

Другий крок «diamond» визначає висоту точок, що лежать на серединах сторін:

- Обчислюються значення на серединах сторін. Варто зазначити, що тут знаходиться середнє не лише двох точок «зверху» і «знизу» (якщо говорити про точках на вертикальній стороні), а і пара точок «зліва» і «справа» - тобто ще дві отримані на кроці «square» центральних точки. 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»;
- Розглядаються менші квадрати. Для них повторюються кроки алгоритму, поки квадрати не стануть потрібного розміру.

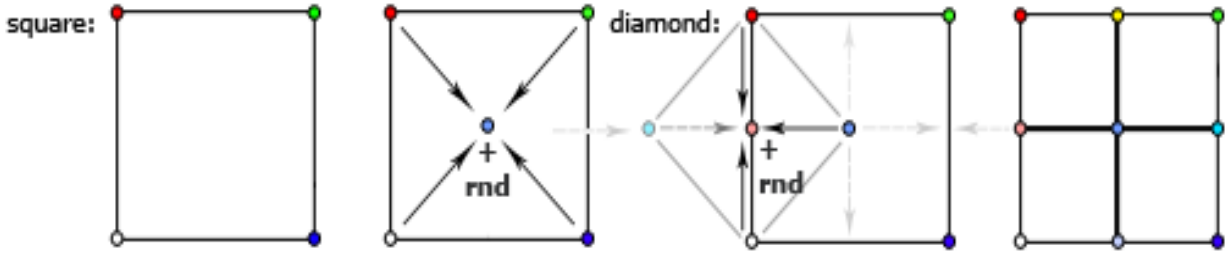


Рисунок 1.13 – Візуальне представлення алгоритму Diamond Square

1.4 Висновки

У сучасному світі найбільшу популярність здобувають ті ігри, чиє оточення виглядає максимально реалістично. Запорукою цієї реалістичності і її фундаментом є ігровий ландшафт.

Створення High realistic ландшафтів відбуваються за допомогою алгоритмів перетворення вхідних даних з побудовою на їх базі карти висот (HeightMap).

Для створення регулярної сітки висот використовують масив 2D значень. Інший спосіб зображення карти висоти – це зображення у градаціях сірого. Наступним способом є використання іррегулярної сітки вершин і їх зав'язків (тріангуляційна сітка). Третій спосіб представлення ландшафтів називається посегментною картою.

Для побудови карти висот використовують такі алгоритми, як:

- використання вбудованої функції Random;
- використання шуму Перліна (Perlin noise);
- використання пагорбового алгоритму (Hill Algorithm);
- моделювання ерозії – моделювання фізичної поведінки рельєфу в часі;
- використання діамантового (Diamond Square) алгоритму.

Техніка карт висот порівняно довго домінувала в поданні відкритих просторів, але цей підхід має очевидні обмеження:

- Не можна описати прямовисні скелі, печери, великі валуни;
- Можна контролювати точну висоту кожної точки, тільки якщо розмір сітки відповідає координатам текстури;
- Якщо чотири вершини, що визначають покадрову область, не перебувають на одній площині, то розкол між двома вершинами стане видимим. Зазвичай це відбувається на крутих скелях з краями, які не дотримуються одного напрямку.

Виявлення цих обмежень дало змогу з'ясувати, що з'явилася необхідність появи принципово нових методів генерації ландшафту в ігрових програмах.

2. ВОКСЕЛЬНА ТЕХНОЛОГІЯ, ЯК МЕТОД ГЕНЕРАЦІЇ ЛАНДШАФТУ В ГРОВИХ ПРОГРАМАХ

2.1 Опис методів і моделей воксельної технології

Полігональне моделювання було золотим стандартом в тривимірному моделюванні та поданні місцевості більше двадцяти років. Його розвиток задав рівень реалізму в комп'ютерній графіці, який раніше не був можливий при поданні тривимірних об'єктів, що, разом з ефективністю в області зберігання даних і рендерингу об'єктів, забезпечило популярність полігонального моделювання.

Однак, з більш нової сенсорною технологією, почала змінюватися, точність і доступність вхідних даних датчиків, що використовуються для моделювання об'єктів 3D-ландшафту, за останнє десятиліття збільшилася в кілька разів. У той же час відповідна вартість таких витрат знизилася. Були виявлені деякі властиві обмеження в полігональному моделюванні, які стали серйозною проблемою для цієї техніки.

Було розроблено безліч алгоритмів, що дозволяють отримувати дуже великі масиви висот з високою продуктивністю, що є необхідним для обслуговування багатьох типів додатків. Проте, обмеження на двовимірну область створює значні обмеження щодо характеристик рельєфу місцевості і ці обмеження стають менш прийнятними для деяких типів додатків, оскільки потужність графічного обладнання продовжує збільшуватися.

При моделюванні рельєфу з кроком в один метр можна працювати з використанням методів багатокутного моделювання, але проблеми виникають, коли цей проміжок доступний на рівні дециметрів або сантиметрового рівня. В певний момент необхідно використовувати так багато полігонів, необхідних для точної моделювання даних високої точності, що традиційна багатокутна модель розвалюється через підвищення вимог до розміру файлу та його рендерингу.

Зважаючи на недоліки техніки карт висот виникла потреба у новому методі генерації ландшафту. Таким альтернативним варіантом генерації

ландшафту є використання воксельної технології [27]. Воксельні карти дозволяють подавати місцевість з набагато більшою топографічною і топологічною складністю, дозволяючи створювати нові функції, які були неможливі при використанні полігонального моделювання.

Воксель (англ. Voxel) – це елемент об'ємного зображення, що містить значення елемента растра в тривимірному просторі. Вокселі є аналогами двовимірних пікселів для тривимірного простору (рис.2.1). В свою чергу пікселі (PICture ELements) – це спосіб поділу 2D-простору (наприклад, зображення або карти висоти) на однорідні осередки, зазвичай квадрати. Аналогічно, вокселі поділяють 3D-простір на однорідні тривимірні осередки, зазвичай куби. Вокселі можуть використовуватися для зберігання високо деталізованих ландшафтів, але мають відносно високі вимоги до зберігання. У порівнянні з полігональним моделюванням, таке подання може дозволити більш точне фізичне моделювання, оскільки кожен воксель може мати унікальні фізичні характеристики. Результуюча якість використання вокселів, як і пікселів, полягає в тому, що місце розташування вокселя явно не зберігається як набір координат XYZ, а скоріше визначається його відносним положенням до інших вокселів і початком координат набору даних. Це сильно відрізняється від полігонального моделювання де координатне розташування кожного кута трикутника має бути явно збережено. Єдиний розмір, як вокселів, так і пікселів на осередках дозволяє використовувати таку ефективну просторову прив'язку [27].

Також основними перевагами вокселів є те, що, як і більшість даних датчиків, вони засновані на растрах, і на відміну від карт висоти, вони дійсно 3D. Іншими словами, вокселі можуть моделювати навіси, печери і пористі структури – жодна з яких не може бути змодельована з картами висот 2,5 D, і всі з яких представляють собою проблеми для автоматизованих методів багатокутного моделювання.

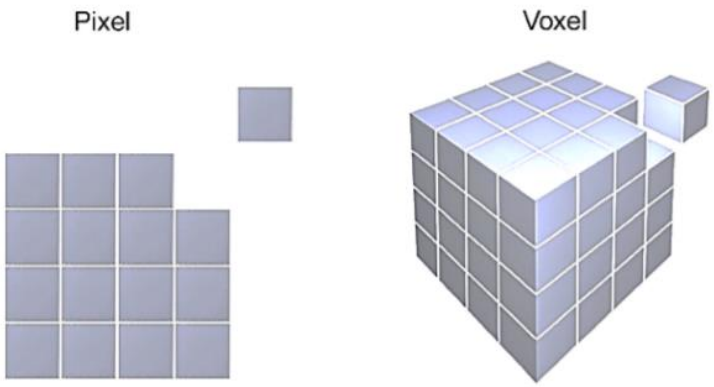


Рисунок 2.1 – Представлення пікселя (Pixel) і вокселя (Voxel)

Використання вокселів бере свій початок в медичній візуалізації. У 1973 році комп'ютерна томографія на рентгенівському знімку (КТ) була введена Годфрі Хаунсфілдом [28]. Це було одним з перших застосувань об'ємного моделювання в медичній візуалізації. Г. Хаунсфілд отримав Нобелівську премію з фізіології в медицині в 1979 році (разом з Алланом Кормаком) за його новаторську роботу в комп'ютерній томографії.

Саме Voxel-рендеринг місцевості вперше вийшов на комерційний ринок відеоігор в 1992 році з введенням Comanche: Maximum Overkill. Це була перша комерційна гра моделювання польоту, яка застосувала технологію вокселів і забезпечила набагато більш реалістичне і деталізоване уявлення місцевості, ніж те, що було можливо з векторною графікою того часу.

Перевагами використання вокселів є:

- Безперервні 3D-дані. Вокселі – це майже єдиний ефективний спосіб зберігання безперервних даних про приховані характеристиках місцевості;
- Легка модифікація. Стиснені дані вокселя можуть бути легко змінені;
- Розширені можливості рельєфу. Можна створювати печери і тунелі;
- Цікава генерація ландшафту. Наприклад, у грі «Minecraft» це відбувається шляхом накладення шумових функцій і градієнтів з зумовленими характеристиками місцевості (дерева, підземелля).

Використання вокселів для генерації ландшафту має і свої недоліки. По-перше, візуалізація таких даних є нетривіальним завданням і окремим предметом безлічі досліджень. По-друге, як уже згадувалося, виникають проблеми через великі вимоги до пам'яті комп'ютера. Справа в тому, що карту висот можна розглядати як одне єдине зображення, тоді як воксельні дані представляють собою N таких зображень, де N - максимальна висота об’єму, що моделюється.

Розглянемо дві найбільш поширені воксельні моделі. Перша модель – це сітчаста модель вокселя, в якій розглядають об'ємний набір даних як стек окремих вокселів з кожним воксельним зрізом або сіткою. Обробка і запит цієї воксельної моделі даних може виконуватися зрізом до тих пір, поки всі зрізи не будуть оброблені. Ця модель демонструє ефективність в обробці, оскільки обробляється тільки один зріз за раз. Однак це не дозволяє відбуватися стисненню або агрегації подібних вокселів в менш щільні представлення, як це роблять інші моделі даних вокселів. Кожне місце в межах розмірів XYZ місцевості буде містити воксель, включаючи області без даних, як показано на рис 2.2 нижче.

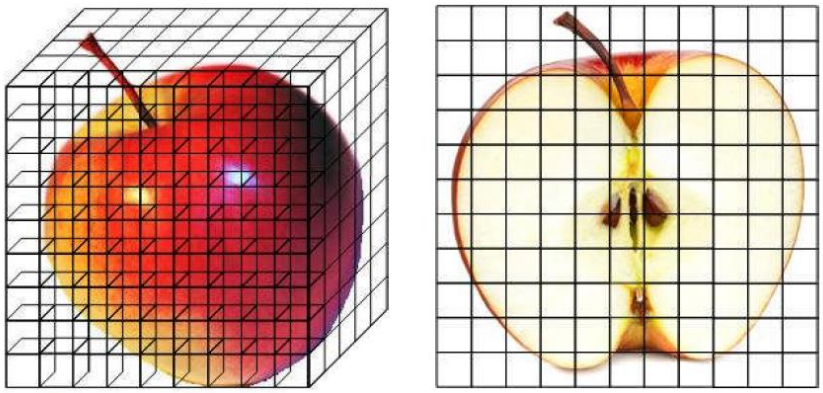


Рисунок 2.2 – Сітчасті вокселі (зліва), поперечний переріз (праворуч)

Друга модель – це розріджена модель вокселів, в якій зберігаються тільки ті вокселі, які фактично містять інформацію. Розглянемо модель вокселя, в якій кожна клітинка зберігає чотири значення: червоний, зелений, синій і непрозорість. Припустимо, ми хочемо представити яблуко. Модель з прив'язкою до сітки буде мати форму куба достатнього розміру, щоб

закрити яблуко. Кожен воксель в кубі буде містити значення кольору / прозорості, навіть якби воно було (0,0,0,0), що представляє «повітря» або «вакуум». Важливо відзначити, що всі вокселі вимагають однакового обсягу зберігання незалежно від того кодують вони «повітря» або якусь частину яблука. Воксельна модель не матиме форму яблука. «Повітряні» вокселі просто не існують, як показано на рис. 2.3, що представляє ще більшу економію для порожніх і пористих структур. Якщо припустити, що внутрішня частина яблука не існує, можна заощадити значну кількість вокселів тільки шляхом моделювання його деталей.

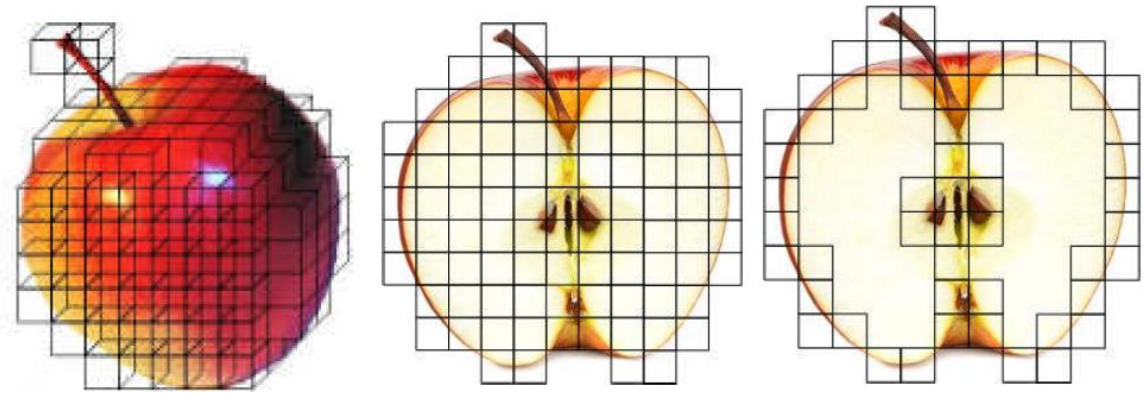


Рисунок 2.3 – Розріджені вокселі (зліва), поперечний зріз (центр) і «пусте» яблуко (праворуч)

Щоб в ряді випадків зменшити як обчислювальне навантаження, так і кількість займаної пам'яті, пропонується використання ще однієї моделі – воксельного октодерева (Octree voxel), яка складається зі структури даних, в якій кожен внутрішній вузол має до восьми дітей (рис. 2.4). Це по суті спосіб стиснення або поділу об'ємного простору шляхом об'єднання колекцій ідентичних і суміжних вокселів в більші агрегати [5], що дозволяє збільшити стиснення даних, особливо в областях, які є однорідними або областями, де немає даних. При моделюванні місцевості «порожні» вокселі часто можуть становити більше 95% вокселів в наборі даних, тому економія простору може бути істотною. У сітчастих і розріджених моделях вокселів ядро буде заповнено сусідніми вокселями однакового розміру і всі вони матимуть однакове значення кольору.

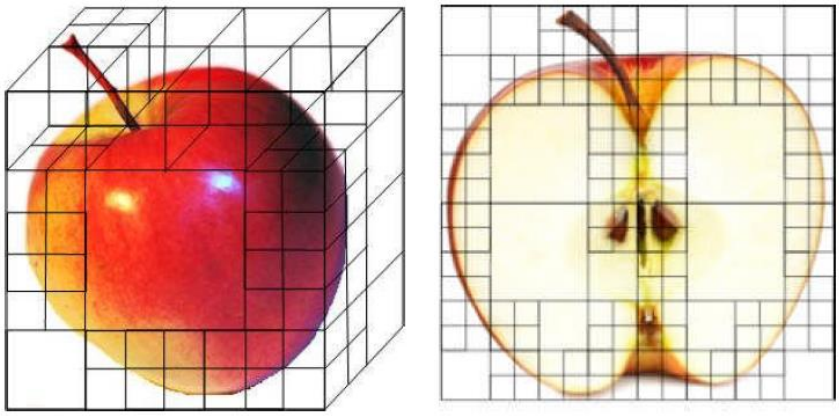


Рисунок 2.4 – Octree voxel (зліва), поперечний переріз (праворуч) (зліва), поперечний зріз (праворуч)

Подання ландшафту у вигляді вокселів має перевагу, яка дозволяє явно створювати складні топології місцевості, уникаючи проблем самопересічення, пов'язаних з даними на поверхні. Основний рівень деталізації набору даних безпосередньо залежить від розміру одного вокселя. Недоліком використання воксельного представлення ландшафту є те, що використання цих елементів з дискретним розміром по своїй суті обмежує максимальний рівень деталізації. Можна адаптивно збільшити деталі (тобто зменшити розмір вокселів) в деяких підвибірках набору даних. Однак для того, щоб це адаптивне збільшення мало сенс, засіб візуалізації також має використовувати підвищений рівень деталізації для цих підвибірок набору даних.

Звернемося до воксельної моделі. Об'ємні дані в загальному і в рельєфних даних, зокрема, піддаються стисненню. Структури даних просторового дерева по своїй суті забезпечують підвибори, які можуть бути перевірені на однорідні, але при цьому виконується типова логарифмічна складність і тому вони добре підходять для подання великих наборів даних місцевості. Порівняємо три структури даних на основі дерева [29] з точки зору споживання пам’яті і швидкості, а саме:

- Sparse voxel octree – розріджене воксельне октодерево;
- Point-region octree – октодерево точкової області;
- kd-tree – kd-дерево.

Структури даних, які не орієнтовані по осі, не розглядалися для оцінки, оскільки оновлення дискретного набору даних вокселів має бути настільки простим, наскільки це можливо, щоб забезпечити інтерактивність. Ці три структури були обрані із-за їх відносної популярності в літературі і відповідності бажаним властивостям структури даних.

Дані три структури даних засновані на рекурсивному підрозподілі набору даних. Такий підрозподіл триває доти, поки не буде досягнута умова завершення, наприклад, суб’єкт є однорідним або суб’єкт досяг занадто малих розмірів для подальшого поділу. Якщо такий підвибір містить однорідні дані, весь суб’єкт можна звести до одного вузла дерева. Рекурсивно розділені томи утворюють ієрархію, яка представлена деревовидною структурою. Кожен вузол дерева визначає і обмежує обсяг його дочірніх елементів.

Розглянемо підтипи об’ємного дерева: Pointerless tree (безвказівникове дерево) і Pointer tree (дерево з вказівником). Pointerless tree створюється без традиційних вказівників, що дозволяє уникнути накладних витрат, пов'язаних з вказівниками у деревах вказівників. Таке дерево приблизно аналогічне одновимірній структурі, використовуваної для сортування масиву, при цьому окремі елементи масиву зіставляються з деревоподібними позиціями. Pointerless trees не можуть використовувати локалізовану гомогенність набору даних і, отже, вимагають ще більшого обсягу пам'яті, ніж тривимірне уявлення масиву. Використання вказівників для побудови дерева дозволяє однаково направити однакові області простору на єдиний вузол дерева за рахунок накладних витрат вказівника.

