МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Львівський національний університет імені Івана Франка

Факультет електроніки та комп'ютерних технологій

Кафедра радіофізики та комп'ютерних технологій

Допустити до захисту

Завідувач кафедри

_____ проф. Карбовник І.Д.

«___»_____2023р.

Кваліфікаційна робота

Бакалавр

"Використання концепції фракталів у комп'ютерній графіці"

Виконав

Студент групи ФЕП-41

Спеціальності:

121 Інженерія програмного забезпечення

_____ Бойко А. Р.

Науковий керівник:

_____ проф. Болеста І.М.

«___»_____2023р.

Рецензент:

_____ Доцент Шувар Р.Я.

Львів 2023

АНОТАЦІЯ

Метою цієї бакалаврської роботи є аналіз застосуванню фракталів в комп’ютерній графіці та створення програмного застосунку для дослідження фрактальних ландшафтів. Для легкої взаємодії з фракталом реалізовано графічний інтерфейс в якому можна динамічно працювати з фрактальним ландшафтом.

Для створення актуальної розробки було проаналізовано основні проблеми реалізації таких застосунків, розглянуто існуючі аналоги та обрано підхожі засоби для розробки.

При виконанні роботи також було здійснено тестування програмної розробки та проаналізовані його результати для виявлення можливих недоліків системи та шляхів їх подолання.

ABSTRACT

The purpose of this bachelor thesis is to analyze the use of fractals in computer graphics and to create a software application for the study of fractal landscapes. For easy interaction with the fractal, a graphical interface has been implemented in which you can dynamically work with the fractal landscape.

To create an actual development, the main problems of implementing such applications were analyzed, existing analogues were considered, and suitable development tools were selected.

In the course of the work, we also tested the software development and analyzed its results to identify possible system shortcomings and ways to overcome them.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ПЗ	Програмне Забезпечення.
ММ	Математичний монстр.
ФВ	Функція Вейєрштрасса.
RANGE	Діапазон.
DS	Diamond-square algorithm – алгоритм діамант-квадрат.
LC	List-Comperhension, спосіб для створення списків в python.
КВ	Карта величин, висот, позначення для змінної ‘square_terrain’.
GUI	Графічний інтерфейс користувача.
ООП	Об’єктно-орієнтоване програмування.
IT	Інформаційні технології.
GIT	Розподілена система керування версіями файлів.
GIL	Global Interpreter Lock – глобальне блокування інтерпретатору.
ФЛ	Фрактальний ландшафт.
PYTHON	Мова програмування.

ЗМІСТ

ВСТУП

Термін фрактал був введений у 1975 році французьким математиком Бенуа Мандельбротом у його книзі “Fractals: Form, Chanse and Dimension”(перекладена англійською мовою у 1977 році) [2].

Фрактал(лат. Fractus – роздріблений, дробовий) – в поширеному розумінні часто означає деяку нерегулярну, самоподібну структуру. Для більш точного визначення фракталу потрібні глибокі знання з курсів алгебри та математичного аналізу. Фрактали зазвичай розуміють як самоподібні множини, тобто множини, що складаються з подібних частин до неї самої. Це проста геометрична фігура яку зазвичай визначають, або малюють, або генерують за допомогою рекурсивних правил. Рекурсивні правила означають що вони застосовуються раз за разом, до якоїсь межі або вже до нескінченності. Рекурсія це не завжди оптимально для програмування, але при роботі з фракталами, це неминуче.

З початку другої половини XX століття, концепція фракталів починає активно використовуватися в ІТ.

Актуальність проблеми: “Фрактальна геометрія” це досить молода, маловідома і мало досліджена галузь геометрії, яка вивчає фрактали. Але через те що фрактали здатні описувати деякі навколишні об’єкти з великою точністю, та явища, це дуже перспективна галузь.

Мета роботи полягає в тому щоб ознайомитись з теорією фракталів та розробити застосунок для дослідження фрактальних ландшафтів.

Для досягнення мети необхідно:

- Провести літературний огляд по цій темі бакалаврської роботи.
- Вивчити питання розмірності фракталу.
- Ознайомитись з алгоритмами котрі застосовують концепцію фракталів.
- Ознайомитись з деякими застосунками котрі застосовують концепцію фракталів.
- Розробити власний застосунок із використанням концепції фракталів.

РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА

1.1. Історія появи фракталів

Історія фракталів простежує шлях від переважно теоретичних досліджень до сучасних застосувань у комп'ютерній графіці, причому кілька видатних людей зробили свій внесок у фрактальні форми на цьому шляху, а саме: Кантор, Пеано, Лебег, Гаусдорф, Серпінський, Мандельброт та інші. Поширеною темою африканської архітектури є використання фрактального масштабу. Це приводить до того, що малі частини будівлі виглядають так само, як і великі частини, наприклад, кругле село, що складається з круглих будинків.

Об'єкти, що виникають при створенні фракталів, вивчалися задовго до появи самого терміну "фрактал". Такі етноматематики, як Рон Егласс, в своїй роботі "Африканські фрактали" документує теперішні поширені геометричні фігури, які виникають із фрактальної побудови в місцевому мистецтві тубільців. У 1525 році німецький митець Альберт Дюрер опублікував свою працю "Керівництво Художника", один із розділів якої має назву "Черепичні шаблони, утворені п'ятикутниками". П'ятикутник Дюрера багато в чому схожий на килим Серпінського, але з п'ятикутниками замість чотирикутників(квадрат). Джексон Поллок(американський експресіоніст 50-х років минулого століття) малював об'єкти, схожі на ті, що відбуваються при створенні фракталів.

Згідно з Пікавером, математика, що лежить в основі фракталів, почала формуватися в 17 столітті, коли математик і філософ Готфрід Лейбніц розмірковував про рекурсивну самоподібність(хоча він припустився помилки, вважаючи що тільки пряма лінія є самоподібною в цьому сенсі) [1, 4].

Лейбніц використав термін "дробові експоненти"(fractional exponents), але геометрія цього терміну ще не знала [1]. Дійсно, згідно з різними історичними повідомленнями, після цього мало хто з математиків займався цими питаннями, а роботи тих, хто це робив, залишалися в тіні, в основному через опір таким незнайомим новим концепціям, які з'являлися, іноді їх називали "математичними монстрами". Ідею "рекурсивної самоподібності" було висунуто філософом Лейбніцем, який розвинув багато деталей цієї ідеї.

Всього через два століття, 18 липня 1872 року, математик Карл Вейєрштрасс, знаний як "Батько математичного аналізу", через свій акцент на суворості математики, дійшов до відкриття функції, яка поставила під сумнів праці великих математиків і кардинально змінила представлення математиків про математичний аналіз. У Прусській королівській академії наук, Вейєрштрасс описав ММ, шокуючу для всіх функцію, яка суперечить загальновизнаним теоремам, та представив перше визначення функції з графіком, який сьогодні вважався б фракталом, що має інтуїтивно незрозумілу властивість бути всюди неперервною, але не є

диференційованою в жодній точці (Рис. 1.1). Він викладав багато предметів і багато студентів отримали користь від його викладання [11].

$$f(x) = \sum_{n=0}^{\infty} a^n \cos(b^n \pi x)$$

Ця функція це нескінченний ряд суми косинусів.

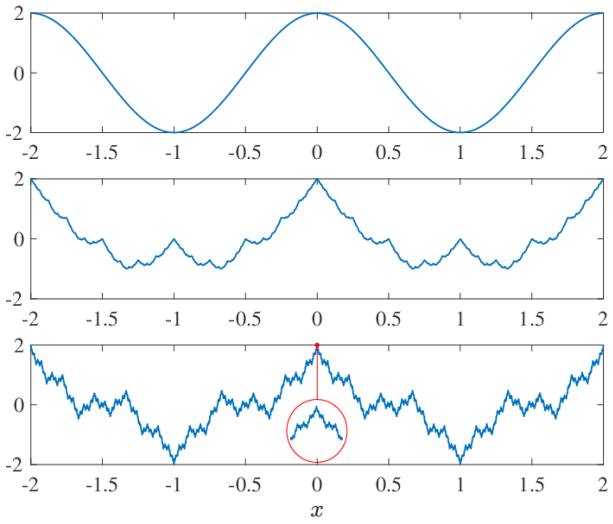


Рис. 1.1. Візуалізація функції Вейєрштрасса [26]

Математикам того часу було тяжко це уявити, вона ламала порядок і такі функції не хотіли помічати, вдавали що це просто якесь непотрібне відхилення яке тільки псує математичний аналіз замість додавання чогось корисного. Це була функція яка незрозуміло як виглядає, тим паче в той час було неможливо її намалювати.

Чим більше доданків тим більше хвиль, коли доданків нескінченно багато коливань та різких змін відбувається ще більше, в кожній точці ФВ різко змінює свій напрямок на протилежний, змінює напрямок зростання чи падіння, тому в жодній точці не можна порахувати похідну.

ФВ суперечила загальновизнаним результатам, наприклад теоремі Ампера про те, що кожна неперервна функція має бути диференційована, хоча вона була доказана чисто аналітичним методом, і по сьогоднішнім міркам за доведення його не прийняли б. І стиль доказу Вейєрштрасса був невідомим для багатьох. ФВ ламала уявлення про суть математичного аналізу, вона суперечила геометричній інтуїції, коли функції були аналогами якихось явищ в реальному світі. Але ФВ нічого такого не описувала, просто страшна, складна, марна функція, яка відірвана від реальності, як тоді здавалось, але згодом виявилось що це зовсім не так.