Щодо умов завершення, створення дерева триває рекурсивно до тих пір, поки не будуть задоволені певні умови завершення. Дві умови завершення розподіляються по всім трьом структурам даних. Перша полягає в тому, що підрозподіл зупиняється, коли даний об'ємний є однорідним. Тоді однорідний суб’єкт може бути представлений одним

вузлом. Якщо суб’єкт неоднорідний, то вокселі в суб’єкті явно зберігаються у вигляді тривимірної матриці, що утворює воксельний кубоід.

Кожен внутрішній вузол дерева може містити до восьми дітей. Якщо один з цих дочірніх вузлів є листовим вузлом і представляє тільки один воксель, службова інформація покажчика дочірнього вузла буде вище, ніж фактична пам'ять, яка використовується для зберігання вокселя. Тому більш важливо, щоб листові вузли зберігали більш одного вокселя в тривимірному масиві вокселів. Таким чином, додаткова умова завершення використовується для запобігання подальшого поділу підвезень, менших, ніж попередньо вибраний обсяг.

2.2 Опис структур воксельних даних на основі дерева

У розрідженому воксельному октодереві кожен суб’єкт рекурсивно поділяється на вісім октантів. Кожен суб’єкт розділяється по центру, причому кожному октанту присвоюється певне число (рис.2.5).

Оскільки кожен суб’єкт ділиться, ґрунтуючись виключно на своїх розмірах, які його обмежують, немає необхідності зберігати будь-які дані, що стосуються підрозділа; розміри кожного вузла можуть бути отримані безпосередньо з його положення в дереві. Велика кількість супервузлів може бути створена просто для подання однорідних підвиборів.

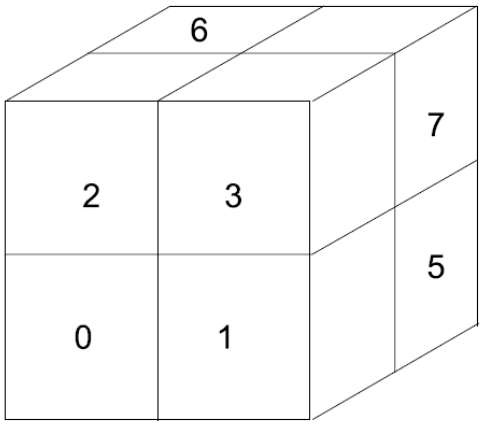


Рисунок 2.5 – Приклад нумерації октетів октодерева

Вузли октодерева завжди поділяються по центру, причому різні розміри обмежуючого обсягу ідентичні (або майже ідентичні). На рис.2.6 (a) показано, як двомірна область підрозділяється для зберігання однієї точки даних в однорідній області. На рис.2.6 (b) показано результуючу деревоподібну структуру з кольоровим листовим вузлом, що зберігає тривимірний воксельний кубоід, що містить точку даних.

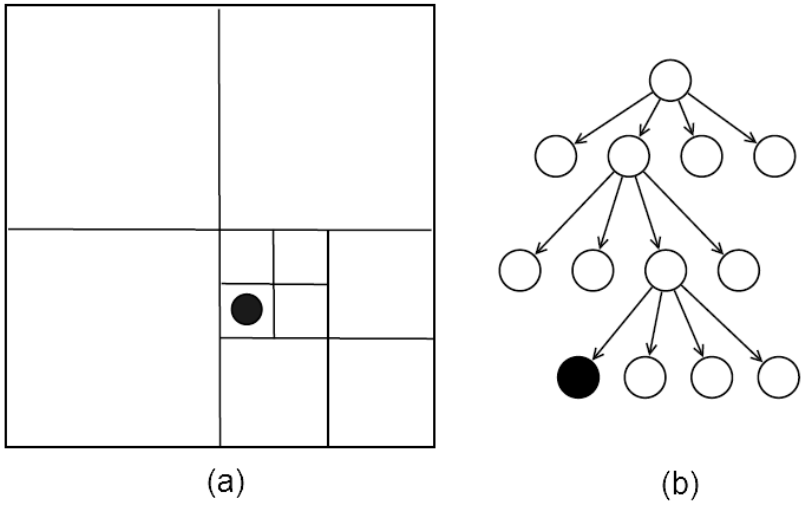


Рисунок 2.6 – Двовимірне квадро-представлення. Тривимірне октодерево аналогічне

Для Point-region octree область, представлена вузлом в октеті, підрозділяється через її геометричний центр незалежно від базового набору даних. Таким чином, однорідні характеристики ландшафту не завжди належним чином використовуються. Наприклад, якщо тільки нижня третина допоміжного обсягу неоднорідна, може бути доцільним переміщати положення розщеплення вниз.

Point-region octree намагається поліпшити октет, маючи потенційно змінюване положення розщеплення. Об’єми, що обмежують, для кожного з восьми дітей вузла можуть бути однозначно визначені з цієї позиції поділу. Це означає, що необхідно зберегти тільки три значення для координат розщеплення, а не шість значень, необхідних для зберігання розмірів об’єма, що обмежує. Октанти дитини вузла нумеруються так само, як і в розрідженому воксельному октодереві (рис.2.5).

Вузли Point-region octree не завжди поділяються по центру, що потенційно дозволяє використовувати вузькі плити з одним відносно великим розміром і один, відносно невеликий, розмір. Якщо порівняти рис.2.7, де (a) – Point-region octree для представлення однієї точки, (b) – структура квадрантів Point-region octree, створена під час поділу, точка зберігається в кольоровому листковому вузлі, з рис.2.6, то можна побачити, що потрібна менша кількість підрозподілів, так як підвибори можуть бути розташовані безпосередньо поруч з точкою даних, тоді як кожен листової вузол має однакому мінімальну площу.

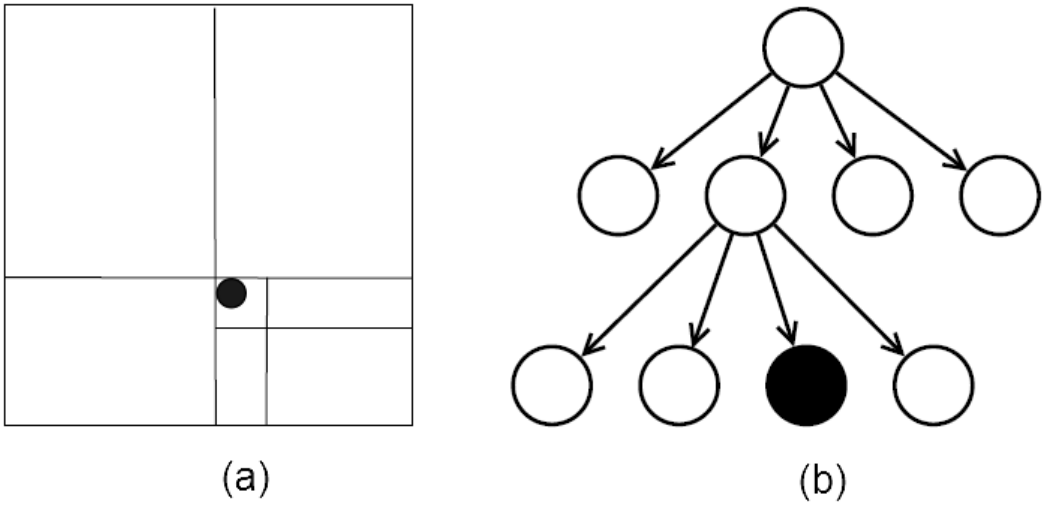


Рисунок 2.7 – Point-region octree квадро-представлення. Point-region octree аналогічне

Kd-tree являє собою двійкове дерево, яке ділить обмежуючий об’єм на два підлеглих по осі згладжування. Кожен внутрішній вузол розбивається по певній осі в заданому положенні. Вісь може бути обрана як функція висоти вузла в дереві таким чином, що вісь поділу не повинна бути явно збережена для кожного вузла дерева.

В якості альтернативи, вісь поділу може бути обрана для оптимізації вимог до зберігання. Оскільки у kd-дерева тільки дві дитини, потрібно менше пам'яті для кожного вузла, ніж для Sparse voxel octree або Point-region octree реалізацій. Однак kd-дерева вимагають більшої кількості вузлів а, отже, потенційно довшого обходу дерева.

На рис.2.8 показано, як kd-дерево підрозділяється на представлення специфікатора. Спочатку kd-дерево розділяється уздовж осі x, далі відбувається поділ уздовж осі y, осі x і нарешті осі y. Цей поділ триває до тих пір, поки точка даних не буде збережена як листовий вузол. Відповідно на рис.2.8 (a) зображено kd-tree для представлення однієї точки, а на рис.2.8 (b) – kd-tree структура, створена під час поділу. Точка зберігається в кольоровому листовому вузлі. Вісь, уздовж якої був розділений кожен вузол, вказана всередині цього вузла.

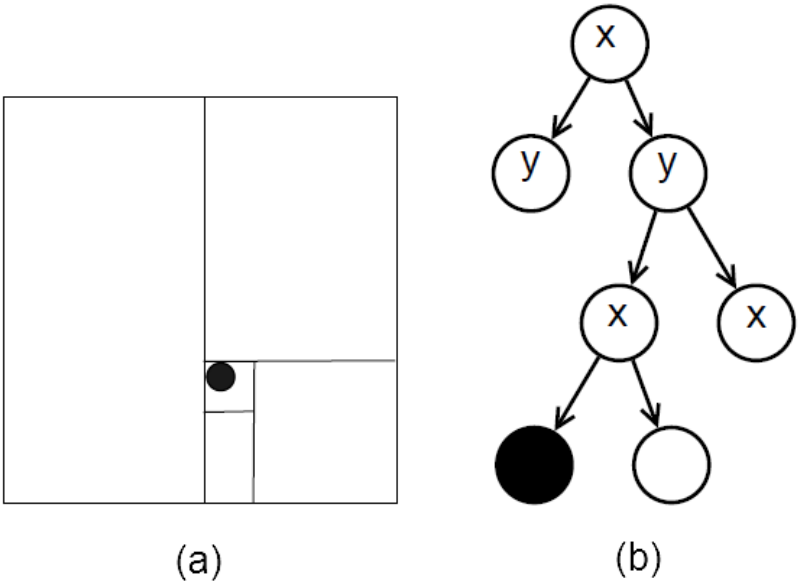


Рисунок 2.8 – Представлення kd-tree, яке розбивається вздовж додаткової осі, але в цілому ідентичне з попередніми

2.3 Оцінка структури дерева

Три структури даних оцінюються для перевірки інтерактивності і зберігання даних. Для тимчасового набору даних використовуються два типи ландшафту і вимірювання використання пам'яті.

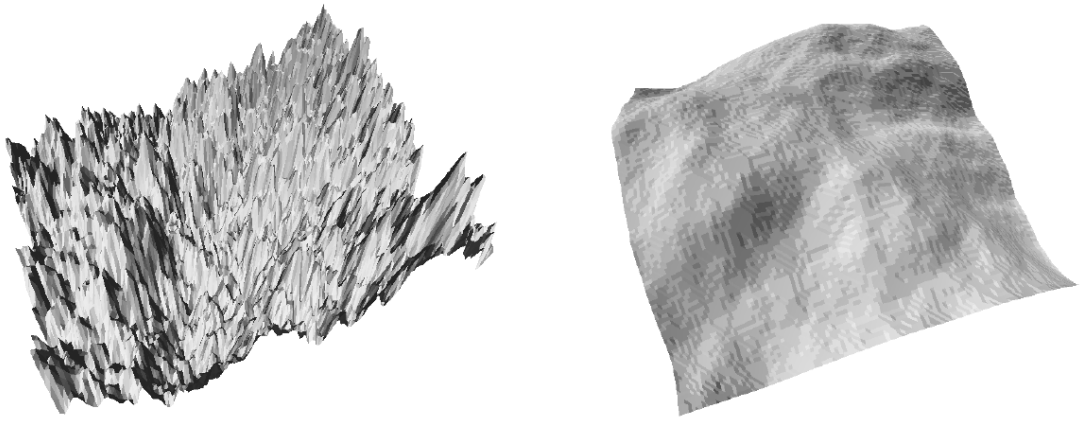
приклад місцевості

Створимо шум Перліна за допомогою псевдовипадкового шумового ефекту PerlinNoise(x). Шум Перліна має перевагу в створенні когерентного шуму, на відміну від більш прямої псевдовипадкової функції. Шум в точці

x створюється за допомогою суми масштабованих результатів шуму Перліна:

$$Noise(x) = \sum_{i=0}^{n-1} \frac{PerlinNoise(\beta^i x)}{\alpha^i},$$

де параметр n – це кількість октав Noise(x), які повинні бути об'єднані для створення PerlinNoise(x), а параметр α використовується для масштабування кожної наступної октави, параметр β використовується для управління частотою шуму. На рис.2.9 показано приклад місцевості, створеної з використанням шуму Перліна із зазначеними аргументами для (a): α = 1.5, β = 2, n = 7. Для (b): α = 3, β = 2, n = 7.



(a) (b)

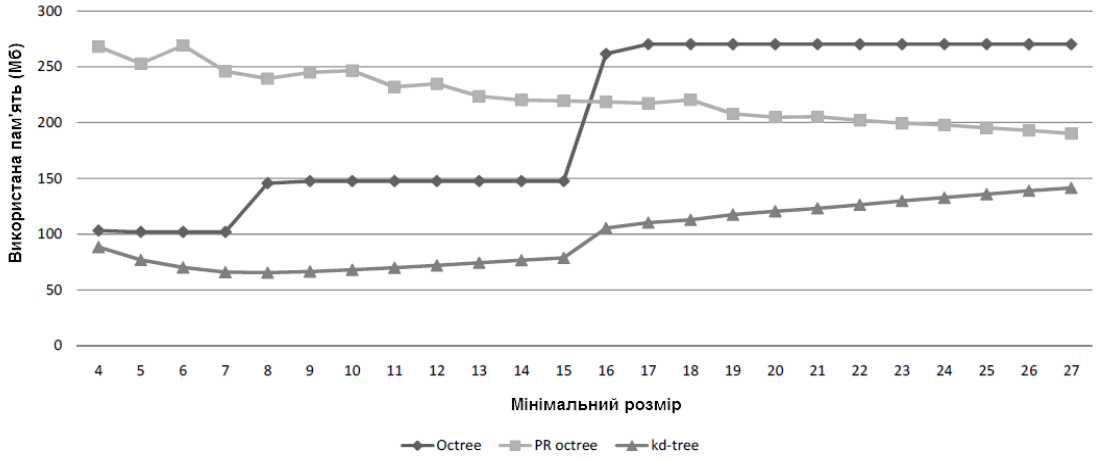
Рисунок 2.9 – Приклад 128 × 128 полів висот, згенерованих за допомогою шуму Перліна

У вимогах для зберігання, основна мотивація використання структури просторових даних для зберігання дискретних вокселів полягає у використанні однорідності одиничних об’ємів для забезпечення стиснення однорідності. Це призводить до того, що використовується менше пам'яті, ніж у випадку прямого тривимірного воксельного масиву. Вимірюємо використання пам'яті шляхом підсумовування розмірів окремих вузлів дерева з розмірами пам'яті, що використовується для зберігання кубів вокселів в листових вузлах.

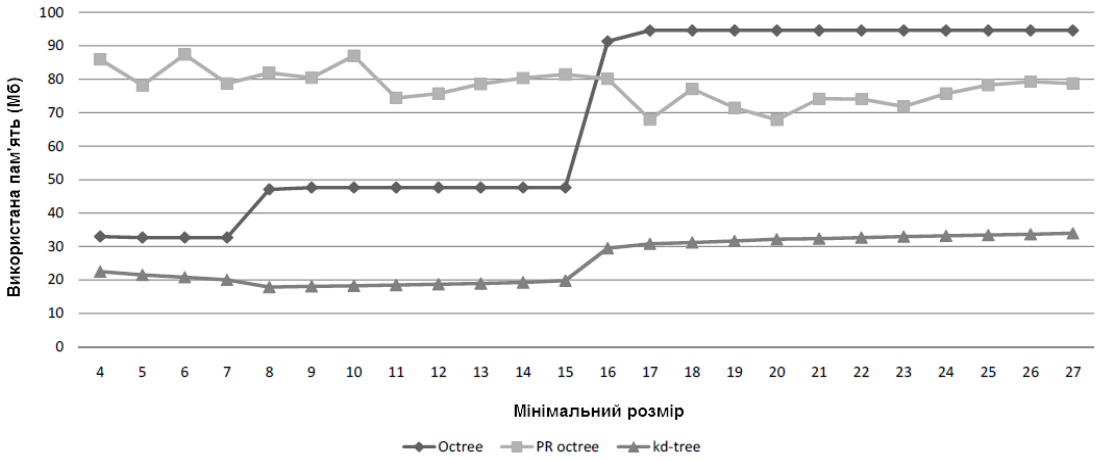
Вимоги до пам'яті кожної з трьох структур даних залежать від умов завершення, використовуваних при створенні дерева. Концепція

мінімальної розмірності використовується для визначення довжини сторони суб'єкта, нижче якої вузол надалі не поділяється.

Шорсткість графа октетів (рис.2.10) обумовлена методом створення октодерева, який на кожному етапі рекурсії зменшує кожен розмір об'єму, що його обмежує. Наприклад, якщо поточний обмежуючий об'єм становить 9 × 9 × 9, а умова припинення мінімального розміру дорівнює 8, об'єм буде розділений на вісім суб'єктів, хоча 9 × 9 × 9 об'єм трохи більший, ніж умова завершення. Тобто однакові розміри вузлів генеруються для мінімальних розмірів 8-15.



(a) Воксельний ландшафт на основі поля висот згенерованого при α = 1.5



(b) Воксельний ландшафт на основі поля висот згенерованого при α = 3

Рисунок 2.10 – Вплив вибору мінімального розміру для 1024×1024×1024 даних

По мірі збільшення мінімального розміру, вузли листя октодерева починають збільшуватися за розміром, вимагаючи виділення більших

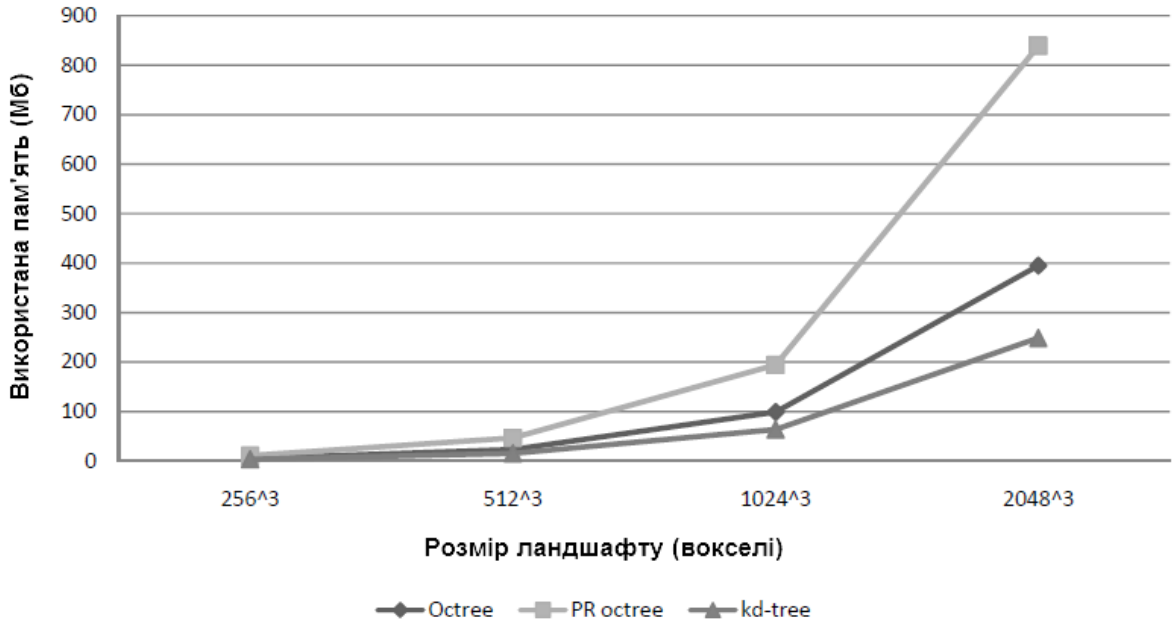
об'ємів масивів вокселів. Результати продемонстровані на зростаючій діаграм октодерева на рис.2.10. Спочатку kd-tree зменшується при використанні пам'яті зі збільшенням мінімального розміру. Це пов'язано з тим, що кількість згенерованих вузлів зменшується швидше, ніж збільшується простір, необхідний для кубиків вокселя. У міру подальшого збільшення мінімального розміру, витрати на розподіл вокселів починають перевищувати пам'ять, збережену за рахунок меншої кількості вузлів. Нерівномірність тенденції Point-region octree показує, що її вимоги до зберігання чутливі до різних мінімальних розмірів.

Характеристики, аналогічні наведеним на рис.2.10, зустрічаються для наборів даних 256^3, 512^3 і 2048^3.

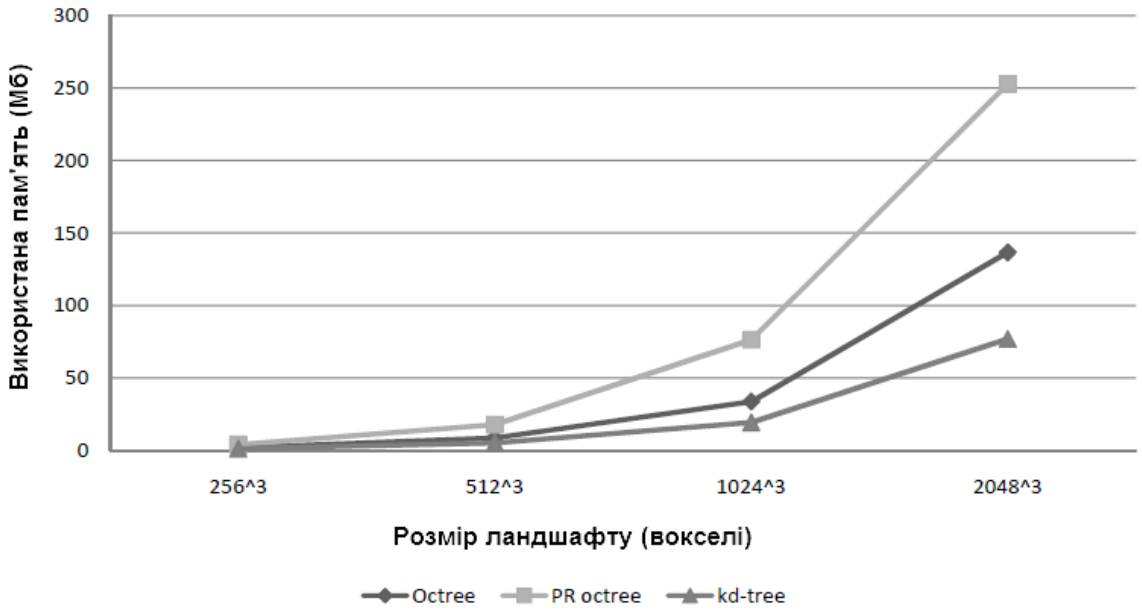
Для розміру ландшафту по мірі збільшення розміру рельєфу число дискретних вокселів в цьому підвиводі збільшується, що в загальному випадку призводить до того, що для представлення ландшафту потрібно більше вузлів і кубиків вокселів.

Для кожної з трьох структур даних було розраховано мінімальне використання пам'яті, відповідне оптимальному вибору сховища мінімального розміру для чотирьох різних розмірів рельєфу (рис.2.11). Ландшафт, аналогічний рис.2.9 (a), вимагає більше пам'яті, ніж рельєф, аналогічний рис.2.9 (b), що видно з порівняння використання пам'яті для рис.2.11 (a) і рис.2.11 (b).

Незалежно від характеристик рельєфу, три структури мали послідовне відносне використання пам'яті. Тому припускається, що для типової штучно створеної місцевості, заснованої на еліптичних полях, дерево kd-tree буде використовувати меншу пам'ять, ніж Sparse voxel octree або Point-region octree.



(a) Воксельний ландшафт на основі поля висот згенерованого при α = 1.5



(b) Воксельний ландшафт на основі поля висот згенерованого при α = 3

Рисунок 2.11 – Пам'ять, яка використовується як ландшафт, що збільшується

Щоб оцінити продуктивність (виконання) вимірюється час, необхідний для побудови деревовидної структури вокселя, яка представляє собою висоту, час, необхідний для часткового алгоритму вилучення сітки, і час, необхідний для зміни об'єму даних. Зміна підвиконання великої площі – це повільна операція і тому вважається неінтерактивної операцією. Диференціальні типи ландшафту не впливають на відносні експлуатаційні

характеристики для різних структур даних, а рандомізовані висоти, що генеруються з ідентичними вхідними значеннями, рівними значенням, зазначеним на рис.2.9 (b), використовувалися для оцінки продуктивності (виконання).

Для зчитування даних, щоб вказати точку всередині суб'єкта внутрішнього октодерева або вузла Point-region octree, октант, в якому знаходиться ця точка, повинен бути визначений шляхом порівняння положення точки з положенням розщеплення.

Для kd-tree вісь і позиція поділу поточного вузла порівнюються з відповідним компонентом потрібної точки. Якщо він більше, ніж компонент необхідної точки, наступний дочірній октант – це лівий вузол, інакше – правий вузол.

Для Sparse voxel octree і Point-region octree всі три компоненти потрібної точки порівнюються з компонентами положення розщеплення батьківського вузла, щоб сформувати бітове кодування відповідного дочірнього вузла. Потім це бітове кодування можна використовувати для вибору відповідного дочірнього октанта (рисунок 2.5).

Коли досягнуто вузол листа, цей вузол повинен містити потрібну точку. Якщо це однорідний вузол, шукана точка має те ж значення, що і решта цього підвибору. Якщо це неоднорідний листовий вузол, це значення безпосередньо визначається з тривимірного воксельного кубоіда.

З даних вокселя витягується полігональна сітка, використовуючи алгоритм Marching Cubes. Цей алгоритм ділить простір набору даних на еквівалентні куби і відображає значення в кожному з кутів куба.

Вертикальна вісь на рис.2.12 представляє log10 числа секунд, використовуваних для відбору всіх кутів куба для різних розмірів ландшафту. Це викликано збільшенням числа вузлів дерева. Kd-дерево є найповільнішим з трьох структур через його велике число вузлів дерева.

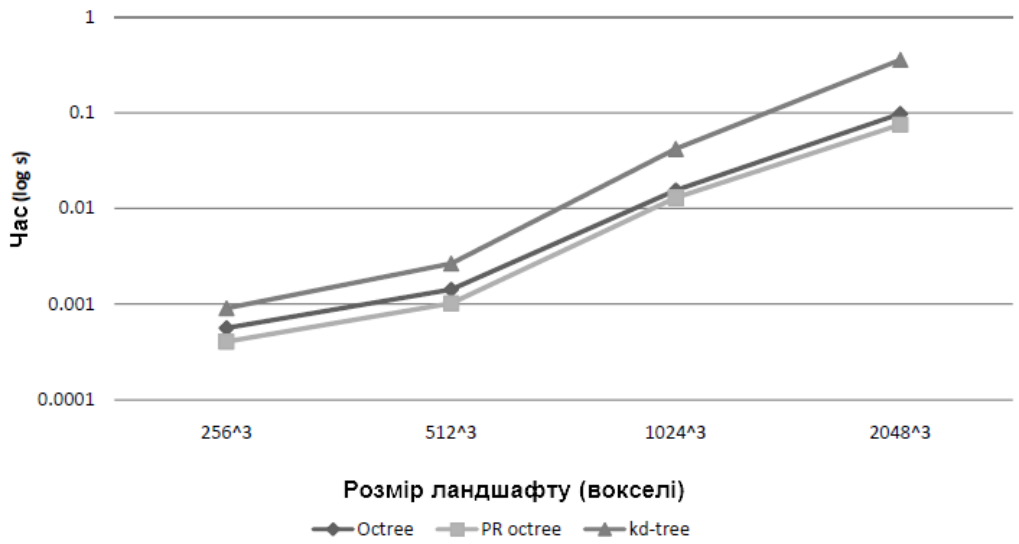


Рисунок 2.12 – Демонстрація алгоритму Marching Cubes з вимірюванням вибірки кутів

Пошук даних у всіх трьох структурах вимагає O(n) операцій в гіршому випадку, де n – загальне число вузлів в дереві. Якщо дерева збалансовані, для даних потрібно O(log n) операцій. Ця верхня межа залежить від n, який, в свою чергу, безпосередньо залежить від деревовидної структури і евристики створення, яка використовується на певному макеті ландшафту. Через відносну диференціацію визначення більш чітких меж були реалізовані і емпірично зіставлені різні структури даних.

Незалежно від того, яка специфічна деревоподібна структура використовується, до того, як вокселю в точці p можна було присвоїти значення d_1, повинен бути визначений нижній вузол дерева N, що містить p.

Якщо N є однорідним суб'єктом типу d_2, де $d_2 \neq d_1$, то N повинен бути далі поділений до тих пір, поки не буде досягнуто мінімальне значення розміру вокселя (якщо $d_1 = d_2$, подальша дія не потрібна, так як N вже зберігає значення d_1 при p). Цей процес поділу залежить від типу використовуваної деревовидної структури.

Октодерево завжди ділиться через його центр, так що вузол листа октодерева, що містить p, повинен бути рекурсивно розділений до досягнення мінімальних розмірів.

Вузли Point-region octree і вузли kd-tree поділяються так, що навколо p формується дочірній суб’єкт мінімальних розмірів. Це означає, що вставка довільної точки в Point-region octree октет або kd-tree буде потребувати не більше двох або шести підрозділів відповідно, тоді як октет може потребувати набагато більше, в залежності від положення в дереві.

Після того, як знайдено відповідний воксельного блок, відповідні зміни зроблені. Якщо весь блок вокселя буде однорідним після зміни, блок вокселів може бути представленим як єдиний вузол дерева, і пам'ять, необхідна для блоку вокселів, може бути звільнена.

Воксельне різьблення використовується для застосування об'ємного оператора до підвибірки набору даних вокселів. Оператор, що виконує різьблення, замінює всі вокселі в наборі даних, які перекриваються з оператором суб’єкта різним видом вокселів. Цей інструмент деформації дозволяє користувачеві модифікувати невеликі підвибори ландшафту.

На рис.2.13 показано час, необхідний для зміни типів вокселів підвибору 200^3 суб’єкта в центрі набору даних для кожної структури даних. У той час як Sparse voxel octree і Point-region octree мали подібні експлуатаційні характеристики, kd-tree виконувалося повільніше по мірі збільшення розміру ландшафту. Вузли Sparse voxel octree і Point-region octree вимагають менше підрозділів, ніж вузли kd-tree, коли повинні створюватися дочірні вузли.

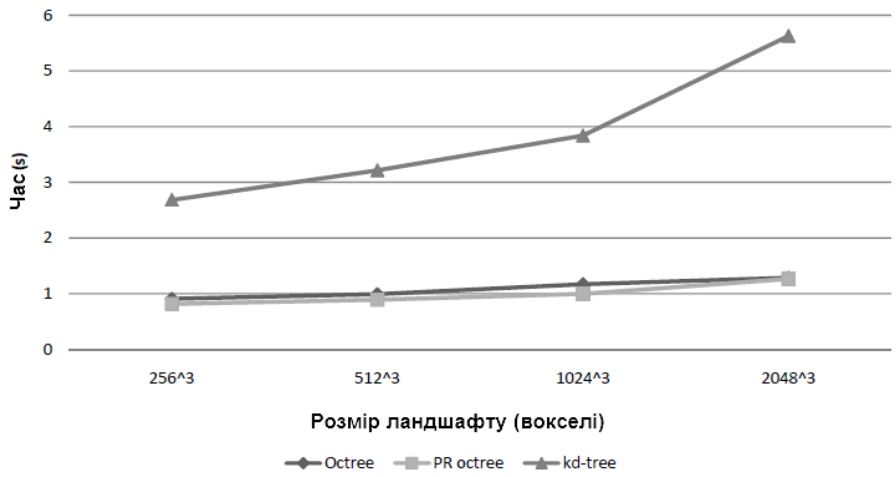


Рисунок 2.13 – Продуктивність (виконання) воксельної різьби

При ініціалізації, генерація деревовидної структури, що відповідає висоті, може важко обраховуватися. Point-region octree і створення kd-tree відбувається повільніше, ніж створення Sparse voxel octree через додаткові перевірки однорідності.

Октодерево створюється шляхом дослідження чи представляє поточний обмежуючий об’єм однорідних даних. Якщо це не так, обмежуючий об’єм рекурсивно підрозділяється через його центр до тих пір, поки не буде досягнута умова завершення.

Point-region octree і kd-tree прагнуть мінімізувати використання пам'яті, вибираючи позицію поділу для кожного підрозділу на основі збереженого набору даних. Визначення оптимального положення розщеплення є NP-повною проблемою [30] і, таким чином, положення розщеплення вибирається з використанням евристики. Ефективність структури даних і оновлення даних безпосередньо залежать від цих евристик.

Для вузлів Point-region octree позиція розщеплення вибирається як точка всередині обмежуючого об’єму, яка призводить до єдиного найбільшого однорідного суб’єкта. Це положення розщеплення знаходиться шляхом ітерацій по трьох вимірах обмежуючого об’єму. Якщо не знайдено жодної позиції розщеплення, яка створює однорідний суб’єкт, обмежуючий об’єм розбивається по його центру.

Аналогічна евристика використовується для kd-tree, за винятком того, що обсяги поділяються тільки уздовж однієї осі. Вісь, по якій поділяється вузол, вибирається на основі її осі розщеплення прямого предка. Наприклад, якщо батьківський вузол був розділений уздовж осі x, поточний вузол розбивається уздовж осі y. Вибирається площина розщеплення, яка призводить до ітеративного вибору найбільшого однорідного підвибору. Якщо суб’єкт не може бути розділений на два підвибора, так що, хоча б, один з цих підборів є однорідним, об’єм розділяється уздовж наступної послідовної осі. Якщо жодна з двох інших осей не допускає однорідного

підвибору, обсяг розділяється по центру. Цей процес триває до тих пір, поки умови завершення не будуть виконані.

Вертикальна вісь на рис.2.14 вказує $\log_{10}$ часу, необхідного для побудови воксельного представлення висоти з використанням трьох структур даних. 2048×2048×2048 Sparse voxel octree було створене за 12 секунд, в той час як Point-region octree і kd-tree використовували 124,9 і 636,6 секунди, відповідно.

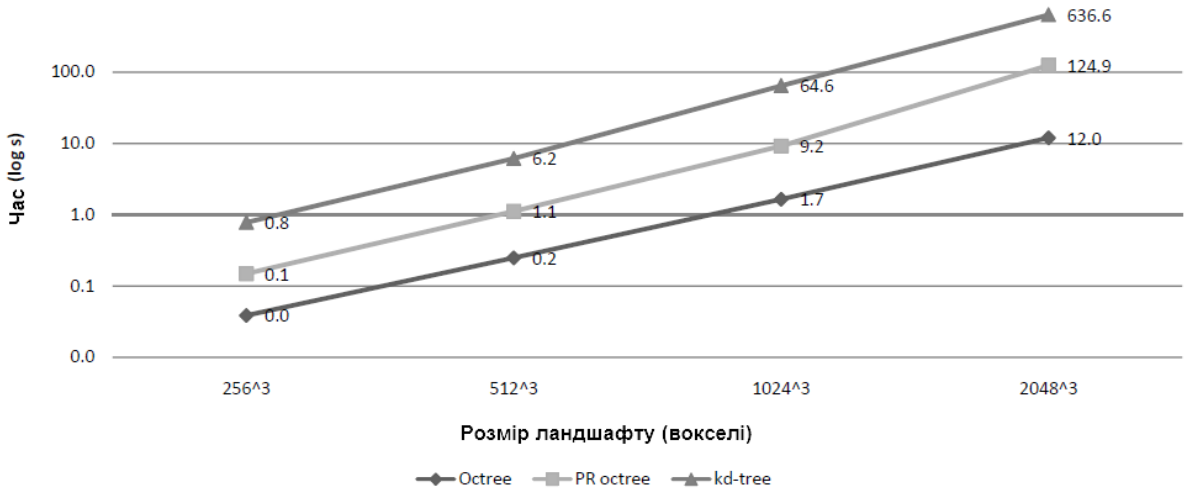


Рисунок 2.14 – Час, що використовується для генерації наборів даних вокселів з полів висоти

2.4 Висновки

Було досліджено придатність трьох структур воксельних даних для подання та моделювання алгоритмів еволюції ландшафту.

Було виявлено, що kd-tree і Point-region octree є кращими в мінімізації числа фактичних вокселів, які повинні зберігатися через їх здатності тісно обмежувати об’єм навколо неоднорідних даних. Kd-tree забезпечувало зберігання найбільшого об’єму пам'яті трьох структур даних (рис.2.11).

Показники Point-region octree і Sparse voxel octree мали аналогічні експлуатаційні характеристики, коли для деформування набору даних використовувалася різьблення по вокселях (рис.2.13) і при вибірці вокселів (рис.2.12). Kd-tree виконувалося повільніше, ніж Sparse voxel octree і Point-region octree при різних розмірах і типах ландшафту.

На рис.2.14 показано, що тільки Sparse voxel octree здатний створювати воксельне представлення висотної області з допустимою швидкістю для великих наборів даних. Час ініціалізації kd- tree і Point-region octree занадто повільний, щоб забезпечити швидку ітерацію.

Таким чином, використання Sparse voxel octree є найкращим варіантом для вирішення задачі генерації ландшафту в ігрових програмах. Застосування Sparse voxel octree є оптимальним з точки зору зменшення вимог до простору (рис.2.11) в порівнянні з прямою воксельної матрицею, яка вимагає 8 ГБ для зберігання дискретного набору даних вокселів 2048×2048×2048.

Враховуючи переваги Sparse voxel octree, саме цей алгоритм вибираєтеся для подальших досліджень і оптимізації воксельної технології генерації ландшафту в ігрових програмах.