Всього через 30 років після публікації, все починає кардинально змінюватися, ММ починають впевнено захоплювати науковий світ. ФВ дала поштовх для теорії фракталів. Після нього все більше і більше математиків стали робити свої ММ, подібні до ФВ.

Вона перевернула погляд людства на зв’язок математики та реального світу та зробила великий внесок у математику, допомогла створити нові розділи, практичні та прикладні.

Невдовзі після цього, у 1883 році, Георг Кантор, який відвідував лекції Вейєрштрасса, опублікував приклади підмножин дійсної прямої, відомої як множини Кантора (Рис. 1.2), які мали незвичні властивості і тепер визнані фракталами. За допомогою простої рекурсивної процедури перетворив лінію в набір незв’язаних точок(Пил Кантора або пряма Кантора). Він брав лінію і видаляв центральну третину і після цього повторяв те ж саме з відрізками які залишились [6].

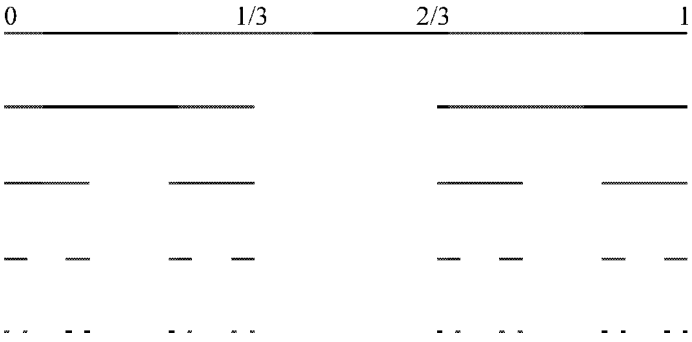


Рис. 1.2. Множина Кантора

Пеано намалював особливий вид лінії (Рис. 1.3). Для її малювання Пеано використав наступний алгоритм. На першому кроці пряма лінія заміняється на 9 відрізків довжиною в 3 рази менше, ніж довжина вихідної лінії. Далі робиться те ж саме з кожним відрізком лінії, що вийшла. І так до нескінченності. Її унікальність в тому, що вона заповнює всю площину, тобто її розмірність дорівнює 2 (про розмірність далі в окремому розділі). Доказано, що для кожної точки на площині можна знайти точку, яка належить лінії Пеано.

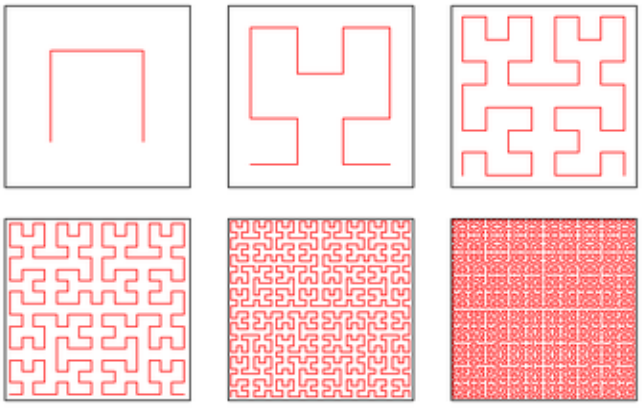
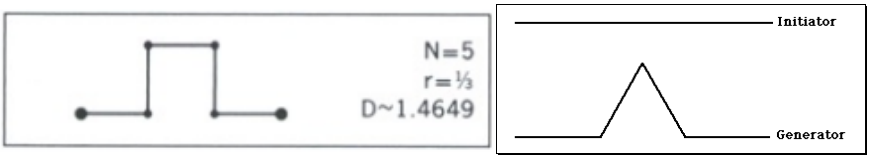


Рис. 1.3. Крива Пеано

Крива Пеано і порох Кантора виходили за рамки звичайних геометричних об’єктів. Вони не мали чіткої розмірності. Порох Кантора будувався на підставі одномірної прямої, але складався із точок, а крива Пеано будувалась на підставі одномірної лінії, а в результаті

виходила площина. В багатьох інших областях науки з'являлись задачі, рішення яких приходило до дивних результатів, на кшталт броунівського руху, ціни на акції [6].

Одна з наступних віх настала в у 1904 році коли шведська математичка Гельге фон Кох, розвиваючи ідеї Пуанкере і незадоволена надто абстрактно-аналітичним визначенням Вейєрштрасса, розробила більш геометричне визначення подібної функції, включаючи намальовані від руки зображення функції, яка тепер називається "Сніжинка Кох" (Рис. 1.5), яка властивість самоподібності. Ця фігура має скінченну площу обмежену нескінченною окружністю. Є два варіанти сніжинки Кох. Перша – крива площини, отримана, коли ініціатором є квадрат і генератор є Рис. 1.4:



а)генератор квадрат б)генератор трикутник

Рис. 1.4. Генератор для сніжинки Кох [1]

а)квадрат б)трикутник

Рис. 1.5. Сніжинка Кох побудована із генератором [1]

Ще одна віха настала через десять років, в 1915 році, коли польський математик Вацлав Серпінський побудував свій знаменитий трикутник (Рис. 1.6) – крива всі точки якої є точками відгалудження, а потім, роком пізніше – свій килим (Рис. 1.7), по тій же самій методиці. Насправді, його трикутник можна ще побачити в середньовічних замках чи церквах як гарний мозаїчний візерунок. Схема для трикутника така, все починається з рівностороннього заповненого трикутника, через вершини які знаходяться на серединах його відрізків, вирізається трикутник, залишаються три заповнені трикутники для яких виконується ця ж процедура [13]. Також можна домальовувати замість того щоб вирізати. Ще є досить цікавий спосіб, зовсім неінтуїтивний котрий має назву "Гра хаосу". Малюються точки вершин трикутника, ставиться точка в області трикутника, далі вибирається одна із трьох вершин, і малюється нова точка на середині відстані між тими двома, далі з цієї точки вибирається знову люба вершина трикутника і виконується ця цю ж операція. Проробивши її досить разів,

близько 1000, вималюється знайомий контур. Ці правила можна задати за допомогою матриць [19].

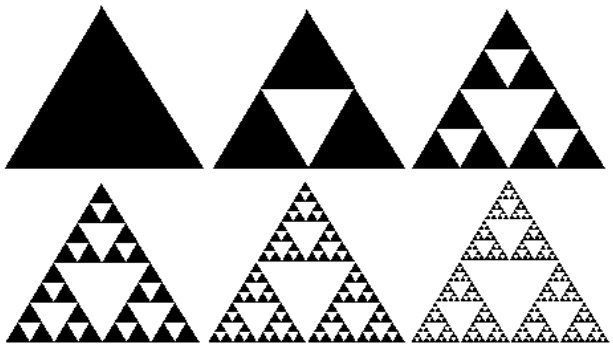


Рис. 1.6. Трикутник Серпінського

Рис. 1.7. Килим Серпінського

До 1918 року два французькі математики, П’єр Фату і Гастон Жюліа, хоч і працювали незалежно, але, по суті, одночасно отримали результати, що описують те, що зараз розглядається як фрактал з відображенням комплексних чисел та ітеративних функцій, що призвело до подальших ідей про атрактори і репелери (тобто точки, які притягують або відштовхують інші точки), які стали дуже важливими у вивченні фракталів.

Невдовзі після цієї роботи, до березня 1918 року, Фелікс Гаусдорф розширив визначення “розмірності”, що мало велике значення для еволюції визначення фракталів, дозволивши множинам мати нецілі розмірності. Ідея самоподібних кривих була далі розвинена Полем-П’єром Леві, який описав нову фрактальну криву у своїй книзі 1938 року “Площинні та просторові криві та поверхні, які складаються із частин, схожих на ціле” (Plane or Space Curves and Surfaces Consisting of Parts Similar to the Whole), опублікованій цього ж року. Вона стала відомою як крива Леві – трикутний фрактал з рівномірним центром мас.

Різні дослідники припускають, що без допомоги сучасної комп’ютерної графіки перші дослідники були обмежені тим, що вони могли зобразити на ручних малюнках, тому їм не вистачало засобів, щоб візуалізувати красу і оцінити деякі наслідки багатьох відкритих ними закономірностей (наприклад, множину Жюліа можна було візуалізувати лише за допомогою декількох ітерацій) [1].

Аж до 20 століття відбувалось накопичення даних про такі дивні об’єкти, без будь якої спроби їх систематизувати. Так було, поки за них не взявся Бенуа Мандельброт – батько

сучасної фрактальної геометрії і терміну фрактал. Працюючи в IBM математичним аналітиком, він вивчав шуми в електронних схемах, які неможливо було описати за допомогою статистики. Поступово співставивши факти, він відкрив новий напрямок в математиці – фрактальна геометрія [6].

Сам Мандельброт вивів термін fractal від латинського слова fractus, що означає розбитий, поділений на частини. І одне з означень фрактала – це геометрична фігура, яка складається із частин і яка може бути поділена на частини, кожна з яких буде представляти зменшену копію цілого(принаймні приблизно).