3. РІШЕННЯ ЗАДАЧІ ГЕНЕРАЦІЇ ЛАНДШАФТУ В ІГРОВИХ ПРОГРАМАХ. МЕТОДИ МОДИФІКАЦІЇ ЛАНДШАФТУ

3.1 Застосування алгоритму Marching Cubes. Використання методу Voxel carving для генерації ландшафту в ігрових програмах

Застосуємо попередньо описане воксельне октодерево в процесі генерації ландшафту в ігрових програмах. Необхідно звернути увагу, що обладнання для обробки графічних зображень оптимізоване для рендерингу поверхонь, а не для рендерингу об'ємних даних. Тому, для початку, перетворимо представлення об'ємних даних в представлення на основі сітки шляхом вилучення поверхонь набору даних.

Застосуємо метод вилучення таких поверхонь – алгоритм Marching Cubes [31]. Цей алгоритм ділить загальний обсяг набору даних на набір менших обсягів. Топологія поверхні для кожного з цих підвиборів вибирається на основі значень набору даних в кутах цього підвибору. Далі різні підвибори об'єднуються для формування результуючої сітки. Вихідний алгоритм Marching Cubes, а також практично всі похідні методи працюють з даними, отриманими шляхом оцінки функції щільності. Ця функція щільності використовується для оцінки різних точок перетину і нормалей всередині кожного з суб’єктів.

Алгоритм Marching Cubes можна використовувати для вилучення поверхневої сітки за один крок. Для поліпшення зовнішнього вигляду сітки використовуються два додаткових кроки. Перший з них полягає в тому, щоб згладити сітку і приховати набір даних вокселів з дискретним розміром, а другий – розрахувати нормалі вершин для згладженої сітки. Алгоритм екстракції поверхні працює на логічно незалежних підвибірках, тоді як згладжування сітки і нормальні розрахункові операції залежать від результатів попереднього кроку. Таким чином, ці три етапи реалізовані як три окремих проходи.

Фундаментальне розуміння алгоритму Marching Cubes полягає в тому, що топологія поверхні, що проходить через кожен підвибір, може бути

однозначно визначена шляхом вивчення восьми кутів куба. Ці значення кута визначають, чи знаходиться кут всередині або зовні поверхні. У разі набору даних дискретного рельєфу ці значення визначають тип матеріалу з переходом між різними типами матеріалів в базовому наборі даних, який визначає ізоповерхні.

Єдиними нормально видимими переходами матеріалу є переходи між поверхнями і повітрям, тому кожен кут може бути представлений як двійковий вибір між повітрям (зовні) і матеріалом (всередині). Таким чином, є 2^8 можливих контурів поверхні на кожен обсяг. Конфігурація поверхні не змінюється, якщо значення на її кутах інвертовані, тобто кути, які були зовні, змінені на внутрішні кути і навпаки. Крім того, обернено-симетричні зміни мають одну топологію. Вихідний алгоритм Marching Cubes зменшує вихідні дані з 2^8 до 15, використовуючи ці властивості. Наступним кроком ці 15 випадків використовуються для побудови таблиці, яка описує всі 256 можливих сценаріїв [29]. Значення індексу в цю таблицю формується шляхом об'єднання значень восьми кутів куба у вигляді вибірок з набору даних. Використовуються дві окремі таблиці пошуку, пов'язані з цим індексом. Таблиця E вказує краї, які перетинаються поверхнею для заданого індексу.

Спеціальне значення індексу може призвести до створення до 5 трикутних граней, при цьому кожен трикутник визначається як потрійний, що складається з індексів трьох ребер, які він перетинає, з ребрами, пронумерованими від 0 до 11, як на рис.3.1.

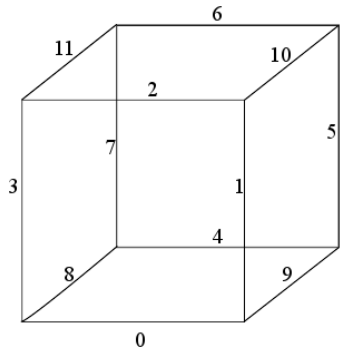


Рисунок 3.1 – номери кутів куба в алгоритмі Marching Cubes

Опис і застосування алгоритму Marching Cubes в генерації ландшафту в іграх не є темою даної магістерської дисертації. Враховуючи також те, що на даний момент його застосування при генерації ландшафтів вже є відомим і описаним в багатьох літературних джерелах, детально описувати даний алгоритм потреби немає.

Однак, розглянемо візуалізацію дискретних воксельних даних. Набір даних в значній мірі залежить від стиснення однорідності для зменшення вимог до пам'яті. Таким чином, обмежений діапазон елементів може бути представлений набором даних. Обмежений набір даних їх джерела передбачає певну однорідність відображуваного зображення, що ускладнює сприйняття глибини.

Текстурування використовується для сприйняття глибини. Нормальне двомірне текстурування поверхні є проблематичним, так як вся сітка повинна бути оновлена кожен раз при зміні будь-якого підвибора. Крім того, це повторне копіювання повинно відбуватися таким чином, щоб візуально не дратувати користувача.

В якості альтернативного підходу для кожного візуалізованого фрагмента застосовується карта рельєфу (Bump mapping) [32], яка являє собою карту, яка описує збурення нормалей. Коли примітив відображається, відповідне відхилення на цій карті вибирається для кожного візуалізованого фрагмента, аналогічно текстуруванню. Це збурення потім додається до інтерпольованої нормалі, розрахованої для цього фрагмента, щоб звести до нової нормалі, яка використовується при розрахунках освітлення для цього фрагмента. Таким чином, відображення вибоїн дозволяє простій геометрії виглядати так, як ніби вона має вибоїсту поверхню.

Тривимірна об'ємна текстура використовується як карта рельєфу. Кожен тексель [33] в цій текстурі задає вектор (x; y; z) шуму, збережений у вигляді колірного триплета (r; g; b). Ця текстура викладена по всьому простору об'єкта. Для запобігання візуально різких переходів між плитами використовується тривимірна безшовна версія шуму Перліна.

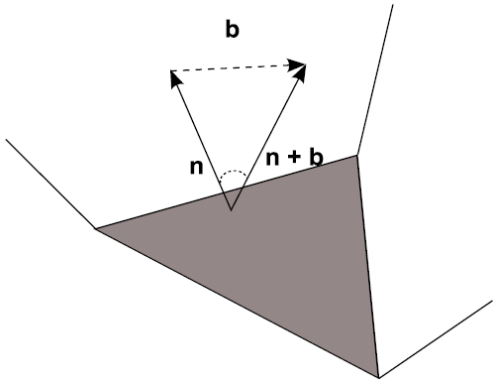


Рисунок 3.2 – Відображення рельєфу

Єдині координати текстури, передані фрагментарним шейдерам, потрапляють в діапазон [0; 1]. Розташування кожної вершини в координатах об'єктного простору ділиться на довжину сторін об'ємної текстури в шейдері фрагмента, так що кожен тексель в об'ємній текстурі відповідає одному вокселю в структурі даних. Цей параметр координати потім передається в шейдер фрагмента без подальшого перетворення.

Фрагмент-шейдер [34] використовує ці координати для вилучення нормального збурення з карти об'ємної текстури. Потім ця нормаль b (рис. 3.2) додається до нормалі n, інтерпольованої з вершин примітиву. Ця збурена нормаль n + b потім використовується в якості поверхневої нормалі в моделі затінення Blinn-Phong [35] для обчислення кольору фрагмента. У міру деформування ландшафту, сітка переміщається по об'ємної карті рельєфу.

Витяг сітки ефективно заснований на виборі між внутрішнім або зовнішнім. Тому різні типи матеріалів не представлені явно під час процесу екстракції. Швидше, ці відмінності реалізуються тільки під час рендерингу, де різним типам матеріалів присвоюються різні візуальні властивості.

Так як кожен кут кожного куба є двійковим значенням, то властивості типу матеріалу лінійно інтерполюються по поверхнях, на відміну від основного складу матеріалу. Інтерпольований колір використовується як коефіцієнт використання, так і коефіцієнти дзеркального відображення по моделі затінення. На рис.3.3 показано приклад використання цього відображення рельєфу. Лівий рендер був згенерований без порушення

нормалей з картою рельєфу. Правий рендер був згенерований з використанням відображення рельєфу. Стандартне відображення текстур не використовувалося; деталь кращого рендерингу повністю створюється моделлю освітлення зі збурених нормалей.

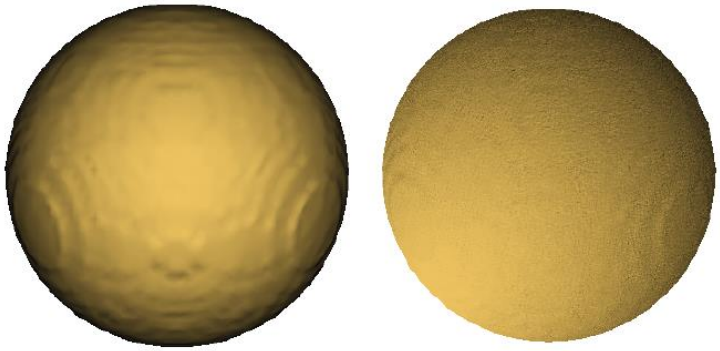


Рисунок 3.3 – Ефект відображення рельєфу (Bump mapping)

Для швидкого рендерингу потрібне використання вершинних масивів, на відміну від передачі окремих вершин апаратного забезпечення. Для повного виділення з вершинних масивів вершин в одному кубі недостатньо, тому масиви повинні бути сформовані з більших суб'єктів, що охоплюють кілька кубів. Це досягається шляхом вставки вершинних буферів в октети маршових кубів на обраній висоті (рис.3.4).

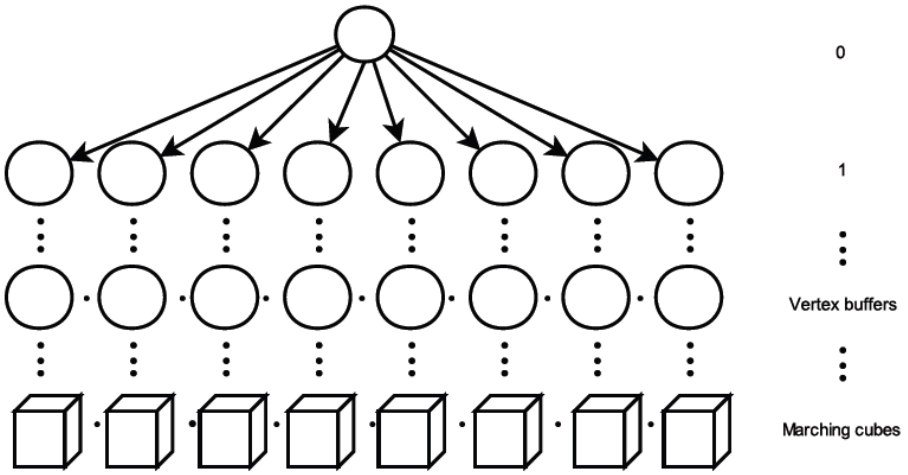


Рисунок 3.4 – Рівні дерева в алгоритмі Marching Cubes

Візуалізація дерева маршових кубів виконується шляхом рекурсивного спуску з кореневого вузла до тих пір, поки не буде досягнутий рівень вузлів з вершинними буферами. На кожному етапі рекурсії всі дочірні вузли утримуються всередині обмежуючого рівня поточного вузла.

Якщо обмежуючий об'єм не проектується на вікно перегляду, жоден з його дочірніх вузлів не може бути видимим, що дозволяє використовувати просторову організацію дерева маршових кубів безпосередньо для перегляду усічення. Це особливо підходить для ідентифікації об'єкта під курсором миші (тобто вибору), оскільки для перегляду доступна тільки піксельна область 1 × 1.

У міру збільшення висоти, на якій розташований вершинний буфер, збільшується кількість маршових кубів, представлених одним вершинним буфером, з очікуваним збільшенням продуктивності рендерингу. Це також означає, що весь масив вершин повинен бути перебудований щоразу, коли змінюється один з маршових кубів в суб’єкті. Таким чином, між швидкістю рендерингу і оновлюванням має бути досягнута рівновага. Ця оптимальна точка балансу повинна вибиратися окремо в кожному конкретному випадку в залежності від візуалізації структури даних, бажаності інтерактивності і доступних апаратних ресурсів. Хоча вибір цієї точки балансування не має відношення до цієї магістерської дисертації, система здатна до інтерактивного рендерингу .

Розмір кожного похідного куба можна збільшити, щоб зменшити загальну кількість створених кубів. Це призведе до збільшення продуктивності рендерингу. Збільшення розміру також зменшує розмірність представлення сітки, тому необхідно досягти компромісу між продуктивністю рендерингу і якістю зображення.

Перейдемо безпосередньо до процесу генерації, а також подальшої модифікації ландшафту в ігрових програмах. Попередньо було визначено, що найкращим варіантом серед досліджених структур є використання Sparse voxel octree. Варто зазначити, що інтерактивна деформація великих наборів даних може бути досить затратною для обрахування через велику кількість використовуваних вокселів. Особливі вимоги до ресурсів ставить ітеративне моделювання фізичних процесів. Застосуємо структуру Sparse voxel octree з використанням методики Voxel carving або воксельного

різьблення – інтерактивної методики скульптурингу, яка дозволяє користувачу вибірково виконувати операції з вокселями на підрозділах наборів даних [36]. Користувачеві надається інструмент, який деформує ландшафт, виконуючи різьблення по вокселю і ефективно встановлює вокселі відповідно обраному типу місцевості, що дозволяє користувачеві вручну обрізати необхідні ділянки.

Необхідно також врахувати, що зміна базової структури вокселя може привести до того, що витягнуті маршові куби перестануть бути репрезентативними для місцевості. Виділені куби утворені виключно з вокселів, що лежать на краю одного з маршових кубів. Кожен раз, коли змінюється один з цих граничних вокселів, маршеві куби, які представляють цей куб, позначаються для оновлення. Кожен маршевий куб, у свою чергу, відображається через вершинний буфер, який містить цілий ряд маршевих кубів. Виконання повного пошуку дерева кубиків, щоб знайти всі маршеві куби, може бути неефективним. Тому, коли куб позначено для оновлення, кожен попередник вузла, який містить недійсний маршевий куб, до вузла, що містить вершину, включаючи позначку, позначається для оновлення. Це дає змогу безпосередньо оновлювати вибрані маршеві куби без пошуку по всьому дереву, коли воно потребує оновлення.

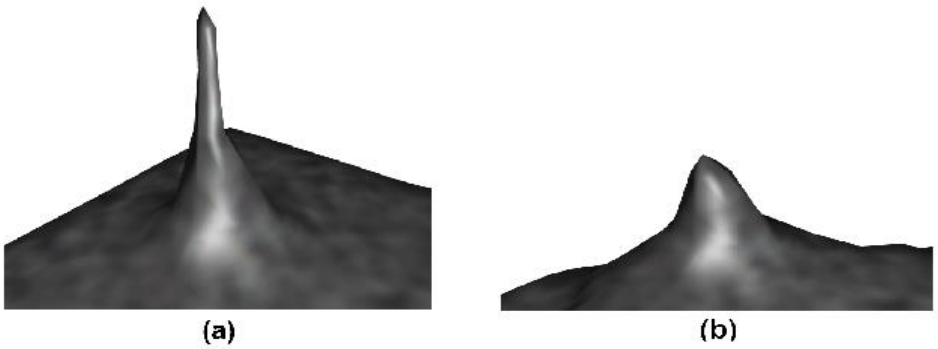


Рисунок 3.5 – Невеликі зміни на місцевості з різьбленням по вокселю

Таким чином, застосування цього інструменту разом з використанням Sparse voxel octre дозволяє користувачеві безпосередньо змінювати ландшафт. Наприклад, воксельне різьблення може використовуватися для інтерактивного створення невеликих змін ландшафту (рис.3.5, де (а) –

оригінал ландшафту, (b) – місцевість після воксельного різьблення, яка була використана для заміни об'ємів поверхневих вокселів повітряними вокселями) або для створення тунелів і печер (рис.3.6, де (a) – це вид на поверхню з входом тунелю, (b) – повністю тривимірна тунельна мережа).

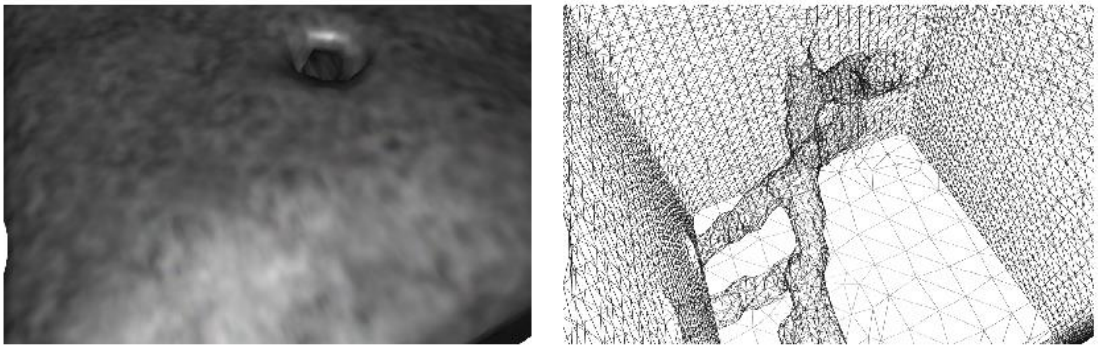


Рисунок 3.6 – Тунельна мережа

Для більш детального розгляду застосування Sparse voxel octree розглянемо явище термічної ерозії. Термічна ерозія обумовлена розширенням і стисненням рельєфу, що призводить до розриву фрагментів ландшафту [33]. Ці фрагменти потім переміщуються вниз до нижньої сусідньої місцевості.

Кожна інерція ерозії повинна визначати, який з поверхневих вокселів повинен бути зруйнований. Пошук поверхневих вокселів можна прискорити, використовуючи існуючий список маршевих кубів, який генерується вже описаним алгоритмом Marching cubes. Для кожного з цих вокселів повинна бути знайдена найнижча сусідня точка місцевості. Хоча зчитування локалізованих даних про місцевість може бути відносно швидким, повторення цієї операції для кожного вокселя, незважаючи на використання Sparse voxel octree, робить термічну ерозію затратною для обчислень. Крім того, перед досягненням стабільності може знадобитися велика кількість ітерацій.

Високі обчислювальні витрати на алгоритм, такий, як термічна ерозія, є вагомим аргументом, що виключає використання моделювання фізичного процесу в інтерактивному середовищі проектування ландшафту. Більш

складні явища, такі як гідравлічна ерозія, можуть бути ще більш затратними [34]. Ці результати впливають на вибір алгоритмів, які створюють особливості місцевості на відміну від імітації фізичних процесів, тому саме для таких випадків використання Sparse voxel octree є потрібним альтернативним рішенням.

У той час як глобальні деформації на вокселі не є практичними, можна розглянути локалізовані оператори деформації рельєфу.

Реалізація локалізованого оператору термічної ерозії, який істотно обмежує глобальну термічну ерозію на менший об'ємний обсяг. Однак принципово ітеративний характер ерозії заважає ефективному використанню локалізованих операторів ерозії. Хоча кожна ітерація може бути інтерактивною, для значної видимої деформації потрібна велика кількість ітерацій. Зрозуміло, що інтерактивні дослідження деформації вокселів були зосереджені на неітеративних і сильно локалізованих деформаціях.

Незважаючи на те, що, звичайно, є можливість для використання не-фізичних операторів деформації рельєфу, проте в центрі уваги деформацій, є більш масштабні деформації рельєфу місцевості. Наприклад, поворот може бути використаний для інтерактивного створення невеликих змін ландшафту (рис.3.5) або для створення тунелів і печер (рис.3.6).

Повернемося до інструменту Sparse voxel octree – воксельного різьблення, яке дозволяє користувачеві проводити спеціальні зміни, хоча й не надає можливість легко створювати великомасштабні особливості місцевості, такі як гори і долини. Прийнятий спосіб їх створення у тому, щоб інтерпретувати оброблені процедурою дані як фрактали. Однак попередні, підходи, що засновані на фракталах, підходи мають деякі недоліки:

- Кількість контрольованих користувачем ресурсів обмежена. Вхідними даними для цих фрактальних функцій є початкові значення, які зазвичай керуються тільки глобальним видом ландшафту, але не дозволяють створювати специфічні функції;

- Початкові значення мають обмежений прямий ефект на місцевості і тому розробникам складно наперед визначити яким буде ландшафт.

Інтерфейси малювання дозволяють користувачеві попередньо передбачити загальну форму рельєфу, з якого далі вже процедурно деталізуються. Хоча цей метод демонструє перспективу, був застосований підхід, в якому використовується відомий алгоритм деформації вільної форми в поєднанні з даними, що процедурно генеруються. Цей алгоритм дозволяє користувачеві в цілому визначати характеристики рельєфу місцевості, при цьому автоматично створюючи об'єкти меншого розміру.

3.2 Застосування дротової деформації при генерації воксельного ландшафту в ігрових програмах

Розглянемо варіант використання Sparse voxel octree в поєднанні з Wire deformation – дротовою деформацією, яка аналогічна дротам, що використовуються для створення скульптур в реальному світі [35]. Ці дроти можуть використовуватися для визначення форми або перебільшення конкретних особливостей. Дизайнеру потрібно тільки взаємодіяти з провідником, причому потенційно більш складна базова модель деформується через вплив дротів.

Подібні деформації мають дві стадії: перша полягає в тому, що дизайнер розміщує дроти на моделі. Кожна точка моделі пов'язана з набором точок на одному або декількох дротах; далі друга стадія, яка полягає в том, що дріт деформується дизайнером. Модель деформується разом з дротами.

Алгоритм Wire deformation дозволяє користувачеві інтуїтивно і зрозуміло керувати деформаціями, що робить його придатним для інтерактивного великомасштабного дизайну властивостей місцевості.

Сітка, яку ми отримуємо за допомогою алгоритму Marching cubes, може бути використана, але існують візуальні недоліки, викликаними провідниками, що перетинають діагональну частину кубиків. Розглянемо

випадки, коли одна геометрична половина куба має повітря, а друга половина – рельєф (рис.3.7 (a), де зображена типова сітчаста конфігурація). Якщо дріт тепер наближається до протилежної діагоналі (рис.3.7 (b), де відображена проблема крос-зернистого дроту і як наслідок деформація, спричинена дротом, наближається до форми f уздовж дроту, що призводить до нерівних деформацій).

Використання дротової деформації місцевості в першу чергу призначене для створення відносно грубих об'єктів, таких як гори і річки. Використання Sparse voxel octree разом з дротовою деформацією застосовується до безлічі точок. Хоча можна також деформувати кожен воксель в наборі точок, але затрати для обчислення були б досить високими і це не дозволило би використовувати інтерактивний дизайн ландшафту. Тому деформація застосовується до вершин багатокутної сітки.

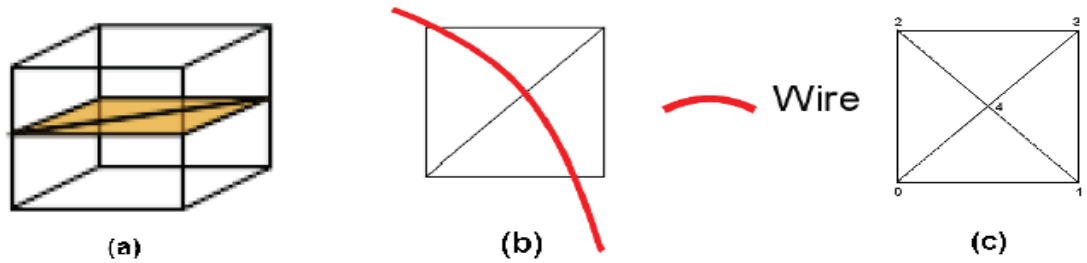


Рисунок 3.7 – Структури сітки, які генеруються алгоритмом маршевих кубів, не підходять для деформації вільної форми

Отже, деформації дроту застосовується до вершин висоти, які наносяться на існуючі дані вокселя, так що висота відповідає найвищим точкам його даних. Кожна висота квадрата вирівнюється в чотири трикутники, розміщуючи додаткову вершину в центрі (рис. 3.7 (c)).

Застосуємо алгоритм Wire deformation для деформації ландшафту, побудованого на базі Sparse voxel octree. Створимо еталонну криву R шляхом розміщення контрольних точок для кривої Безьє. Далі деформуємо криву R в W.

Функцію f виберемо так, що f (0) = 1, f (x ≥ 1) = 0 і f ′(0) = f ′(1) = 0. Для цього f (x) = $(x^2 - 1)^2$. Цей вибір призведе до відносно поступової функції, яка монотонно зменшується (рис.3.8).

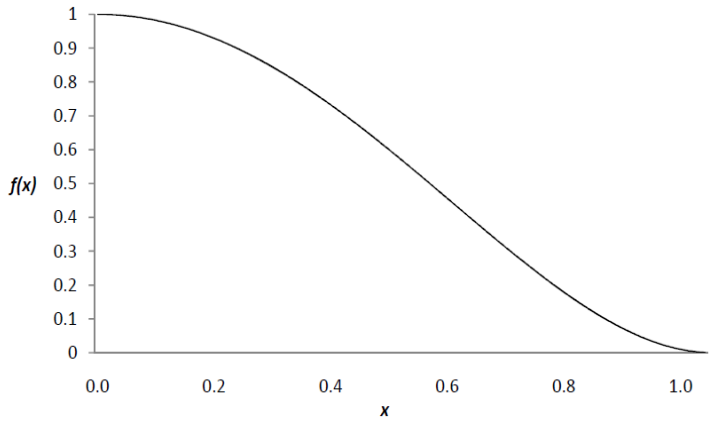


Рисунок 3.8 – Функція f (x) = $(x^2 - 1)^2$

Для будь-якої вершини P на сітці, нехай C (P) – найближча точка до P на кривій C. Крім того, нехай

$$F(P, C) = f(\frac{\|P - C(P)\|}{r})$$

Функція F дорівнює нуля для всіх ||P – C(P)|| ≥ |r| і дає вплив дроту на P для всіх ||P – C(P)|| < |r| тому F(P;C) ∈ [-1; 1].

Вектор деформації ΔP зображений на рис.3.9 для певної точки P розрахуємо як ΔP = (W (P) – R(P)) · F(P,R):

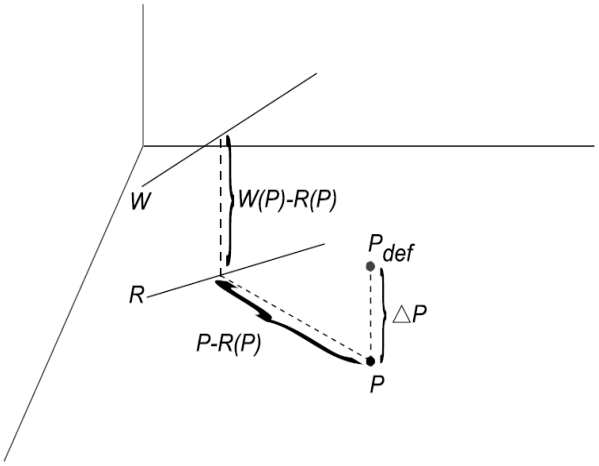


Рисунок 3.9 – Дротова деформація. Опорна крива R деформована користувачем до W, внаслідок чого відбувається деформація P до P_{def}.

Деформовану точку визначимо як P_{def} = P + ΔP. Параметр r визначає область навколо R, яка буде модифікована дротом. Вибір великого значення

r призведе до значного впливу на область. Від’ємне r може протидіяти близьким додатнім значенням F (P, C).

Виконаємо неявне дротове розбиття. Кожну оригінальну точку P на поверхні сітки прикріпимо тільки до однієї точки R(P) на еталонній кривій. На рис.3.10 (a) показана крива, лівий кінець якої деформується вниз, а її правий кінець деформується вгору. Точки в центрі кривої будуть перемикатися від впливу лівої контрольної точки через те, що на неї впливає права контрольна точка, яка знаходиться між двома сусідніми точками сітки.

На дві сусідні вершини на рис.3.10 (a) має вплив R(P), що диференціюється. Якщо одна половина деформована, а інша половина збережена як є, недеформована половина буде ефективно закріплювати другу половину сітки. Це може викликати неприродньо різкий рельєф, де відбувається перехід між контрольними точками (рис.3.11 (a)).

Можемо усунути ці різкі переходи, якщо на кожну точку впливатиме більш ніж одна точка на R. В ідеалі, рельєфні вершини, які стикаються з R, повинні бути деформовані так, щоб вони були падаючими з W [36]. Тому поділимо дріт неявно на піддроти, а комбіновані деформації цих піддротів використаємо для розрахунку кінцевої деформації. Кожен з цих n піддротів призведе до індивідуальної деформації в точці P_i.

$$\Delta P_i = (W_i(P) - R_i(P)) \cdot F(P, R_i)^m$$

Мірою впливу кожної піддеформації є $F(P, Ri)^m$, де m – додатнє число, яке використовується для експоненціального масштабування відстані ефекту, на кожному вкладі субдеформації.

Хоча кожна P_{def} може опинитися під впливом кожної точки на R, більш ефективно зв’язувати можна лише кожну P з вибраною кількістю точок на R. Отже, кожна P пов'язана з кожним з n місцевих мінімумів і максимумів ||R – P||, включаючи початкові та кінцеві точки R (рис.3.10 (b)).

Нехай R′ – підмножина дротів, обраних з розбитої R, так що локальні мінімуми або максимуми ||P – R|| знаходяться на $R_i' \in R'$. Кожен з цих

мінімумів і максимумів знаходиться в положенні $R_i'(P)$. Деформований дріт W відповідно підрозділяється так, що кожен $R_i'(P)$ має асоційований $W_i'(P)$.

Обчислимо деформовану позицію вершин [39]:

$$P_{def} = P + \frac{\sum_{R_i' \in R'} (W_i'(P) - R_i'(P) \cdot F(P, R_i'))^m}{\sum_{R_i' \in R'} F(P, R_i')^m)}$$

Продемонструємо ефект неявного поділу дроту, який впливає на деформацію місцевості (рис.3.11). Аналогічно на рис.3.10 (a), контрольна крива W була деформована так, що ліва половина трохи нижче, а права половина була піднята. На рис.3.11 (a) покажемо різкий перехід, оскільки вершини рельєфу перемикаються через те, що їх права половина W знаходиться виключно під впливом лівої половини W. На рис.3.11 (b) покажемо змішаний вплив, обумовлений неявним поділом дротів.

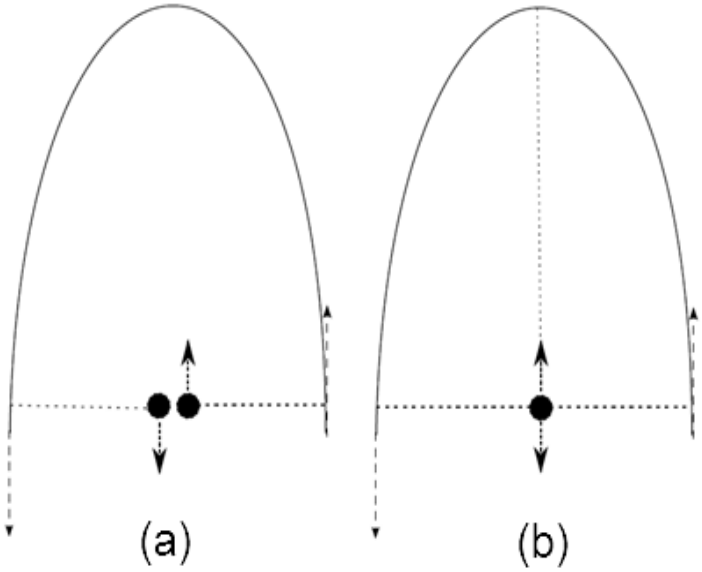


Рисунок 3.10 – Вплив контрольних точок на точки вершин сітки

Розглянемо область ефекту впливу. Параметр r управляє радіусом залежності кожного сегмента дроту. Користувач зможе змінити значення цього параметра в кожній контрольній точці кривої. Величина r уздовж кожного сегмента кривої лінійно інтерполюється за значеннями r в сусідніх контрольних точках.

Покажемо приклади деформацій на рис.3.12, які можуть мати додатні і від’ємні значення r. Змінюючи додатні значення r, можна створювати

більш тонкі чи широкі підвиди деформацій (рис.3.12 (a)). Найдальша контрольна точка на рис.3.12 (b) має велике від’ємне значення r, але сама контрольна крапка не переміщена. Велике від'ємне значення r домінує над меншими додатними значеннями r для вершин сітки, найближчих до найдальшої контрольної точки, в результаті чого маємо нетто-деформацію, яка чисельно близька до нуля.

Розділимо внутрішньо Криві Безьє на сегментні лінії. Ці сегменти використовуються для візуалізації кривої, а також для різних R(P): лінія між найближчою точкою L(P) на нескінченній лінії L до точки P завжди буде знаходитися під прямим кутом до самої L. Найближча точка L(P) на кінцевому сегменті L може бути однією з кінцевих точок відрізка лінії і в цьому випадку кут, ймовірно, не буде прямим кутом. Кожна з цих кінцевих точок є найближчою точкою на сегменті лінії для будь-яких точок в певній області.

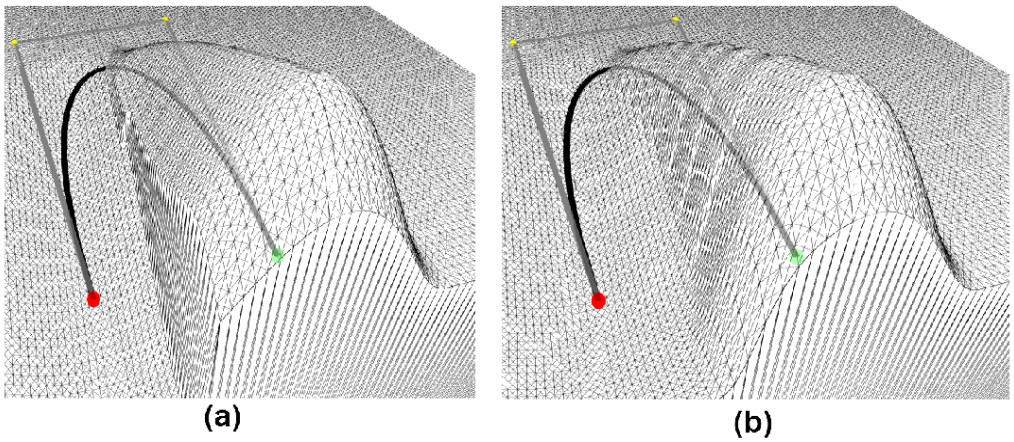


Рисунок 3.11 – Неявні результати підрозподілу

Кожна точка, в якій поєднуються два сегмента лінії, також є найближчою точкою на безлічі сегментів лінії для певної області (заштрихована область на рис.3.13). Таким чином, вершини сітки всередині таких областей будуть мати у своєму розпорядженні найближчу точку в поданні лінійного сегмента кривої.

Деякі вершини, що розділяють таку найближчу точку, також можуть знаходитися на одній і тій самій відстані від цієї найближчої точки. Так як

рівняння $F(P,C) = f(\frac{\|P-C(P)\|}{r})$ буде використовувати еквівалентні вхідні параметри, вихід також буде ідентичним. Так як перемички вершин будуть трансформуватися аналогічно, відбувається переплітання. Ця смуга показана на рис.3.12. Узагальненого аналітичного методу для знаходження найближчої точки на кривій Безьє не існує. Ітераційні алгоритми швидше використовуються для наближення цього значення [38].

Обв'язку можна зменшити за рахунок збільшення кількості згенерованих сегментів лінії, що зменшить площу збігів вершин сітки, пов'язаних з однією точкою. Проте, збільшення кількості сегментів лінії збільшить час, необхідний для всіх найближчих точок, що знизить загальну чутливість системи під час налаштування дроту. Таким чином, кількість сегментів лінії збільшиться тільки для роботи в вокселі. При додаванні шуму додається смуга.

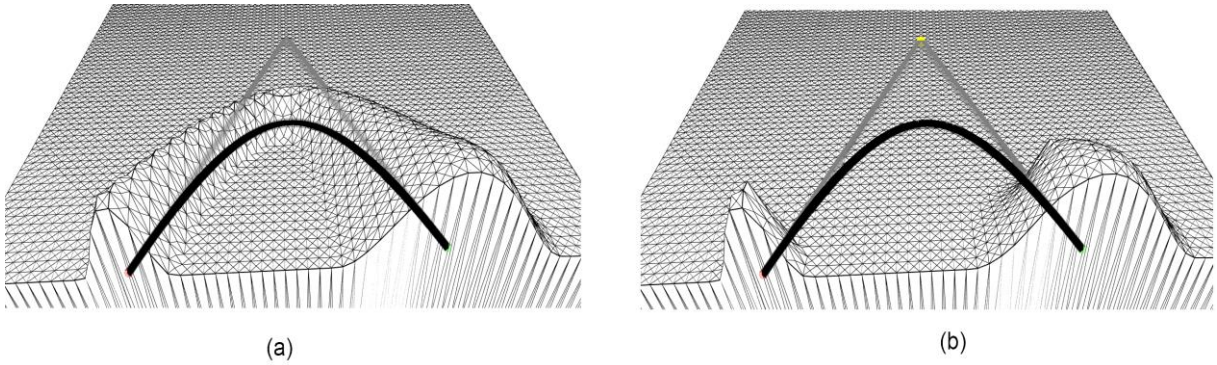


Рисунок 3.12 – Вплив параметра r

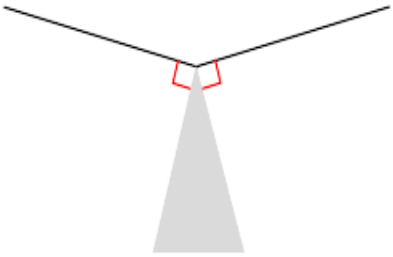


Рисунок 3.13 – Найближчі точки до сегментів лінії. Вершини всередині заштрихованої області мають спільну найближчу точку на сегментах лінії

Розглянемо використання шуму при воксельній генерації ландшафту в ігрових програмах більш детально. Деформації, викликані дротами, за

своєю природою мають ту ж форму, що і деформація f. Це призводить до того, що деформована поверхня буде неприродньо гладкою. Подібно звичайним підходам до проектування ландшафту, шум може використовуватися для додавання деталей. Використаємо шум спеціально для створення невеликих збурень місцевості, а не для створення більшого масштабу.