У 1975 році Мандельброт, на основі цих всіх робіт, вивів термін ‘fractal’ від латинського слова fractus, що означає розбитий, поділений на частини. І одне з означень фрактала – це геометрична фігура, яка складається із частин і яка може бути поділена на частини, кожна з яких буде представляти зменшену копію цілого(принаймні приблизно). Мандельброт ввів термін ‘fractal’, розмірність Гаусдорфа яких перевищує їх топологічну розмірність. Він пояснив математичне визначення за допомогою захоплюючих комп’ютерних візуалізацій. Ці візуалізації, засновані на рекурсії, що сприяло поширеному розумінню поняття “фрактал”.

У 1980 році Лорен Карпентер виступив з доповіддю SIGGRAPH, де представив своє програмне забезпечення для генерації та візуалізації фрактальних ландшафтів.

Як тільки Мандельброт відкрив концепцію фракталів, виявилося, що світ буквально оточений ними. Фрактальні металеві бруски і гірські породи, візерунок листя, капілярна система рослин, фрактальна структура гілок. Кровоносна, лімфатична, нервова системи в організмах тварин, хмарні поверхні, фрактальні річкові басейни, гірський рельєф, океанські узбережжя і так далі.

1.2. Геометрія природи

Класична математика сягає своїм корінням у регулярні геометричні структури Евкліда та неперервної динаміки Ньютона, що постійно розвивається. Сучасна математика почалася з теорії множин Кантора та кривої заповнення простору Пеано. Історично, революцію в математиці спричинило відкриття математичних структур, які не вписувалися в моделі Евкліда і Ньютона. Ці нові структури розглядалися, як “патологічні”, як “галерея монстрів”, Математики, які створили цих ММ, вважали їх важливими для демонстрації того, що світ чистої математики містить багатство можливостей, що виходять далеко за межі простих структур, які вони бачили в природі. Математика двадцятого століття розквітла в переконанні, що вона повністю подолала обмеження, накладені її природним походженням [1].

Тепер, як зазначає Мандельброт, природа зіграла жарт з математиками. Математикам 19-го століття, можливо, бракувало уяви, але Природі – ні. Ті ж самі патологічні структури, які математики винайшли, щоб вирватися з натуралізму 19-го століття, виявляється притаманні знайомим об'єктам навколо нас. Він підтвердив спостереження Блеза Паскаля про те, що уява втомлюється перед природою.

Тим не менш, фрактальна геометрія не є прямим "додатком" математики 20-го століття. Це нова галузь, що запізно народилася з кризи математики, яка розпочалася, коли Дюбуа Реймонд у 1875 році вперше повідомив про неперервну недиференційовану функцію, побудовану Вейєрштрассом. Криза тривала приблизно до 1925 року, головними дійовими особами були Кантор, Пеано, Лебег та Гаусдорф. Ці імена, а також імена Бесіковича, Больцано, Чезаро, Кох, Осгуда, Серпінського та Урисона, зазвичай не зустрічаються в емпіричному вивченні природи, але на думку Мандельброта, вплив робіт цих гігантів далеко виходить за рамки передбачуваного обсягу [1].

"Мова математики виявляється безпідставно ефективною в природничих науках, чудовий дар, якого ми не розуміємо і не заслуговуємо." (Вігнер, 1960) [1].

"Хмари не є сферами, гори не є конусами, береги не є колами, а кора не є гладкою, і блискавка не рухається по прямій лінії" (Мандельброт 1983) [1].

1.3. Види фракталів

Взагалом фракталом можна назвати предмет який володіє одним із зазначених властивостей:

- Має складну нерегулярну структуру яка має самоподібність на всіх масштабах. Це те, що відрізняє його від звичайних фігур, таких як коло чи еліпс. Якщо дивитись на невеликий фрагмент регулярної фігури у великому масштабі, то він буде виглядати як фрагмент прямої. Для фракталу, збільшення масштабу не веде до спрощення структури, і всі масштаби будуть показувати однаково складну картину.
- Є самоподібним або наближено самоподібним.
- Володіє дробовою метричною розмірністю.

Розрізняють 5 типів самоподібності фракталів(три перших з яких основні) [25]:

- Точна самоподібність – найсильніший тип самоподібності. Фрактали виглядають однаково при різних збільшеннях, наприклад сніжинка Кох. Фрактали створені за допомогою ітераційних функцій, часто демонструють точну самоподібність.

- Квазі(quasi) самоподібність – слабка форма самоподібностей, наближає той самий патерн у різних масштабах. Може містити малі копії всього фракталу у спотворених і вироджених формах. Наприклад, сателіти множини Мандельборта є наближеними версіями всієї множини, але не її точнимим копіями. Фрактали створені за допомогою рекурентних співвідношень, зазвичай приблизно(хоча і не зовсім) самоподібними.
- Статистична самоподібність – це найслабша форма самоподібності. Повторює шаблон стохастично, так що числові або статистичні міри зберігаються в різних масштабах. Наприклад, випадково згенеровані фрактали як вже добре відомий приклад лінії узбережжя Британії, для якої не можна було б очікувати знайти сегмент, масштабований і повторюваний так само акуратно, як повторювана одиниця, що визначає фрактали, такі як сніжинка Кох. Стохастичний фрактал є прикладом фракталу, який є статистичним фракталом, але не є однаковим або дуже схожим
- Якісна самоподібність: як у часовому ряді
- Мультифрактальне масштабування: характеризується більш ніж одним фрактальним виміром або правилом масштабування

В основному фрактали класифікують за трьома видами:

- Алгебраїчні фрактали.
- Геометричні фрактали.
- Стохастичні фрактали.

1.3.1. Алгебраїчні фрактали

Алгебраїчні фрактали – це найбільша група фракталів, названих за їх використання в алгебраїчних формулах, методах. Їх також часто використовують для фрактального живопису.

Одним із способів отримання алгебраїчних фракталів є багаторазове (рекурсивне) обчислення функції $z_{n+1} = f(z_n)$, де z – комплексне число, а f – деяка функція. Розрахунок цієї функції триває, доки не будуть виконані певні умови. І коли ці умови будуть виконані, на екрані з’явиться крапка.

Береться деяка початкова точка z_0 на комплексній площині. Далі утворюють неперервну послідовність чисел на комплексній площині, кожне наступне із яких виходить із попереднього: z_0, $z_1 = f(z_0)$, $z_2 = f(z_1)$,…, $z_{n+1} = f(z_n)$ [27].

В той же час функції в різних точках комплексної площини можуть поводитися по-різному і люба точка z комплексної площини має свій характер поведінки при ітераціях функції $f(z_n)$, вона може прямувати до нескінченності, сходитись до якоїсь кінцевої точки,

циклічно приймати ряд фіксованих значень, тому вся площина ділиться на частини. При цьому точки, які лежать на краю цих частин, володіють такою властивістю: при як завгодно малому зміщенні, характер їх поведінки різко міняється(такі точки називаються точками біфуркації). Виявляється, що множини точок, які мають один конкретний тип поведінки, а також множини біфуркаційних точок часто мають фрактальні властивості. Це і є множина Жюліа (Рис. 1.8) для функції $f(z_n)$.

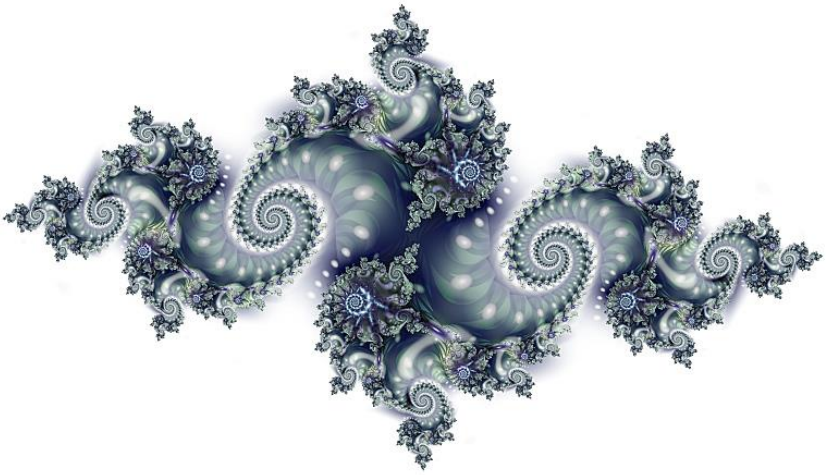


Рис. 1.8. Множина Жюліа [10]

Таким чином утворюється і множина Мандельброта – фрактал, визначений як множина точок С на комплексній площині, застосовані до функції $z_{n+1} = z_n + C$, при $z_1 = 0$. Ця модель стала класичною, представляючи собою класичний приклад самого фракталу, і часто використовується для демонстрації краси фракталів, залучення художників для фрактального живопису, дослідників, і просто зацікавлених людей.

1.3.2. Геометричні фрактали

Ці фрактали найнаочніші, тому що в них одразу видно самоподібність. У двовимірному випадку такі фрактали будуються поетапно, на основі вихідної фігури (лінії багатокутника чи багатогранника), шляхом її дріблення і виконання різноманітних перетворень отриманих фрагментів. На кожному наступному етапі, частини вже побудованої фігури, котрі аналогічні фрагменту який перетворили, знову заміняється на фрагмент, перетворивши його таким же чином. Всілякий раз масштаб зменшується. Коли зміни стають візуально незамітними, рахують, що побудована фігура добре приближена до фракталу і дає представлення того як вона виглядає. Щоб отримати сам фрактал потрібно нескінченне число ітерацій. Міняючи основу і фрагмент, можна отримати багато різних геометричних фракталів [27].