Шум Перліна використовуються для створення деталей. Користувачеві надається інтерфейс для зміни n, а також, який використовується для масштабування збурення. Збурена точка потім обчислюється як $P_p = P_d + \gamma \cdot F(P, C) \cdot Noise(P)$: Оскільки F(P, C) знаходиться під контролем r, забезпечується контроль над областю, в якій застосовується шум. Трохи деформуючи P, маючи великий r, можна створити традиційний фрактальний ландшафт (рис.3.14 (b)). І навпаки, менше значення r зробить шум бути більш локалізованим (рис.3.14 (a)).

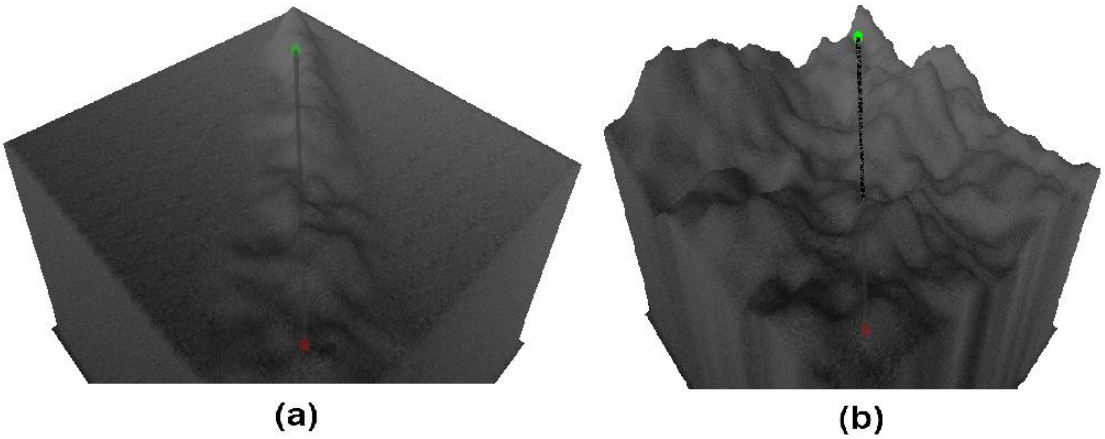


Рисунок 3.14 – Вплив зміни параметра r на рівень шуму

Запишемо алгоритм деформації:

Мета:	деформувати місцевість з віртуальним дротом.
Дано:	⟨W, R, s, r, f ⟩ A дріт
P	A список рельєфних вершин
Повертаємо:	деформована версія P
Структури даних:	p_α список найближчих дротів

$\forall\ p_i \in P$

$R' \leftarrow$ всі значення на R так $||R| - p||$ є локальним мінімумом чи максимум деформації $\leftarrow 0$

$\forall R'_i \in R'$

$\Delta p_i \leftarrow (W_i'(p) - R'_i(p)) \cdot F(P, R'_i)$

$$\text{деформація} \leftarrow \text{деформація} + \frac{\sum_{R'_i \in R'} \Delta P_i \cdot F(P, R'_i))^m}{\sum_{R'_i \in R'} F(P, R'_i)^m)}$$

$p_i \leftarrow p_i + \text{деформація} + \gamma \cdot \text{Noise}(p_d)$

Перейдемо безпосередньо до воксельної деформації. Деформоване висотне поле дозволить користувачеві інтерактивно спроектувати ландшафт з використанням дротів. Деформація, що застосовується до цих вершин, повинна бути застосована до структури даних вокселя до того, як будуть виконані інші узагальнені операції на місцевості.

Через те, що деформація дротів вертикально обмежена, деформація вокселів може бути представлена рядом деформацій воксельної колони. Замість того, щоб зіставляти кожен стовпець вокселя з інтерпольованою точкою між вершинами вершин висоти, деформація дротів застосується до верхнього вокселя колони. Результатом цієї деформації буде нова висота колони вокселів.

Відносні позиції і розміри рельєфних шарів будуть підтримуватися в міру того, як стовпець буде вертикально розтягнутий або стиснутий до нової висоти. В якості альтернативи, ландшафт можна просто замінити повітрям, якщо висота нового стовпчика менша існуючого, на відміну від перерозподілу вокселів в стовпці.

Застосування деформації до набору даних вокселів вимагає, щоб кожен оброблений стовпець був перебудований. Потенційно це може потребувати, щоб весь набір даних був перебудований і цей процес, безумовно, не є інтерактивним. Інтерактивний дизайн, проте, можливий, оскільки висота, накладена на набір даних вокселів, дозволить створювати швидкі проектні модифікації, даючи розумну апроксимацію поверхні, витягнутої з деформованої воксельної сітки.

Алгоритм Wire deformation можна використовувати для визначення інших широкомасштабних характеристик місцевості, таких як долини. На рис.3.15 показано, як створюється початкова долина, де (a) – зображення кривої з трьома контрольними точками. Спочатку поміщаємо криву в загальний вигляд бажаного ландшафтного об'єкта. Потім контрольні точки цієї кривої переміщаються вниз, що призводить до негайного зворотного зв'язку користувача з перекрученою висотою. Далі деформація застосовується до структури воксельних даних, (структура спотворюється дротом, а пов'язані структури відображення оновлюються (b)).

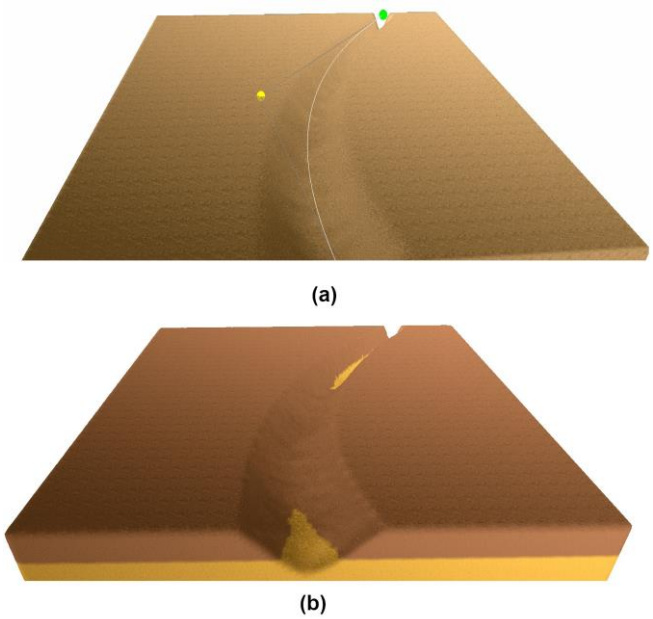


Рисунок 3.15 – Створення простої долини (алгоритм Wire deformation)

Тепер роздвоєння в існуючій долині можна створити, повторивши той же процес (рис.3.16). Створюємо криву, відповідну формі роздвоєної долини, а її контрольні точки аналогічним чином потрапляють вниз. Коли користувач задовольняє інтерактивно оновлюваний попередній перегляд, повна трансформація застосовується до структури вокселя. Воксельне різьблення може бути потім використане для регулювання місцевості навколо роздвоєння долини в міру необхідності.

На рис.3.17 (a) показано вихідний опорний дріт, що позначає загальну форму лінії гірського хребта. Потім цей дріт деформується, одночасно деформуючи навколишній ландшафт. Додаємо Шум Перліна, що призводить до явища, зображеного на рис.3.17 (b). Для створення цієї особливості місцевості, включно із застосуванням деформації вокселя, було потрібно близько десяти хвилин.

Потім опорні дроти поміщаються на недеформований ландшафт, що оточує лінію хребта (рис.3.18). Потім рельєф деформується з використанням шуму з меншою амплітудою, ніж при початковій деформації. Цей процес дублюється для областей перед і за гірською місцевістю. Для деформування навколишньої місцевості потрібно ще десять хвилин.

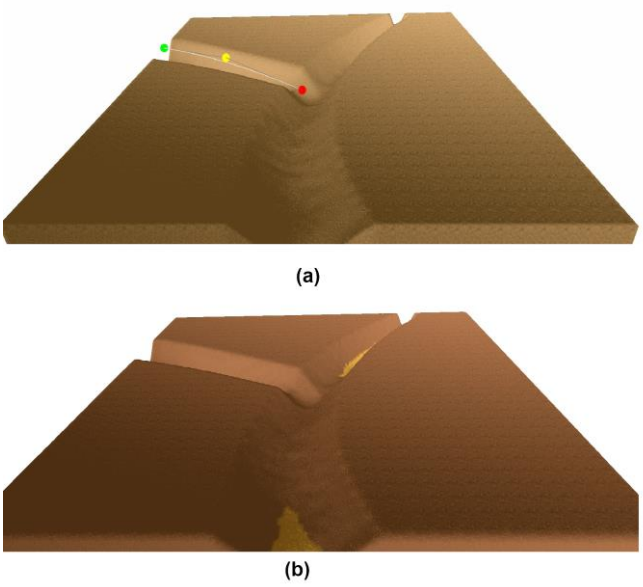


Рисунок 3.16 – Створення роздвоєною долини (алгоритм Wire deformation)

3.3 Представлення згенерованого ландшафту і деформації за допомогою алгоритму Wire deformation

Адаптація алгоритму Wire deformation, як частини структури Sparse voxel octree, до деформації рельєфу, представлена в цьому розділі, надає можливість проектування, в якій дротова сітка управляється інтерактивно. Після отримання задовільного попереднього перегляду повна деформація

застосовується до базової структурі даних вокселя. Маніпулювання великими кількостями вокселів не є інтерактивним.

Фундаментальний принцип дротового алгоритму полягає в тому, що сітки вершини пов'язуються з дротом, а потім деформуються в залежності від деформації дроту W. Кожна вершина сітки таким чином пов'язана з вершиною (або безліччю вершин) від еталонної кривої R. Якщо еталонна крива залишається нерухомою, цей зв'язок не потрібно перераховувати по мірі деформування W. Розрахункова вартість обчислення цієї асоціації тоді є функцією числа вершин сітки і числа вершин в W.

Крива Безьє не обов'язково проходить через контрольні точки. Щоб отримати корисну метрику для вимірювання збільшення обчислювального часу в міру збільшення кількості контрольних точок, R формуємо шляхом рівномірного розміщення контрольних точок по прямій (рис.3.19). Це гарантує, що число вершин в W збільшується приблизно лінійно зі збільшенням числа контрольних точок.

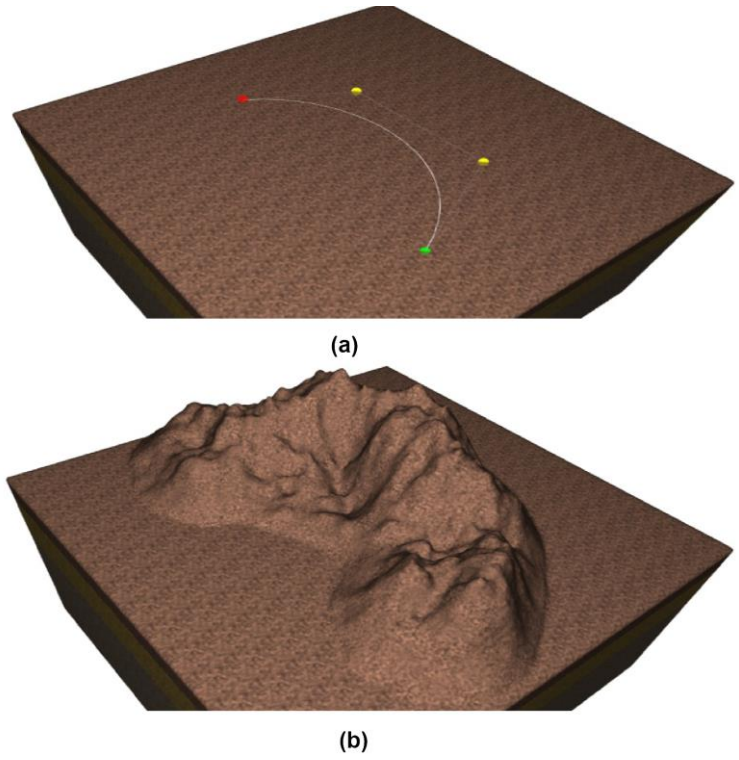


Рисунок 3.17 – Початкова дротова деформація для створення центральної гірської місцевості

У таблиці 3.1 показано ефект додавання додаткових контрольних точок за час, необхідний для перерахунку асоціацій між вершинами сітки і R. Збільшення кількості контрольних точок не матиме значущого вимірного впливу на час, необхідний для стандартного виконання дротового алгоритму. Однак час, що використовується розширеної реалізацією алгоритму Wire deformation, збільшується приблизно лінійно.

Таблиця 3.1 – Час, необхідний для зв’язування вершин сітки з R

Кількість контрольних точок	2	3	4	5
Стандартна реалізація (секунди)	2.27×10^{-3}	2.24×10^{-3}	2.28×10^{-3}	2.29×10^{-3}
З неявним підрозділом (секунди)	2.78×10^{-2}	3.06×10^{-2}	3.36×10^{-2}	3.74×10^{-2}

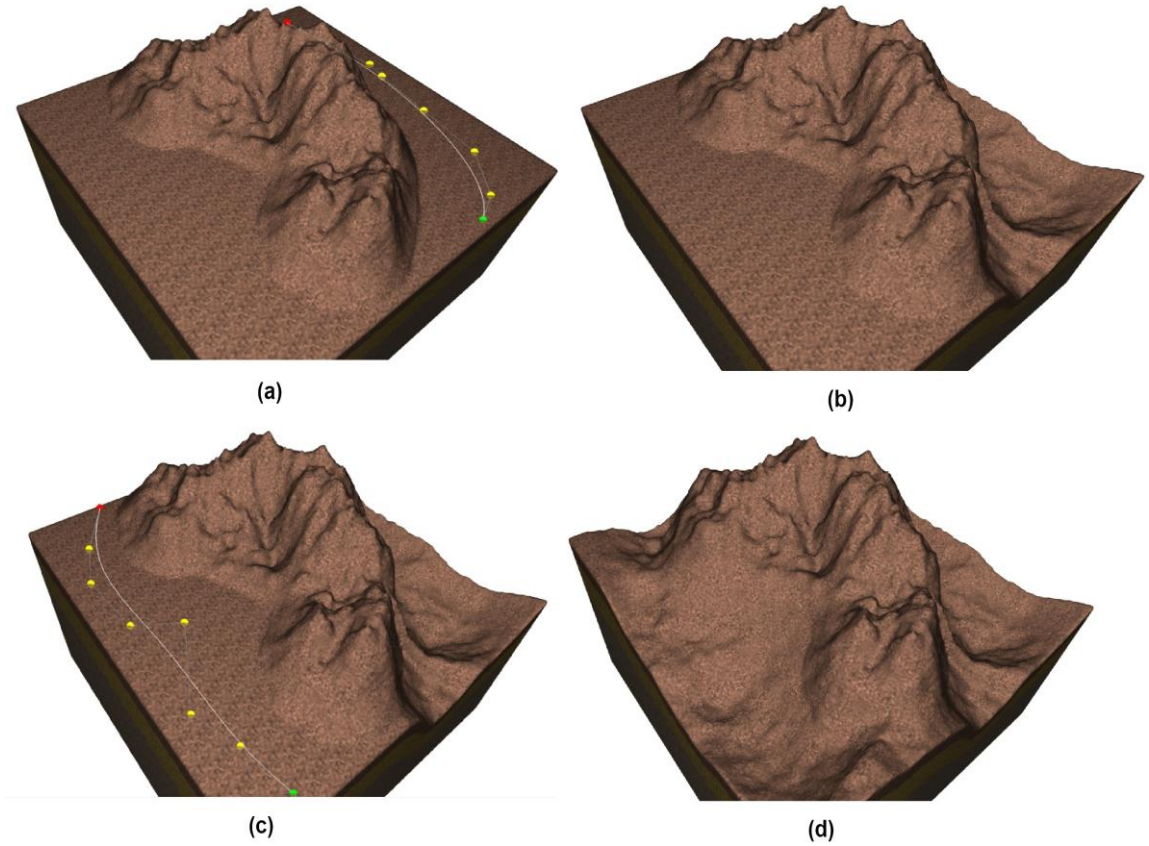


Рисунок 3.18 - Наступні еталонні дроти і деформації для процедурного створення навколишнього простору

Обидві ці реалізації достатньо швидкі, щоб не заважати інтерактивності, навіть якщо сітчасті вершини пов'язані з R щоразу, коли W

деформується. Таким чином, ці реалізації допускають інтерактивну деформацію сітки, яка використовується як попередній перегляд деформації вокселя.

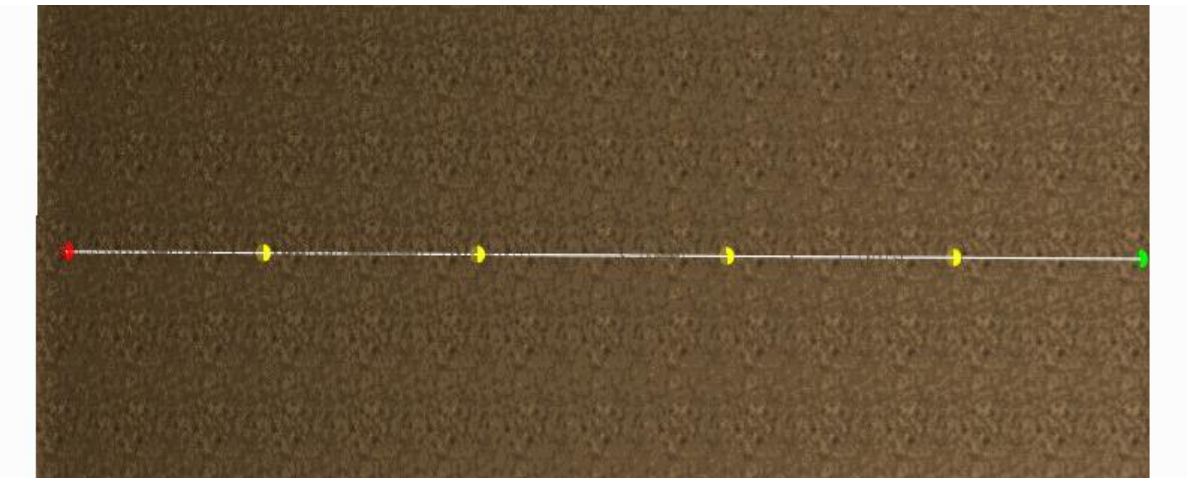


Рисунок 3.19- Конфігурація кривої для вимірювання ефекту додаткових контрольних точок

У таблиці 3.2 показано час, що використовується для застосування приблизних деформацій дроту до 512 × 512 × 512 в воксельних ландшафтах.

У всіх трьох випадках R визначається шляхом розміщення однієї контрольної точки в іншому куті і однієї контрольної точки в найближчому куті.