Геометричні фрактали хороші тим, що з одної сторони є предметом достатньо серйозного наукового дослідженя, а з іншої сторони, їх можна “побачити” – навіть людина,

далека від математики, знайде в них щось для себе. Таке поєднання рідкісне в сучасній математиці, де всі об’єкти задаються за допомогою незрозумілих слів і символів. Виявляється, багато геометричних фракталів можна намалювати буквально на папері в клітинку. Відразу примітка, всі одержувані зображення (зокрема й ті, що наведені тут) є лише кінцевими наближеннями нескінченних за своєю суттю фракталів. Але завжди можна намалювати таке наближення, що око не буде розрізняти зовсім дрібні деталі і наша уява зможе створити вірну картину фракталу. Наприклад, маючи досить великий аркуш паперу і запас вільного часу, можна вручну намалювати таке точне наближення до килима Серпінського, що з відстані в кілька метрів неозброєне око сприйматиме його як справжній фрактал. Комп’ютер дасть змогу заощадити час і папір і при цьому ще збільшить точність побудови.

1.3.3. Стохастичні фрактали та стохастичний аналіз

Іншим відомим класом фракталів є стохастичні фрактали. Вони створюються, коли один із параметрів випадковим чином змінюється в ітераційному процесі. При цьому їх найбільше використовують для утворення об’єктів, які дуже схожі на природні, такі як асиметричні дерева, порізані берегові лінії і т.д. Двовимірні стохастичні фрактали дуже часто використовуються для моделювання рельєфу різних природних об’єктів: рельєф місцевості, поверхня моря, ландшафту, поверхні гори, тощо [27].

Стохастичні фрактали – це клас фракталів, які поєднують в собі стохастичні (випадкові) елементи та фрактальну структуру. Вони використовуються для моделювання складних систем, які включають нерегулярність та непередбачуваність. Один з найвідоміших прикладів стохастичного фрактала – це “розбитий скляний” фрактал або фрактальний брістол. Цей фрактал відображає розбите скло, де кожен подрібнений шматочок також має складну розбиту структуру, схожу на початковий фрактал. Стохастичні фрактали також використовуються в фінансових ринках для моделювання цінових коливань та прогнозування ризиків. Наприклад, модель Геометричного Броунівського Руху (Geometric Brownian Motion) використовується для моделювання цін акцій та фінансових інструментів, де ціна змінюється стохастично з часом.

Природні об’єкти часто мають фрактальну форму. Для їх моделювання можуть застосовуватися стохастичні (випадкові) фрактали.

Стохастичний аналіз відноситься до галузі математики та статистики, яка вивчає випадкові процеси і стохастичні моделі. Він використовується для аналізу та прогнозування невизначеності та випадковості у різних системах та явищах. Стохастичний аналіз має широкі застосування в різних галузях, включаючи фінанси, економіку, фізику, інженерію, біологію та інші науки. Деякі конкретні приклади застосування стохастичного аналізу включають:

- Фінансовий аналіз: Стохастичні моделі використовуються для моделювання цінових рухів на фінансових ринках, прогнозування ризиків та оцінки портфелів.
- Теорія черг: Стохастичні процеси використовуються для аналізу пропускної здатності систем масового обслуговування, часів очікування та оптимізації розподілу ресурсів.
- Моделювання випадкових явищ: Стохастичні моделі дозволяють вивчати та прогнозувати випадкові явища, такі як погода, кліматичні зміни та поширення хвороб.
- Теорія керування: Стохастичні процеси використовуються для аналізу та оптимізації систем керування та прийняття рішень у невизначених умовах.
- Молекулярна динаміка: Стохастичні методи використовуються для моделювання руху та взаємодії молекул у хімічних та біологічних системах.

1.4. Фрактальна розмірність

Досі використовувались поняття “розмірності” у двох значеннях [18]:

- Три виміри евклідового простору (D=1;2;3). Точка як об’єкт розмірності 0, лінія як об’єкт розмірності 1, площину розмірності 2, фігури зі стереометрії розмірності 3.
- Кількість змінних у динамічній системі.

Фрактали, які є нерегулярними геометричними об’єктами, вимагають третього значення – фрактальна розмірність.

Фрактальна розмірність – це поняття фрактальної геометрії та показник, який вимірює складність або ступінь деталізації геометричної структури фрактала. Це величина яка показує ступінь заповнення фракталу по мірі його масштабування. Цей показник може бути використаний для опису фракталів, які мають нецілі або нелінійні розмірності.

Мандельброт почав свій трактат з фрактальної геометрії з розгляду питання: “Якої довжини узбережжя Британії?” [3]. Берегова лінія нерівна, тому вимірювання за допомогою прямої лінійки, як на наступному малюнку, дає приблизну оцінку. Приблизна довжина, L, дорівнює довжині лінійки, s, помноженій на N – кількість таких лінійок, необхідних для покриття вимірюваного об’єкта. На наступному малюнку (Рис. 1.9) вимірюється частина берегової лінії двічі, лінійка праворуч вдвічі коротша за ту, що використовується ліворуч.

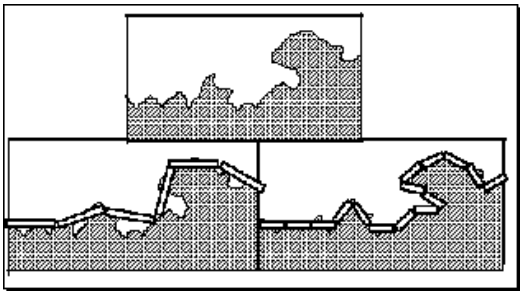


Рис. 1.9. Вимірювання довжини берега за допомогою лінійок різних величин [18]

Але оцінка праворуч довша. Якщо масштаб ліворуч дорівнює одиниці, виходить шість одиниць лінійок, але зменшення масштабу вдвічі дає 15 лінійок ($L = 7.5$), а не 12 ($L = 6$). Якщо знову зменшити шкалу вдвічі, то вийде подібний результат, довша оцінка L. Загалом, коли лінійка стає дедалі меншою, довжина стає нескінченно великою. Поняття довжини починає втрачати сенс [18].

Льюїс Фрай Річардсон вперше помітив закономірність між довжиною національних кордонів і розміром масштабу. Як показано далі, залежність між оцінкою довжини та масштабом є лінійною на графіку "логарифм-логарифм" (Ефект Річардсона) (Рис. 1.10).

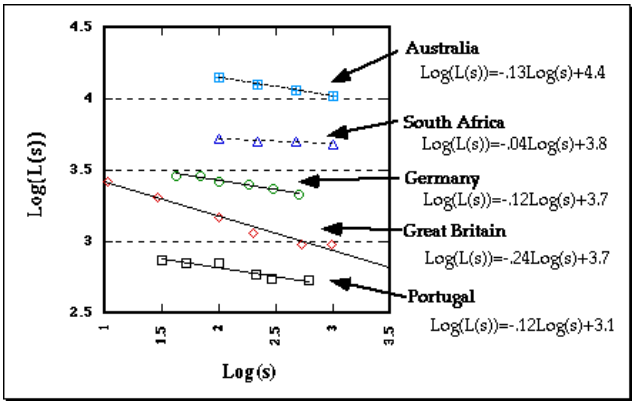


Рис. 1.10. Ефект Річардсона. Лінійна регресія [18]

Мандельброт присвоїв умову $(1 - D)$ для нахилу, тому функції мають вигляд: $\log\big(L(s)\big) = (1 - D)\log(s) + b$, де D – фрактальна розмірність. Для Великобританії $1 - D = -0.24$, приблизно. D = 1-(-0.24) = 1.24. Берегова лінія Південної Африки дуже гладка, практично дуга кола. Нахил, розрахований вище, дуже близький до нуля. $D = 1 - 0 = 1$. Це має сенс, тому що берегова лінія є дуже близькою до правильного евклідового об'єкту, лінія, яка має розмірність одиниці. Взагалі, чим "грубіша" лінія, чим крутіший схил, тим більша фрактальна розмірність [18].

Щоб визначити фрактальну розмірність строго самоподібної множини, нехай N – це кількість менших частин фракталу які є зменшеними копіями всієї фігури, а r – коефіцієнт масштабування, а n число ітерацій. Тоді D (розмірність) визначається рівнянням:

$$N = r^D; D = \lim_{n\to\infty} \frac{\log N_n}{\log r_n}; D = \log_r N$$

Це є співвідношення логарифму від числа нових елементів на ітерації n поділених на коефіцієнт масштабування(нових елементів до старих). Логарифм може мати буть-яку основу, тому можна спростити до виразу $D = \log_r N$ [17].

Загалом це розмірність за означенням німецького математика Гаусдорфа. Якщо взяти об'єкт евклідової розмірності D і зменшити його лінійні розміри на 1/r у кожному

просторовому напрямку, його міра (довжина, площа або об'єм) збільшиться в $N = r^D$ разів порівняно з початковою. Це зображено на Рис. 1.11.