Таблиця 3.2 – Час, необхідний для деформації вокселів

<table>
<tr><td colspan="2">Відносно невелика деформація дротів, отримана шляхом розміщення дроту, що складається з двох контрольних точок на місцевості. Деформувалася тільки найближча контрольна точка.</td></tr>
<tr><td>  </td><td>Кількість порушених вокселів – 6626124
Секунди, використані для деформації – 27
Секунди, використані для поновлення маршових кубів – 20</td></tr>
<tr><td colspan="2">Велику деформацію можна отримати, злегка змінивши обидві контрольні точки.</td></tr>
</table>

Продовження таблиці 3.2

	Кількість порушених вокселів – 23665839 Секунди, використані для деформації – 40 Секунди, використані для поновлення маршових кубів – 65
Відносно велика деформація, коли більшість існуючих вокселів була видалена.	
	Кількість порушених вокселів – 58975031 Секунди, використані для деформації – 58 Секунди, використані для поновлення маршових кубів – 44

Верхня деформація була отримана шляхом переводу найближчої контрольної точки вниз і незначного збільшення її значення r. Середня деформація була отримана, внаслідок переводу обох контрольних точок вниз, а також збільшення їх значення r. Нижня деформація була отримана шляхом переводу обох точок вниз і збільшення їх значень r до тих пір, поки більша частина ландшафту не буде видалена.

По мірі того як загальний розмір деформації збільшується, більше вокселів змінюються в процесі деформації. У міру збільшення кількості вокселів збільшується час, необхідний для поновлення структури даних вокселів в Sparse voxel octree.

Продемонстровані великомасштабні деформації, роблять недійсними великі частини дерева маршових кубів. Оновлення структур даних рендерингу в великих масштабах може потребувати відносно багато ресурсів (таблиця 3.2), однак, для масштабних задач це оновлення є частиною загального часу, необхідного для повної деформації, яка принципово не є інтерактивною.

3.4 Висновки

Було описано і застосовано алгоритм Marching cubes, необхідний для представлення даних на основі сітки, шляхом вилучення з поверхонь набору даних. В процесі генерації ландшафту застосовано попередньо описане воксельне октодерево.

Запропоновано метод Voxel carving для генерації ландшафту, який є корисним для ручної модифікації невеликих об’ємів рельєфу. Було розглянуто варіант генерації ландшафту, заснованого на реальних фізичних процесах, який, за рахунок комбінації високих обчислювальних затрат і дискретного характеру вокселів, не є практичним/інтерактивним. Таке моделювання залишається перспективним напрямком для майбутніх досліджень по мірі збільшення загальної потужності обробки.

Було описано і запропоновано алгоритм Wire deformation, який дозволяє генерувати об'єкти великої площі. Замість зменшення сітки, використовуючи алгоритм Marching Cubes, було використано зменшене значення висоти, щоб зменшити гострі краї. Управління здійснювалось за площею, що проходить по кожному сегменту дроту, дозволяючи користувачеві змінювати параметр r, який також використовується для управління амплітудою шуму і дозволяє генерувати ландшафт (або частини місцевості) зі стандартними фрактальними ландшафтними характеристиками.

Неявне дротове розбиття використовується для подолання складності стандартного алгоритму Wire deformation. Усюди представлений інтерактивний попередній перегляд деформації. Інтерактивність допускається відносно швидким оновленням сітки висот.

Запропонований алгоритм Wire deformation в поєднанні з Sparse voxel octree може бути реалізовано в процесі генерації ландшафтів в ігрових програмах для модифікації згенерованого ландшафту в режимі реального часу. Ця реалізація представлена в наступному розділі магістерської дисертації.

4. ОПИС І РЕАЛІЗАЦІЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ

4.1 Структура розробленого програмного продукту

Для демонстрації процесу генерації ландшафту в ігрових програмах за допомогою вокселів з використанням Sparse voxel octree в поєднанні з Wire deformation було розроблено і реалізовано програмний продукт – воксельний ландшафт з можливістю динамічних модифікацій в режимі реального часу.

Розробка проводилася в середовищі Unity – багатоплатформового інструменту для розробки дво- та тривимірних додатків для ігор, що працює на операційних системах Windows і OS X.

Мова програмування, використовувана для розробки – C# (C Sharp).

Розглянемо діаграму класів, спроектовану при розробці програми, зображену на рис.4.1

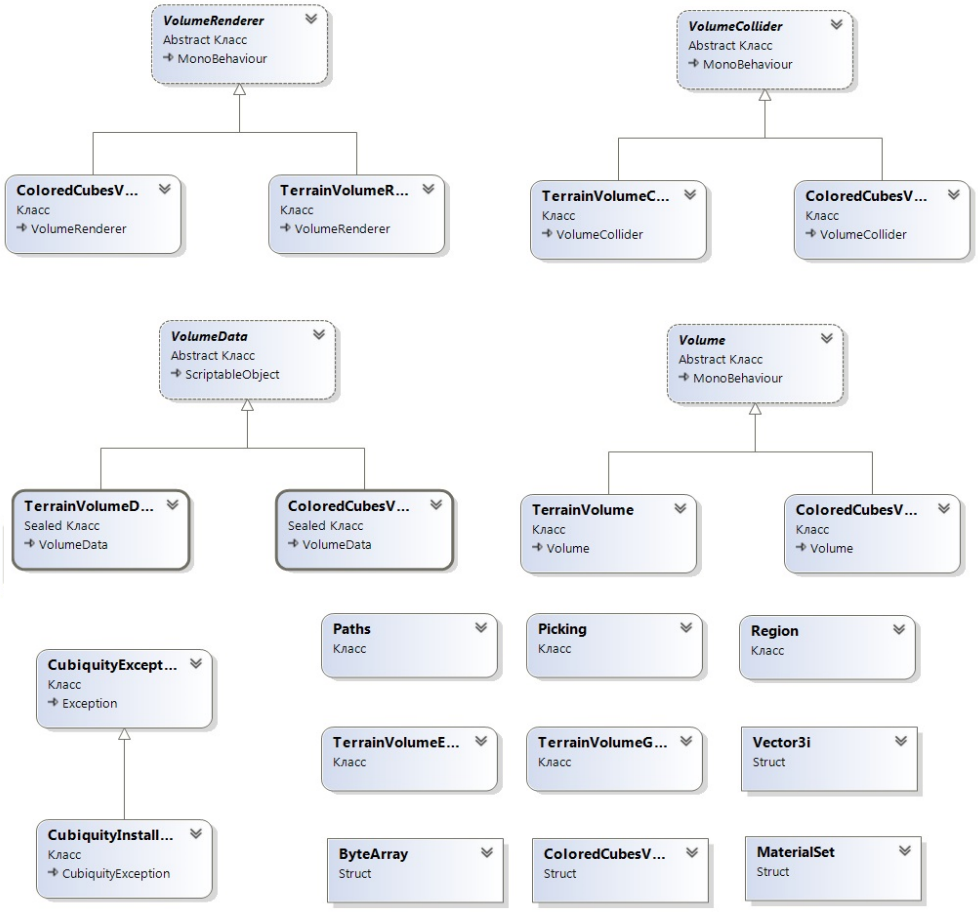


Рисунок 4.1 – Схема структурна класів

Клас програми ColoredCubesVolumeData і клас TerrainVolumeCollider успадковують властивості з класу VolumeCollider. В свою чергу клас ColoredCubesVolumeRenderer і клас TerrainVolumeRenderer успадковують властивості з класу VolumeRenderer. Клас ColoredCubesVolumeData і клас TerrainVolumeData успадковують властивості з класу VolumeData. Клас ColoredCubesVolume і клас TerrainVolume успадковують властивості з класу Volume. Клас CubiquityInstallationException() успадковує властивості з класу CubiquityException().

Специфікацію методів з описанням, а також відповідним класом представлено в таблиці 4.1.

Таблиця 4.1 – Специфікація методів

Клас	Метод	Параметри	Дія
ByteArray	int Length [get]	Кількість елементів у цьому масиві	Надає простий масив байтів з прямим доступом до кожного елементу
	Byte this[uint index] [get, set]	Забезпечує доступ до елементів масиву	
ColoredCubesVolume	new ColoredCubesVolumeData data [get, set]	Представляє собою фактичну 3D-сітку вокселів, що описують ваш об'єкт або навколишнє середовище	Дозволяє створювати середовища з мільйонів кольорових кубиків
ColoredCubesVolumeCollider	-	-	Викликає об'єм кольорових кубів, щоб мати сітку зіткнення і дозволити йому брати участь у зіткненнях

Продовження таблиці 4.1

ColoredCubesVolumeData	Успадковані з VolumeCollider	-	Реалізація VolumeData , яка зберігає Quantized Color для кожного вокселя
ColoredCubesVolumeRenderer	Успадковані з VolumeRenderer	-	Керує деякими візуальними аспектами обсягу кольорових кубиків і дозволяє їх відтворювати
CubiquityException	CubiquityException ()	-	Виявляє помилку, яка виникає всередині бібліотеки власного коду Cubiquity
	CubiquityException (string message)	-	
	CubiquityException (string message, Exception innerException)	-	
MaterialSet	-	-	Представляє комбінацію матеріалів, з яких складається даний воксель

Продовження таблиці 4.1

Paths	-	-	Визначає ряд часто використовуваних шляхів
PickVoxelResult	-	-	Зберігає результат збору вокселя
PickSurfaceResult	-	-	Зберігає результат вибору точки на об'ємної поверхні
Quantized Color	byte red [get, set]	Червона складова кольору	Зберігає приблизне значення кольору з обмеженою глибиною біт
	byte green [get, set]	Зелений компонент кольору	
	byte blue [get, set]	Синій компонент кольору	
	byte alpha [get, set]	Альфа-компонент кольору. Слід встановити тільки на 0 або 255	
Region	-	-	Позначає область тривимірного простору, яка зазвичай представляє кордони для об’єму
TerrainVolume	new TerrainVolumeData data [get, set]	Є фактичною 3D-сіткою вокселів, що описують об'єкт або навколишнє середовище	Дозволяє створювати динамічні ландшафти з печерами і навісами

Продовження таблиці 4.1

Vector3i	int this[int key] [get, set]	Відкриває компонент у вказаному індексі	Тривимірний векторний тип з цілими компонентами
	Vector3i(int x, int y, int z)	Будує вектор з комплекту поставки	
	Vector3i(int x, int y)	Конструює вектор із компонентів, що постачаються (z встановлюється в нуль)	
	Vector3i(Vecto r3i a)	Конструює вектор шляхом копіювання вхідного вектора	
	Vector3i(Vecto r3 a)	Створює вектор шляхом копіювання представленоо вектора і накладаючи компоненти на int	
Volume	VolumeData data [get, set]	Представляє собою фактичну 3D-сітку вокселів, що описують об'єкт або навколишнє середовище	Базовий клас, що представляє поведінку, загальний для всіх об’ємів
	bool isMeshSyncron ized [get, set]	Вказує на те, чи відображається сітки в даний час в актуальному стані даних обсягу	

Продовження таблиці 4.1

VolumeRenderer	Material material[get,set]	Матеріал для цього об'єму	Керує деякими візуальними аспектами об’єму і дозволяє відображати його
	Material_materialLod[get]	-	
	Material materialLod2 [get]	-	
	bool castShadows [get, set]	Контролює, чи цей обсяг створює тіні	
	bool receiveShadows [get, set]	Контролює, чи цей об’єм отримує тіні	
	bool showWireframe [get, set]	Контроль, відображення накладення каркасу, коли цей об’єм вибрано в редакторі	
	float lodThreshold [get, set]	Контроль точки в якій Cubiquity переходить на інший рівень деталізації	
	Int minimum LOD [get, set]	Визначає найнижчий (найменш детальний) LOD, який Cubiquity буде відображати для цього об'єму	

Продовження таблиці 4.1

TerrainVolumeCollider	-	-	Змушує об’єм рельєфу мати колізійну сітку і дозволяє їй брати участь в зіткненнях
TerrainVolumeData	MaterialSet GetVoxel (int x, int y, int z)	Отримує матеріальну вагу вказаної позиції	Реалізація VolumeData, яка зберігає MaterialSet для кожного вокселя
	void SetVoxel (int x,int y, int z,MaterialSet t)	Встановлює матеріальні ваги вказаної позиції	
TerrainVolumeRenderer	-	-	Керує деякими візуальними аспектами обсягу місцевості
VolumeCollider	BooluseInEditMode [get, set]	-	Змушує гучність мати колізійну сітку і дозволяє їй брати участь в зіткненнях
VolumeData	Region enclosingRegion [get]	Отримує параметри VolumeData.	Базовий клас, який має фактичну тривимірну сітку значень вокселів
	string fullPathToVoxelDatabase [get]	Отримує шлях до бази даних вокселя, яка підтримує цей екземпляр	
	WritePermissions writePermissions [get,set]	Контроль запису чи зчитування основної бази даних вокселів	

4.2 Результати генерації тривимірного воксельного ландшафту

Розглянемо безпосередньо згенерований тривимірний воксельний ландшафт із застосуванням Sparse voxel octree, який зображено під різними кутами і з різним масштабом перегляду на рис.4.2. Даний ландшафт не піддається деформації. Демонстраційний варіант передбачає зміну масштабу і поворот ландшафту.

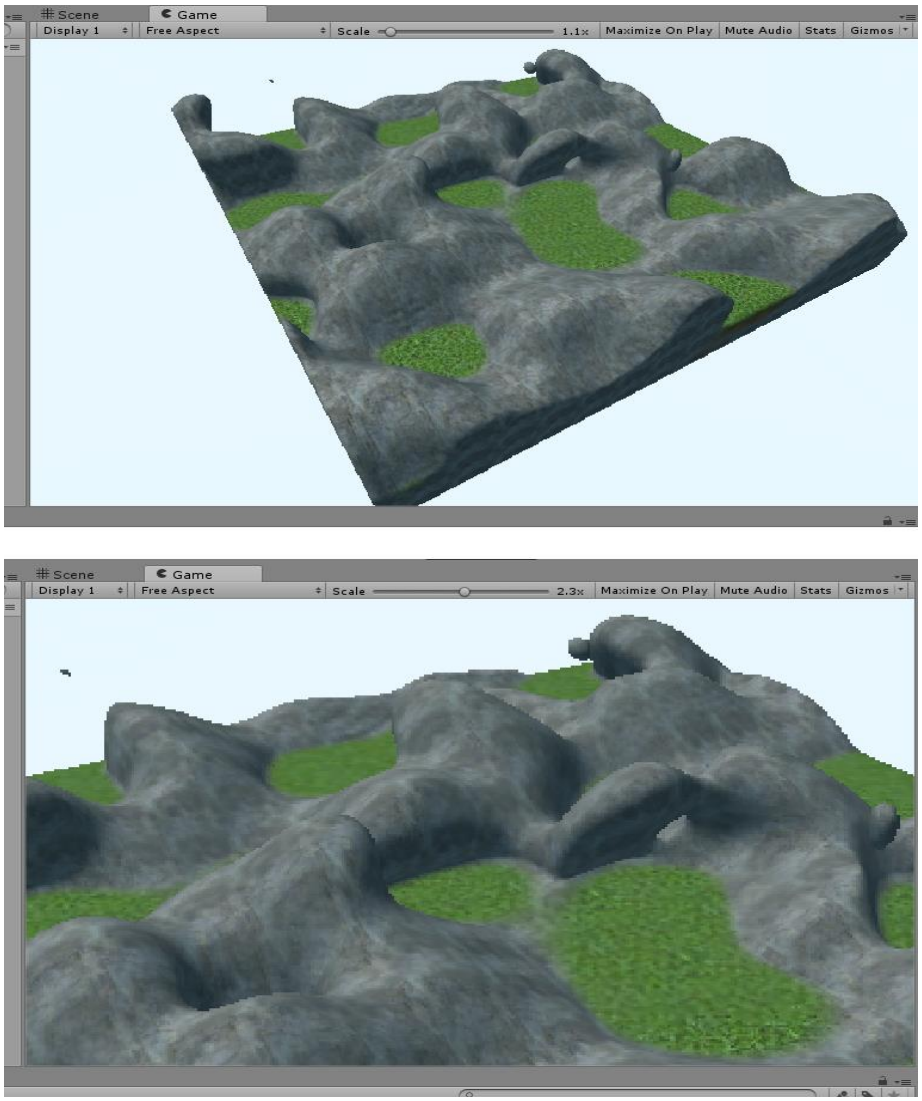


Рисунок 4.2 – Згенерований тривимірний воксельний ландшафт

Далі розглянемо приклад модифікації згенерованого тривимірного воксельного ландшафту із застосуванням Sparse voxel octree в поєднанні з алгоритмом Wire deformation в режимі реального часу. На рис.4.3 представлено згенерований ландшафт до модифікацій. Для демонстрації процесу деформації і її результатів на ландшафт було додано біли кулі, які на початку генерації падають на місцевість згори. Подальший рух куль

запрограмований з врахуванням фізичних законів природи, таких як закон тяжіння.

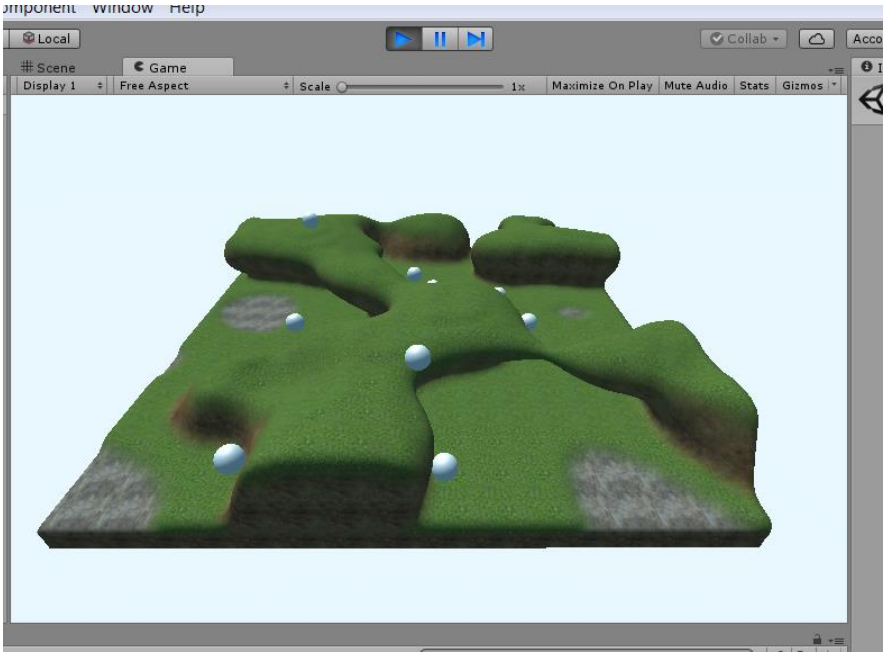


Рисунок 4.3 – Згенерований воксельний ландшафт до модифікації

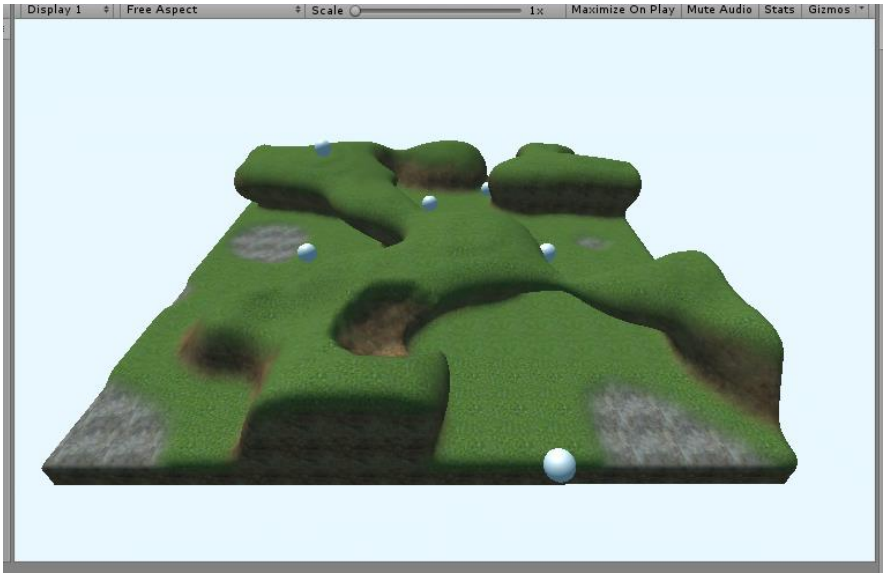


Рисунок 4.4 – Згенерований воксельний ландшафт після незначної модифікації

На рис. 4.4 представленні результати першої спроби деформації ландшафту в реальному часі. При виборі певної зони ландшафту за допомогою кнопки мишки відбувається видалення обраної частини ландшафту. Час натискання визначає наскільки сильною буде модифікація.