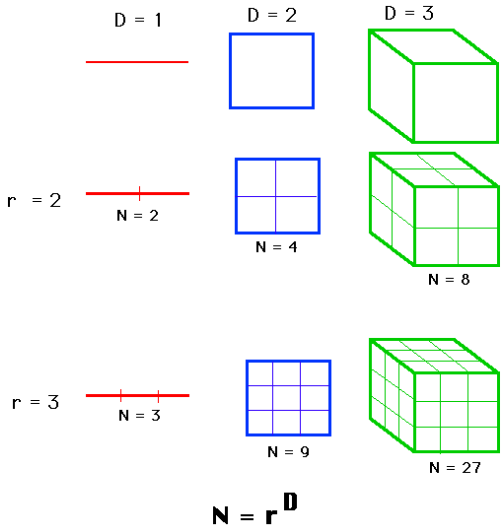


Рис. 1.11. Зміна міри, залежно від довжини, в різних розмірностях [18]

Розглянемо формулу $N = r^D$, візьмемо логарифми з обох сторін цього рівняння, після, D можна винести зі степеня у множник за правилами логарифма, і звідси випливає формула для розрахунку розмірності (D).

Фрактальна розмірність для 2D об'єктів буде $D \in [1; 2]$ і для 3D об'єктів буде $D \in [2; 3]$. Чим ближче розмірність до одиниці, тим більше об'єкт фракталу подібний на пряму лінію, чим ближче до двох, тим більше схожий на площину.

Наприклад, розмірність сніжинки Кох має топологічну розмірність одиницю, але вона не є кривою в жодному разі: довжина кривої між двома точками сніжинки Кох є нескінченною. Жоден найменший шматок цієї кривої не є подібним до лінії, але не є він чимось подібним до шматочку площини тощо. Можна сказати, що цей шматочок є занадто "товстим" щоб класифікувати його як одновимірний об'єкт, але він занадто "тонкий" щоб класифікувати його як двовимірний об'єкт. Тобто розмірність цього об'єкта є числом між одиницею і двійкою.

Розглянемо розмірність прямої Кантора:

Таблиця 1. Розмірність пороху Кантора

Ітерації(n)	Кількість ліній(N)	Коефіцієнт масштабування ліній(r)
0	1	1
1	2	3
2	4	9
3	8	27
…	…	…
n	2^n	3^n

На нульовій ітерації постає питання з чого починається фігура – це називається ініціатор, а на першій ітерації постає питання як його змінювати – генератор. І далі ця ітерація застосовується рекурсивно.

Це означає для фрактальної розмірності прямої Кантора:

$$D = \lim_{n\to\infty} \frac{\log N_n}{\log r_n} = \lim_{n\to\infty} \frac{\log 2^n}{\log 3^n} = \lim_{n\to\infty} \frac{n\log 2}{n\log 3} = \frac{\log 2}{\log 3} = \log_3 2 \approx 0.63\,;$$

Це означає, що пряма лінія Кантора – це навіть не лінія, це десь між точкою та лінією. Тож це дає деяке уявлення про те, що таке множина Кантора [20].

Наприклад, трикутник Серпінського є строго самоподібним. Вихідний трикутник замінюється 3 подібними трикутниками, так що N = 3. Кожна сторона на 1/2 більша за сторону вихідного трикутника, тобто кожна фігура виглядає точно так само, як оригінальна фігура, збільшена у 2 рази (масштабний коефіцієнт), тобто r = 2. Звідси випливає:

$$3 = 2^D;\ D = \frac{\log 3}{\log 2} = \log_2 3 \approx\ 1.584$$

Основні методи визначення розмірності фрактала: розмірність Хаусдорфа; розмірність Мандельброта; розмірність Бокса; кореляційна розмірність; розмірність Мінковського.

На практиці, розмірність Мінковського і кореляційна розмірність широко застосовуються через їхню простоту використання. Хоч для деяких фракталів всі ці розмірності збігаються, загалом вони не є еквівалентними.

Для вимірів Хаусдорфа та інших подібно означуваних вимірів часто використовується спільний термін “фрактальний вимір”. Він істотно відрізняється від топологічної розмірності, яка приймає лише цілі значення.

Таким чином, можна виміряти фрактальну розмірність для фракталів, які визначені формально. Але для реальних об’єктів, які володіють властивістю фракталів, точну фрактальну розмірність порахувати не вийде, але її можна оцінити, і залежно від кількості чи масштабу даних, розмірність може відрізнятись.

Для визначення фрактальної розмірності площин використовуються різні методи, одним з широко використовуваних методів є метод бокс-лічильника (Box-Counting), який дозволяє оцінити фрактальну розмірність об'єктів на основі поділу їх на квадратні блоки. Цей метод по суті використовує ту ж лінійну регресію, тільки вже з площинам замість прямих.

Фракталом називають множину, фрактальний вимір якої відрізняється від топологічної розмірності [5].

1.5. Фрактали в науці та ІТ

Стохастичний аналіз відомий величезними застосуваннями у фінансах. Стохастичний аналіз можна використовувати для опису цін фінансових інструментів, акції чи опціонів, появилась формула Блека-Шоулза, яка описувала як рахувати ціну опціону, в результаті чого стався бум ринку опціонів в 70-х і 80-х роках, з'явились нові трільйонні ринки, люди заробили мільярди на ММ, до сих пір кванти в фінансах, які застосовують стохастичний аналіз вважаються ключовими у багатьох банках і фондах.

Досі антени мали прості форми, які описуються в евклідовій геометрії. В останній десяток років багато вчених по всьому світу намагалися зробити структуру фрактальної геометрії придатною для застосування в області електромагнетизму, що призвело до розробки нових інноваційних конструкцій антен і штучних діелектричних/магнітних матеріалів.

Вперше фрактали використали для створення ефективних антен. В 90 роки, американський інженер Натан Коел взяв звичайний мідний дріт і дав їй форму самоподібної ламаної кривої, після підключив до свого радіоприймача і виявив, наскільки добре вона працює. Продовжуючи розвивати цей напрямок в побудові антен з'ясувалося, що за допомогою фракталів можна суттєво зменшити розміри конструкцій, і розширити смугу робочих частот, тобто можна створити широкосмугову антену. Надалі математичні розрахунки показали, що справді широкосмугова антена, діапазон якої може охопити весь радіохвильовий спектр повинна мати фрактальну форму. Самоподібність корисна при проектуванні багаточастотних антен, як, наприклад, пристрої на основі фрактального трикутника або килима Серпінського, фрактальної кривої Мінковського, які були використані при проектуванні багатодіапазонної антени [15]. С тих пір теорія фрактальних антен продовжує інтенсивно розвиватися. Перевагою таких антен є багатодіапазонність і порівняльна широкополосність. Коен заснував власну компанію і налагодив серійний випуск своїх антен [13].

Фрактали – це абстрактні об'єкти, такі, що задовольняють строгому розумінню математичного опису, не можуть бути фізично реалізовані, оскільки є нескінченними. Однак можна зробити деякі припущення щодо ідеального фракталу, які дозволяють конструювати електромагнітні прилади. Зазвичай такі фігури називають префракталами або урізаними фракталами. Їх також застосовують в інших задачах, які віикористовують їх концепцію. Можна застосовувати різні геометрії, наприклад, конфігурації з декількох трикутників або інші складні конструкції для побудови антен, які можуть бути схожими на фрактальні форми і отримувати деякі з їхніх переваг. Загалом, фрактальна антенна технологія – це термін, який використовується для опису цих антенних інженерних методів, які базуються на таких

математичних концепціях, що забезпечують створення нових антен з деякими характеристиками, які були неможливими навіть у середині 1980-х років.

Нещодавно криві, що самозаповнюються в просторі, такі як фрактали Гільберта і Пеано, були використані для отримання високоефективних, низькопрофільних, конформних антен, з поліпшеними характеристиками випромінювання і підвищеним коефіцієнтом підсилення потужності для різних комунікаційних і радіолокаційних застосувань.

У багатьох електромагнітних пристроях самоподібність і площинне заповнення фрактальних геометрій часто якісно пов’язані з їхніми частотними характеристиками, тобто багаточастотною роботою або малими розмірами в низькочастотних діапазонах. Для роботи багатодіапазонної антени використовується самоподібна конструкція. Структура фрактала, що самозаповнює простір, використовується для проектування малих антен. Масові фрактали та граничні фрактали використовуються для проектування антенних решіток.

Фрактали досить часто використовуються у віртуальному світі, якщо потрібно побудувати величезний віртуальний світ, визначати кожну окрему точку це дуже складна задача, для спрощення задачі потрібно розробити генератор рельєфу, і для цього використовується концепція фракталів. Ландшафти мають як правило властивість приближеної самоподібності. Також таке можна замітити з іншими об’єктами природи, наприклад, як вже згадувалось, хмари, мох, брокоолі, сніжинка і т.д. Тобто всі об’єкти природи, які нагадують фрактали, можна моделювати в віртуальному світі за допомогою генераторів із використанням концепції фракталів.

Концепція фракталів знайшла широке застосування в інформаційних технологіях [7]. Деякі з основних способів використання фракталів у ІТ:

- Графіка та візуалізація: Фрактальні зображення використовуються для створення складних та деталізованих візуальних ефектів. Вони знаходять застосування у комп’ютерних графіках, генерації ландшафтів, дизайні та віртуальній реальності.
- Компресія даних: Фрактальна компресія даних є методом стиснення, який використовує самоподібність фракталів для ефективного зберігання та передачі інформації. Цей метод дозволяє зменшити розмір файлів, зберігаючи високу деталізацію.
- Генерація реалістичних об’єктів: Фрактальні алгоритми використовуються для створення реалістичних об’єктів, таких як хмари, дерева, гірські хребти, ландшафти тощо. Вони дозволяють створювати складні й непередбачувані форми, що додають реалізм до віртуальних середовищ.

- Криптографія: Фрактальні функції можуть бути використані у криптографії для створення криптографічних хеш-функцій та генерації випадкових чисел. Вони володіють властивостями, що забезпечують стійкість та непередбачуваність.

Замість традиційних методів стиснення, які базуються на видаленні деякої інформації зображення, фрактальне стиснення використовує самоподібність фрактальних структур для збереження інформації. Процес фрактального стиснення зображень складається з таких кроків:

- Декомпозиція: Зображення розбивається на невеликі блоки або пікселі.
- Кодування: Кожен блок або піксель зображення аналізується для виявлення самосхожої структури. Це означає пошук подібних частин зображення на різних масштабах.
- Збереження: Замість зберігання всіх пікселів зображення, зберігаються лише інструкції щодо генерації самосхожих структур. Це дозволяє зберегти значний обсяг даних.
- Відновлення: Зображення відновлюється шляхом використання збережених інструкцій та генерації самосхожих структур.

В результаті фрактального стиснення можна досягти значного зменшення обсягу даних зображення при збереженні високої якості. Однак, процес стиснення та відновлення може бути витратним з точки зору обчислювальних ресурсів.

Фрактали мають широке застосування також у науці [7, 14]:

- Фізика: Фрактальна геометрія використовується для дослідження нерегулярних структур, наприклад, в складних фізичних системах чи в природних явищах, таких як грозові хмари, гірські ланцюги, або фрактальні розбиття.
- Медицина: Фрактальний аналіз використовується для вивчення нерегулярностей у біологічних системах, наприклад, для аналізу серцевої діяльності, електроенцефалограми (ЕЕГ) та інших біомедичних сигналів.
- Екологія: Фрактальна геометрія застосовується для вивчення структури природних екосистем, таких як ліси або рифи, і для оцінки їх біологічного різноманіття.

1.6. Застосунки для дослідження фракталів

Загалом, конецепція фракталів використовується для генерації природних об’єктів і явищ, в програмах для 3D моделювання таких як 3DS MAX. Зазвичай в таких програмах є вже вбудовані рішення для створення природних об’єктів, наприклад ландшафту, води і т.д. В цій роботі буде розглянуто застосунки де можна їх окремо дослідити і детально познайомитись.

Існують вони як для ПК, так і для телефонів, так і навіть такі які можна використовувати безпосередньо в браузері [8].

На ПК, було виділено такі утиліти:

- XaoS Software – комплексні фрактали, можна використовувати для фрактального живопису.
- Terragen 4 – малювання краєвидів за допомогою фракталів (Рис. 1.12) [16].

Слід зазначити що Terragen має платну і безкоштовну(тобто спрощену) версію. Слід зазначити, що інтерфейс програми, спочатку є інтуїтивно незрозумілим. Робоче вікно цього застосунку виглядає наступним чином (Рис. 1.12):

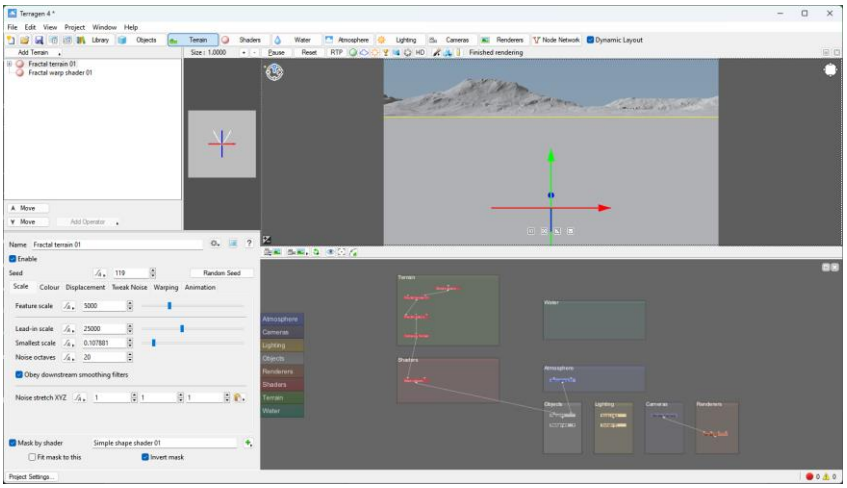


Рис. 1.12.Terragen 4. Робоче вікно [16]

Ліворуч і зверху є елементи керування, ліворуч зверху є список інструментів. Цей застосунок є потужний, оскільки можна керувати всіма складовими краєвидів: ландшафт, тінь, вода, атмосфера, освітлення, хмари тощо.

Також є головне вікно з ландшафтом, де можна використовувати клавішу миші щоб перетягувати навколо і нахиляти. Все що не було зображено, не було згенеровано, і там буде пустота, тільки якщо відпустити клавішу миші, він згенерує на новому місці краєвид. За допомогою правою кнопкою миші можна панорамувати. У лівому верхньому куті є іконка яка відповідає за рух.

Загалом, кожен параметр можна тонко налаштувати, наприклад хмари можна вибрати різні і одразу декілька, їхню наповненість, густоту, рівень над морем можна змінювати. Сонце можна повністю регулювати його місцезнаходження навколо землі, все це взаємодіє між собою, наприклад якщо опустити сонце, то буде захід сонця як в реальному житті.

Можна побачити як процес генерації відбувається за допомогою рекурсивної генерації (Рис. 1.13). В кінці бажано застосувати ‘RTP’, для кращого промальовування.

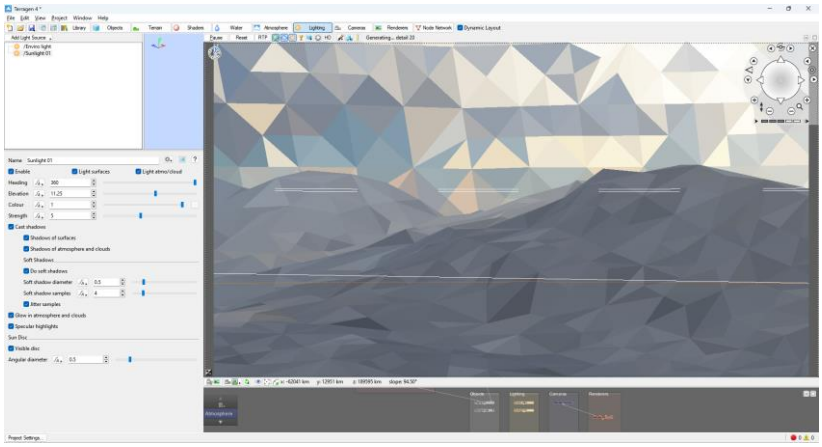


Рис. 1.13. Процес генерації [16]

Програма чудово підходить для того щоб створювати красиві і реалістичні, картини, робити фрактальні живописи. Хоча в застосунку є деякі баги по типу просвічування гори який було виявлено в процесі ознайомлення із застосунком (Рис. 1.14), і при використовуванні паралелізму вона все ще довго працює, але із подібних ПЗ це найкраший вибір. В ній можна створювати всі можливі краєвиди (Рис. 1.15).

Рис. 1.14. Баг із просвічуванням гори

Рис. 1.15. Краєвид згенерований за допомогою використання концепції фракталів

На телефоні IOS, є програма, яка сильно вирізняється між іншими, це програма FRAX. В цій програмі можна дослідити фрактал під назвою ‘множина Мандельброта’ та різні його підвиди. Для нього можна задати розмальовку, керувати текстурою, та напрямом освітлення. Цей застосунок добре підходить для фрактального живопису.

Одна з веб-утиліт для дослідження фракталів зображена на Рис. 1.16 [9]:

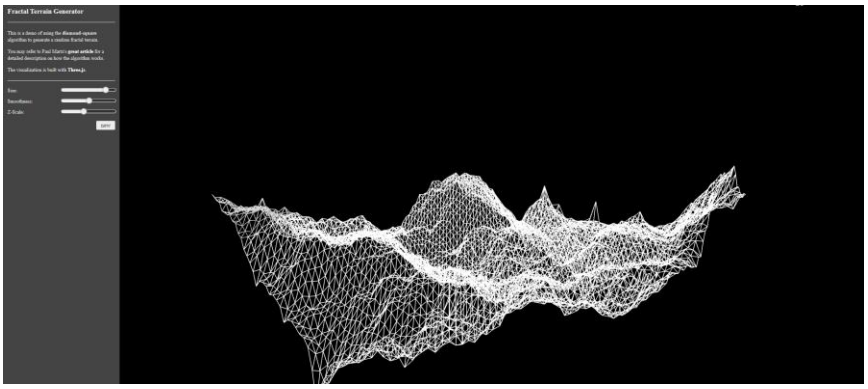


Рис. 1.16. Fractal Terrain Generator [9]

Взагалом ця утиліта дуже не дороблена, в неї є декілька мінусів. По-перше ландшафт крутиться статично, і це не можна поміняти, не можна рухати чи наближати, не можна змінювати положення ландшафту, на можна зупинити анімацію. Наступне те, що при руханні повзунків, дуже часто починає погано пристосовувати зміни, видані в повзунках, наприклад на Рис. 1.17, просто змінено повзунок ‘smoothness’ і веб-застосунок видає вже плоску площину.