Розглянемо послідовний процес модифікації ландшафту представлений на рис.4.5, де (a) – це легка форма деформації валунів; (b) –

створення першої деформації в зоні розташування кулі, яка полягає у видаленні частини місцевості, що призвело до утворення в ній наскрізного отвору , а також падінню кулі в отвір, що демонструє коректність виконаної деформації; (c) – множинна деформація ландшафту; (d) – деформація з видаленням досить великих частин місцевості.

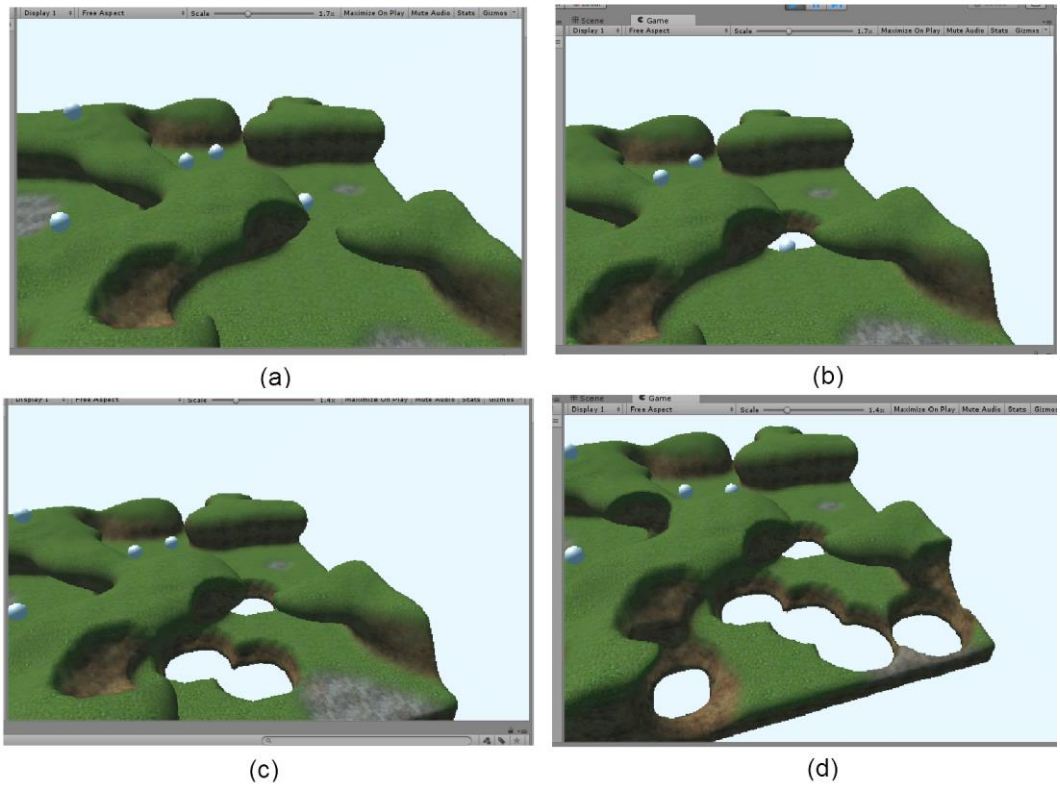
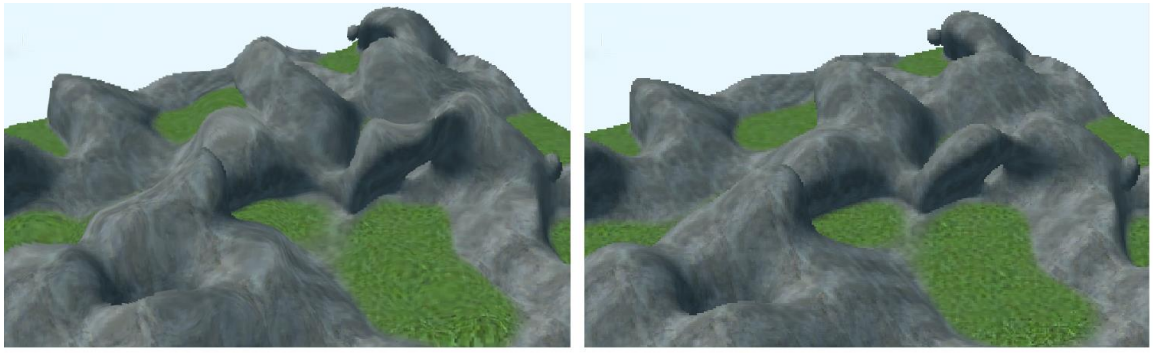


Рисунок 4.5 – Модифікація ландшафту в реальному часі

Також результати роботи програми демонструють ще одну перевагу генерації воксельного ландшафту, яка полягає в тому, що, на відміну від полігонального моделювання, ландшафт не перемальовуються заново, тобто відбувається оптимізація використаних ресурсів.

Проведемо експеримент, згенерувавши ландшафт без застосування Sparse voxel octree. На рис.4.6 зображено два варіанти ландшафту, де (a) – ландшафт, згенерований без застосування Sparse voxel octree, (b) – ландшафт, згенерований із застосуванням цього алгоритму.



(a) (b)

Рисунок 4.6 – Генерація ландшафту з використанням і без використання Sparse voxel octree

Помітно, що місцевість, згенерована із застосуванням Sparse voxel octree виглядає більш природньо за рахунок згладженої поверхні. Особливо недоліки генерації помітні при збільшенні масштабу на зоні трави (рис.4.7).

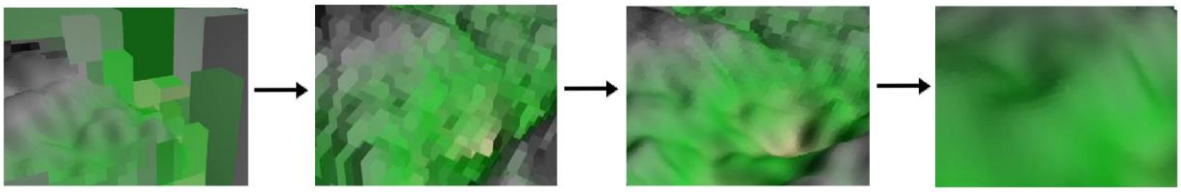


Рисунок 4.7 – Демонстрація роботи використання Sparse voxel octree при збільшеному масштабі

4.3 Оцінка використання розрідженого воксельного октодерева при генерації ландшафтів в ігрових програмах

Було проведено ряд експериментів для оцінки генерації ландшафту і методу побудови Sparse voxel octree на основі вокселів. У табл.4.2 показана складність типового октодерева, отриманого з моделі 3D-ландшафту. Модель ландшафту збудована з використанням сферичного фундаменту з величиною зміщення 0,05. У таблиці дається порівняння між загальною кількістю кордонних вокселів в октеті, а також розбиття кордонних вокселів, що містять камінь і повітря. Без використання Sparse voxel octree загальна кількість вокселів, які повинні бути збережені в пам'яті, дорівнюватиме $2^{\text{глибина}} \times 2^{\text{глибина}} \times 2^{\text{глибина}}$. Можна побачити, що загальна кількість вокселів октодерева значно зменшує вимоги до пам'яті. Крім того,

загальна кількість вершин і трикутників, що утворюють поверхню ландшафту, також представлено в табл.4.2.

Таблиця 4.2 – Порівняння кількості вокселів, вершин і трикутників для різних глибин октодерева

Глибина октодерева	Граничні вокселі			Вершини	Трикутники
	Камінь	Повітря	Загально		
5	250	172	422	674	1296
6	1012	606	1618	2693	5196
7	4235	2284	6519	11135	21376
8	17858	9501	27359	47756	90760
9	73760	43766	117526	207836	388084

Також результати окремого дослідження використання пам’яті наведено в табл.4.3 разом з кінцевим розміром сховища для кожної моделі рельєфу. Метрики були зібрані для плити 1 (одна з найменших плит), плити 7 (єдина плита, що охоплює 100 на 100 метрів) і плити 1-9 (стрес-тест на широку область).

Таблиця 4.3 – Підсумок результатів

Модель ландшафту	Плита 1 (МБ)	Плита 7 (МБ)	Плита 1-9 (МБ)
UPC Points	57	184	1,080
ASCII Points	82	267	1,610
Polygonal/TIN	39	126	779
Sparse Voxel	0.634	3.0	1,3

Як і очікувалося, обсяг сховища моделі з прив'язкою до сітки набагато перевершував будь-який інший формат генерації. Результати доводять, що серед різних полігональних і об'ємних наборів даних місцевості Sparse voxel octree виконується найкращим чином відносно загальної метрики розміру сховища на диску.

4.1 Висновки

Для демонстрації процесу генерації ландшафту в ігрових програмах за допомогою вокселів з використанням Sparse voxel octree, було розроблено і реалізовано програмний продукт – воксельний ландшафт з можливістю динамічних модифікацій в режимі реального часу.

Розробка проводилася в середовищі Unity за допомогою мови програмування C# (C Sharp).

Було спроектовано діаграму класів програми, а також описано класи програми з відповідними методами.

Було згенеровано тривимірний воксельний ландшафт із застосуванням алгоритму Sparse voxel octree з можливістю зміни масштабу і повороту, а також модифікації в режимі реального часу за рахунок використання алгоритму Wire deformation.

Був проведений експеримент генерації ландшафту з використанням Sparse voxel octree і без його використання. Результати довели ефективність його використання.

Було проведено ряд експериментів для оцінки процесуальної генерації печер і методу побудови Sparse voxel octree на основі вокселів, в результаті чого було доведено, що загальна кількість вокселів октодерева значно зменшує вимоги до пам'яті, а також те, що серед різних полігональних і об'ємних наборів даних місцевості, Sparse voxel octree виконується найкращим чином відносно загальної метрики розміру сховища на диску.

ВИСНОВКИ

Метою даної магістерської дисертації було підвищення ефективності процесу генерації ландшафту в ігрових програмах за рахунок використання розрідженого воксельного октодерева в поєднанні з алгоритмом Wire deformation.

Дослідження, виконане в даній магістерській дисертації, дозволило зробити наступні висновки:

1. Використання Sparse voxel octree є найкращим варіантом для генерації ландшафту в ігрових програмах з використанням октодерева;

2. Sparse voxel octree найкращий в зменшенні вимог до пам’яті в порівнянні з прямою воксельної матрицею, яка вимагає 8 ГБ для зберігання дискретного набору даних вокселів 2048×2048×2048;

3. Описаний і реалізований алгоритм Wire deformation дозволив генерувати за допомогою Sparse voxel octree об'єкти великої площі. Замість зменшення сітки, використовуючи алгоритм Marching Cubes, було використано зменшене значення висоти, що зменшило гострі краї;

4. Практична цінність отриманих в роботі результатів полягає в тому, що запропонований алгоритм генерації ландшафту з використанням розрідженого воксельного октодерева в ігрових програмах підвищує основні показники ефективності генерації ландшафту, а саме: точність елементів вихідного ландшафту, а також зменшує вимоги до пам'яті;

5. Використання розрідженого воксельного октодерева в ігрових програмах дозволило генерувати ландшафт без потреби його перемальовування в залежності від зміни кута повороту або модифікацій.

Таким чином, результати дослідження, виконаного в магістерській дисертації, дозволяють більш ефективно створювати візуальну частину гри, що призводить до підвищення якості програмного продукту, а отже його подальшого розвитку і розповсюдження.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. F. Kenton Musgrave, Craig E. Kolb and Robert S. Mace: The Synthesis and Rendering of Eroded Fractal Terrains. Computer Graphics, Volume 23, Number 3, July 1989, pages 41-50.

2. Kenji Nagashima: Computer generation of eroded valley and mountain terrains. The Visual Computer, 1997, pages 13:456-464.

3. David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin and StevenWorley: Texturing and Modeling: A Procedural Approach (Third Edition). Morgan Kaufmann Publishers, 2003.

4. Belhadj, F.: Terrain modeling: A constrained fractal model. In: Proceedings of the 5th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa, pp. 197–204 (2007)

5. Doran, J., Parberry, I.: Controlled procedural terrain generation using software agents. IEEE Transactions on Computational Intelligence and AI in Games 2(2), 111–119 (2010)

6. Scenery generator [Електронний ресурс]. – 2013. – Режим доступу: https://www.revolvy.com/page/Scenery-generator

7. Sparse voxel octree [Електронний ресурс]. – 2010. – Режим доступу: https://en.wikipedia.org/wiki/Sparse_voxel_octree

8. Реферат: Поняття фракталів [Електронний ресурс]. – 2012. – Режим доступу: https://www.bestreferat.ru/referat-185151.html

9. Diamond-square algorithm [Електронний ресурс]. – 2014. – Режим доступу: https://en.wikipedia.org/wiki/Diamond-square_algorithm

10. L-система [Електронний ресурс]. – 2012. – Режим доступу: https://ru.wikipedia.org/wiki/L-система

11. Grome 3.1: генератор ландшафтов [Електронний ресурс]. – 2011. – Режим доступу: https://3dnews.ru/618577

12. The standard for vegetation modeling and middleware [Електронний ресурс]. – 2017. – Режим доступу: https://store.speedtree.com/

13. Esri CityEngine [Електронний ресурс]. – 2017. – Режим доступу: https://www.esri.com/en-us/arcgis/products/esri-cityengine/overview

14. ZBrushCentral [Електронний ресурс]. – 2018. – Режим доступу: http://www.zbrushcentral.com/

15. 3ds Max [Електронний ресурс]. – 2016. – Режим доступу: https://www.autodesk.ru/products/3ds-max/overview

16. Cycles Render Engine [Електронний ресурс]. – 2017. – Режим доступу: https://www.blender.org/

17. Stop hand sculpting 3d terrain [Електронний ресурс]. – 2016. – Режим доступу: https://www.world-machine.com/

18. Render beautifully realistic CG environments [Електронний ресурс]. – 2014. – Режим доступу: https://planetside.co.uk/

19. Tilemap [Електронний ресурс]. – 2015. – Режим доступу: https://docs.unity3d.com/ScriptReference/Tilemaps.Tilemap.html

20. Двох з половиною вимірний опис об'єкта [Електронний ресурс]. – 2018. – Режим доступу: https://uk.wikipedia.org/wiki/Двох_з_половиною_вимірний_опис_об%27єкта

21. OpenGL Headline News [Електронний ресурс]. – 2018. – Режим доступу: https://www.opengl.org/

22. Direct3D 10 [Електронний ресурс]. – 2015. – Режим доступу: https://uk.wikipedia.org/wiki/Direct3D_10

23. Terrain LOD: Runtime Regular-Grid Approaches [Електронний ресурс]. – 2017. – Режим доступу: http://vterrain.org/LOD/Implementations/

24. Perlin noise [Електронний ресурс]. – 2010. – Режим доступу: https://en.wikipedia.org/wiki/Perlin_noise

25. Robot frog [Електронний ресурс]. – 2002. – Режим доступу: https://www.stuffwithstuff.com/robot-frog/3d/hills/hill.html

26. Методи моделювання та експериментів для досліджень проявів водної ерозії [Електронний ресурс]. – 2014. – Режим доступу: http://geoknigi.com/book_view.php?id=1605

27. B. Lichtenbelt, R. Crane, S. Naqvi: “Introduction to Volume Rendering” (Hewlett-Packard Professional Books), Hewlett-Packard Company 1998.

28. T. N. Raju: “The Nobel chronicles. 1979: Allan MacLeod Cormack (b 1924); and Sir Godfrey Newbold Hounsfield (b 1919”), Lancet, 1999.

29. Samet, H.: The design and analysis of spatial data structures. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990. ISBN 0-201-50255-0.

30. Agarwal, P.K. and Erickson, J.: Geometric range searching and its relatives. In: Advances in Discrete and Computational Geometry, pp. 156. American Mathematical Society, 1999.

31. An Implementation of the Marching Cubes Algorithm [Електронний ресурс]. – 2012. – Режим доступу: http://www.cs.carleton.edu/cs_comps/0405/shape/marching_cubes.html

32. Bump mapping [Електронний ресурс]. – 2014. – Режим доступу: https://en.wikipedia.org/wiki/Bump_mapping

33. Тексел (графика) [Електронний ресурс]. – 2016. – Режим доступу: https://ru.wikipedia.org/wiki/Тексел_(графика)

34. Fragment Shader [Електронний ресурс]. – 2010. – Режим доступу: https://www.khronos.org/opengl/wiki/Fragment_Shader

35. Blinn, J.F.: Models of light reection for computer synthesized pictures. SIGGRAPH Computer Graph- ics, vol. 11, no. 2, pp. 192198, 1977. ISSN 0097-8930.

36. Galyean, T.A. and Hughes, J.F.: Sculpting: An interactive volumetric modeling technique. In: Computer Graphics (Proceedings of SIGGRAPH 91), vol. 25, pp. 267274. July 1991.

37. Musgrave, F., Kolb, C. and Mace, R.: The synthesis and rendering of eroded fractal terrains. Computer Graphics, vol. 23, no. 3, pp. 4150, 1989.

38. Zhu, Y. and Bridson, R.: Animating sand as a uid. In: SIGGRAPH '05: ACM SIGGRAPH 2005 Papers, pp. 965972. ACM Press, New York, NY, USA, 2005.

39. Benes, B., Tínský, V., Horny, J. and Bhatia, S.K.: Hydraulic erosion. Computer Animation and Virtual Worlds, vol. 17, pp. 99108, 2006.

40. Побудова фрактальних поверхонь в комп'ютерній графіці [Електронний ресурс]. – 2010. – Режим доступу: https://refdb.ru/look/1482946-pall.html

41. Milliron, T., Jensen, R.J., Barzel, R. and Finkelstein, A.: A framework for geometric warps and deformations. ACM Transactions on Graphics, vol. 21, no. 1, pp. 2051, 2002. ISSN 0730-0301.

42. Singh, K. and Fiume, E.: Wires: a geometric deformation technique. In: SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pp. 405414. ACM Press, New York, NY, USA, 1998. ISBN 0-89791-999-8.

43. Schneider, P.J: Graphics Gems, chap. 8, pp. 408415. Academic Press, Inc., Orlando, FL, USA, 1990. ISBN 0122861655