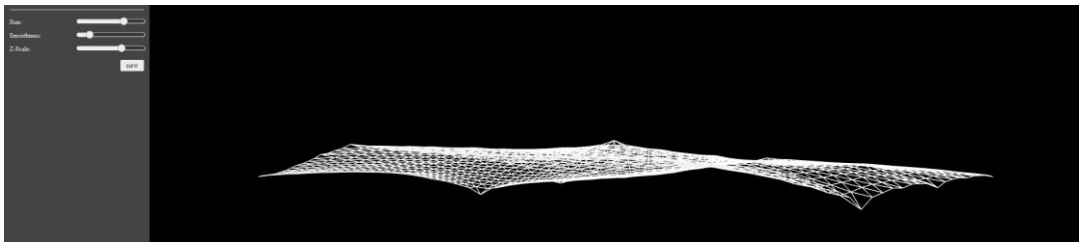


Рис. 1.17. Баг з повзунком ‘smoothness’ [9]

Це через те, що повзунок ‘smoothness’ не працює, просто через кожні 0.1 значення вигляд ландшафту змінюється з об’ємного на плоский чи навпаки.

В цій роботі буде розроблятись ПЗ, подібне цьому застосунку, де можна буде також повертати об’єкт і наближувати.

РОЗДІЛ 2. ЗАСОБИ РОЗРОБКИ ТА АЛГОРИТМИ

2.1. Засоби розробки

При реалізації застосунку буде використовуватись мова програмування Python, в середовищі для розробки PyCharm, також буде використовуватись система контролю версій GIT, за допомогою якої будемо зберігати стан речей та фіксувати всі зміни в мережу GITHUB.

2.1.1. Мова програмування Python

Python – одна з мов програмування, яка одночасно претендує на назву проста і потужна. Python є однією з найпопулярніших мов програмування на сьогоднішній день і використовується в різних сферах, від наукових досліджень до веб-розробки та інтеграції систем. Використовуючи цю мову можна легко зосередитись на рішенні поставленої задачі, а не на синтаксисі і структури мови певної мови програмування. Особливості мови Python є:

- Простий і мінімалістичний – має простий синтаксис, що полегшує розробку програм та їх читання.
- Вільний і відкритий – open source software.
- Мова програмування високого рівня – мова з динамічною типізацією та автоматичним керуванням пам’яттю, орієнтовано на підвищення продуктивності розробника читабельності коду та його якості.
- Портований – python був портований на багато платформ.
- Інтерпретований – python сам перетворює вхідний код в машинну мову.
- Об’єктно-орієнтований – python підтримує як процедурний кодинг, так і ООП, Python надає прості але потужні засоби для ООП.
- Розширюваний – певні куски коду за бажанням можна написати мовою C/C++, а потім викликати її з програми Python.
- Вбудований – його можна вбудовувати в програми на C/C++.
- Великий вибір бібліотек з готовими рішеннями.

2.1.2. Система контролю версій GIT

GIT – це безкоштовна розподілена система контролю версій(CVS) з відкритим вихідним кодом, для керування змінами в кодовій базі проекту, призначена для швидкої та ефективної роботи з будь-якими проектами, від невеликих до дуже великих. Git простий у вивченні, займає мінімум місця та має блискавичну продуктивність.

За допомогою GIT можна легко керувати версіями коду, вертатись до попередніх станів, тобто відновлювати, якщо код в новій версії працює некоректно, порівнювати зміни, створювати гілки для паралельної зміни одного коду і т.д. У файлу може бути один із 4-ох станів (Рис. 2.1)

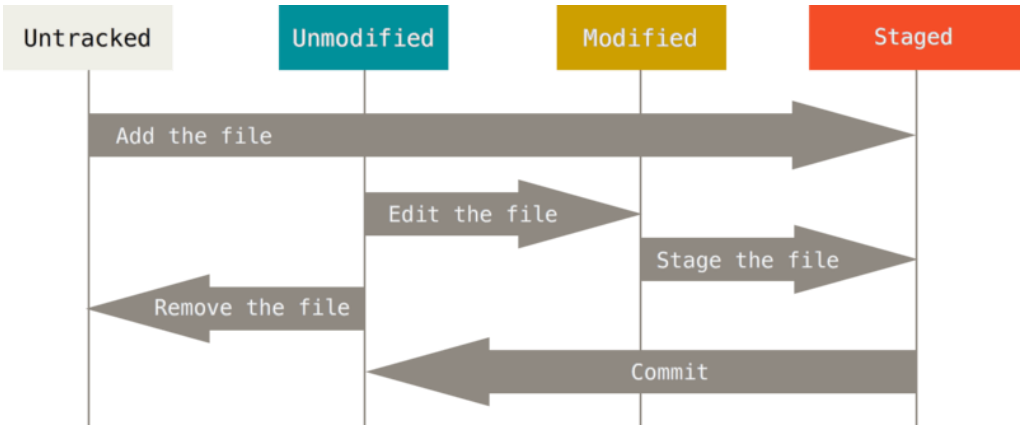


Рис. 2.1. Git стани

Зміни в файлі фіксуються за допомогю коммітів, потім їх можна загружати в хмару на репозиторій в GitHub чи GitLab, що дозволяє зберігати проект в хмарі та мати доступ до нього з будь-якого місця. Тому, використання Git у проєкті допоможе зберегти контроль над версіями коду, ефективно вести розробку та забезпечити коректну роботу коду.

2.1.3. Середовище розробки PyCharm

JetBrains PyCharm – інтегроване середовище розробки (IDE) для мови програмування Python, від компанії JetBrains. Воно надає розширені можливості для розробки, налагодження і керування проектами на Python. Основні функції та особливості:

- Це потужний редактор коду з підсвічуванням синтаксису, автодоповненням, перевіркою помилок, рефакторингом та багатьма іншими функціями, які і полегшують написання коду.
- Має вбудований debugger, за допомогою якого можна ставити точки зупину, крокувати по коду, переглядати значення змінних в конкретний момент виконання коду та інші дії для аналізу та виправлення помилок.
- Керування версіями: PyCharm інтегровано з системою керування версіями Git і надає зручні інструменти для керування репозиторіями, злиття гілок, відстежування змін тощо.

2.1.4. Бібліотеки

Для розробки застосунку будуть застосовані такі бібліотеки:

- Matplotlib – для візуалізації нашого фрактального ландшафту.
- Numpy – для створення матриці КВ та матриць з координатами для передачі в візуалізатор.
- Random – для додавання стохастичного(випадкового) елементу, основної складової стохастичного фракталу, в нашу КВ. Буде використовуватись для випадкового змінення одного з параметрів в ітераційному процесі.
- Copy – для створення повної копії згенерованої КВ.
- Tkinter – для створення GUI і зручної взаємодії з фрактальним ландшафтом.
- Time – для фіксації різниці часу роботи між різними методами реалізації.
- Numpy-stl – для збереження ландшафту у форматі stl.
- PyInstaller – інструментом для компіляції Python-програм у самодостатні виконувані файли, які можна запускати без встановлення самого Python.

2.2. Алгоритми генерації ландшафтів

Мета цього розділу полягає не стільки в тому, щоб забезпечити математично повний і строгий облік методів, що використовуються для генерації фрактальних ландшафтів, скільки в тому, щоб забезпечити огляд цих методів і їх застосування, доступних для когось з математикою вищого шкільного рівня. Ті, хто має більш повне розуміння цієї сфери, помітять, що є деякі складності.

Корисним застосуванням фракталів є генерація випадкової місцевості для комп’ютерних ігор та віртуальних світів. Такі форми рельєфу, як гори та долини, мають властивість фракталів до самоподібності. Наприклад, якщо збільшити масштаб вершини гори, вона нагадує форму всієї гори. При збільшенні масштабу загальна форма гори повторюється в скелях, камінні та осипах. Один з методів, що використовуються для генерації фрактальних ландшафтів, є алгоритм зміщення середньої точки, інші методи будуть його наслідниками [12].

Можна помітити що віртуальні світи комп’ютерних ігор зараз досить великі. Є такі ігри, де можна блукати ландшафтом розміром з країну. Тепер це неможливо для дизайнерів, щоб визначати кожну вершину, кожен багатокутник, який утворює цей ландшафт. Тому створюються ландшафтні генератори, і це використовує принципи фракталів та їх властивість, тобто самоподібність.

В цій роботі розглядатимуться три найпоширеніших методи, які можна використовувати для створення ландшафтів.

2.2.1. Зміщення середньої точки

Перший – найпростіший алгоритм – зміщення середньої точки (Рис. 2.3). Це розроблено для того, щоб малювати щось на зразок 2D. Все починається з лінії, тому в цій роботі це називатиметься 1D ландшафтом, так як лінія має розмірність одиницю, але в кінцевому підсумку виходить плоска гора яка намальована на площині 2D.

1. В алгоритмі переміщення середньої точки, спочатку є лінія, обчислюється середина лінії, тому лінія має початок і кінець, обчислюються координати середньої точки. Отже, наприклад для координати x, якщо середина дорівнює індексу i+1/2, тоді координати середньої точки будуть:

$$x_{i+\frac{1}{2}} = \frac{1}{2}(x_i + x_{i+1})$$

2. Обчислюється середина, далі переміщується вертикально координата на невелику випадкову величину. Вона випадкова але не довільна, вона має існувати в цьому діапазоні, тож є деяке мінімальне значення діапазону та деяке максимальне значення. Тож зміщується точка трохи вверх чи вниз, скажімо що r – це випадкова величина для якої $r \in (r_{min}; r_{max})$, яка додається:

$$x_{i+\frac{1}{2}} = x_{i+\frac{1}{2}} + r$$

3. Після цього вийдуть дві лінії, замість одної. Потім зменшується діапазон значень, яких може набувати випадкова величина r, зміщення, тому виконується:

$$r_{min} = \frac{1}{2} r_{max}; \quad r_{max} = \frac{1}{2} r_{max}$$

4. Далі повторюються кроки 1 до 3.

Тому, якщо продовжити це робити, зрештою вийде те, що має бути впізнаваним як гора або ж долина (Рис. 2.2).

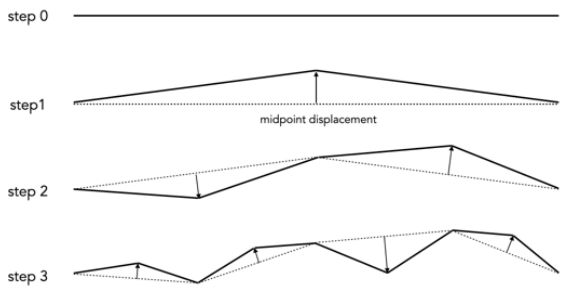


Рис. 2.2. Процес застосування алгоритму зміщення середньої точки

Рис. 2.3. Ландшафт згенерований за допомогою алгоритму зміщення середньої точки [22]

2.2.2. Параметр жорсткості H

Величина зміщення вибирається з діапазону, який зменшується з однієї ітерації до наступної. Очевидний спосіб зменшити діапазон – це зменшувати його вдвічі на кожній ітерації. Якщо r – випадкове зміщення на ітерації n, тоді:

$r_1 = [-1; 1], r_2 = [-0.5; 0.5], r_3 = [-0.25; 0.25]$ і так далі.

Зміщення буде залежати від того наскільки гладкий чи горбастий ландшафт потрібно отримати. Якщо потрібна нерівна гірська місцевість, треба щоб вона було досить зубчастою, діапазон для параметру зміщення задається досить великим, якщо потрібен рівний ландшафт, скажімо пагорби чи якісь низинні ділянки, потрібно щоб параметр зміщення мав менший діапазон [21].

Для того, щоб мати певний контроль над жорсткістю рельєфу, вводиться параметр жорсткості H, який визначає швидкість зменшення для діапазону випадкових чисел. Нехай діапазон позначає максимальне абсолютне значення в діапазоні випадкових чисел для ітерації n, тоді:

$$range = 2^{-2nH}$$

H набуває значень в [0;1]. Наприклад, якщо H = 0.5, то $range = 2^{-n}$ і діапазон зменшується вдвічі на кожній ітерації. Якщо H = 0 то $range = 1$ і діапазон не зменшується, тому завжди виконується вибірка між тим самим діапазоном, що призводить до нерівностей, а якщо H = 1, то $range = 2^{-2n}$ і діапазон зменшується в чверть на кожній ітерації, що призводить до згладжування рельєфу. Для алгоритму зміщення середньої точки фрактальна розмірність пов'язана з параметром жорсткості H співвідношенням, з якою можна оцінити її розмірність:

$$D = 2 - H$$

Ландшафти згенеровані використовуючи три різні величини H показані на Рис. 2.4.

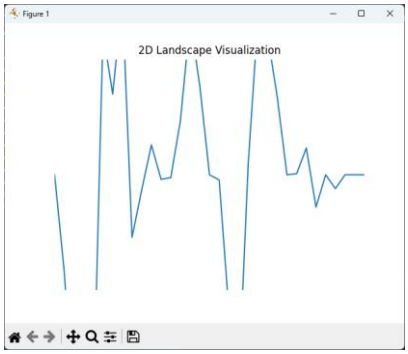


Рис. 2.4. а) H=0.2

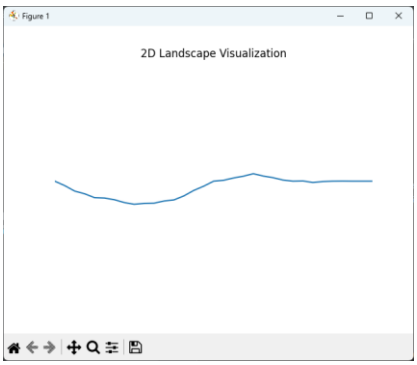


Рис. 2.4. б) H=0.8

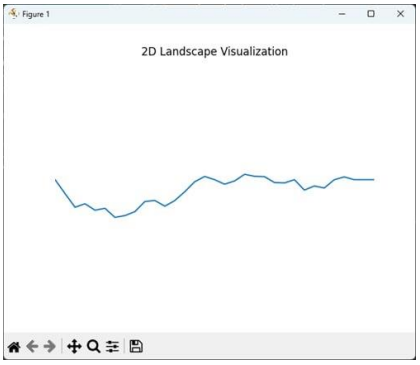


Рис. 2.4. в) H=0.5.

Це згенеровано за допомогою 5 ітерацій, а складність коду O($size * \log size$).

2.2.3. 2D зміщення середньої точки

Це був 1D метод, що створює гори, у віртуальних світах використовується 3D, тому створюється поверхня для ніг. 2D називається так, тому що починається з плоскої площини, і з неї створюється ландшафт у 3D. Алгоритм подібний до представленого для одновимірного випадку, починаючи з ітерації 0 з горизонтального квадрата, побудова фрактального ландшафту відбувається наступним чином [21]:

1. Обчислюються середні точки по краях квадрата (сірі кружечки) (Рис. 2.5 а) та зміщується вертикальна координата:

$$z_{i+\frac{1}{2},j} = \frac{1}{2}\left(z_{i,j} + z_{i+1,j}\right) + r,$$

$$z_{i+1,j+\frac{1}{2}} = \frac{1}{2}\left(z_{i+1,j} + z_{i+1,j+1}\right) + r,$$

$$z_{i+\frac{1}{2},j+1} = \frac{1}{2}\left(z_{i,j+1} + z_{i+1,j+1}\right) + r,$$

$$z_{i,j+\frac{1}{2}} = \frac{1}{2}\left(z_{i,j} + z_{i,j+1}\right) + r.$$

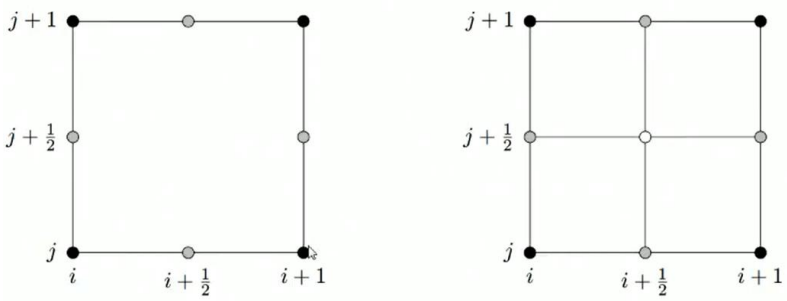
2. Обчислюється середня точка в центрі квадрата (біле кружечко) (Рис. 2.5 б), і зміщується вертикальна координата:

$$z_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{4}\left(z_{i,j} + z_{i+1,j} + z_{i,j+1} + z_{i+1,j+1}\right) + r$$

3. Зменшується діапазон значень, яких може набувати випадкова величина r, тобто зміщення:

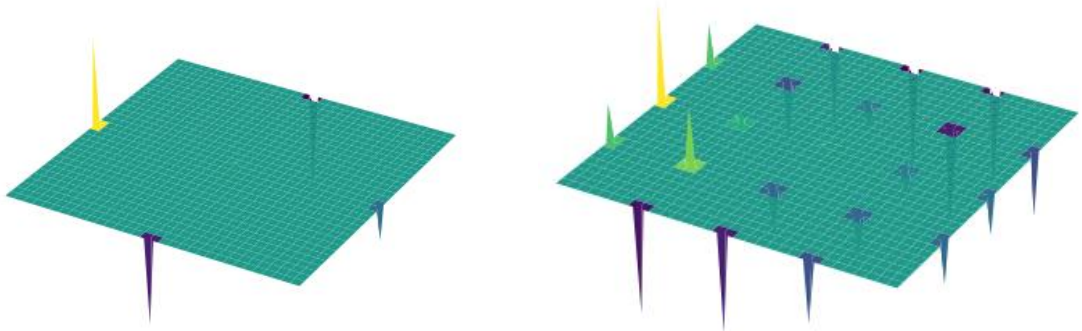
$$r_{min} = \frac{1}{2}r_{max}; \quad r_{max} = \frac{1}{2}r_{max}$$

4. Повторюється всі кроки (Рис. 2.6).

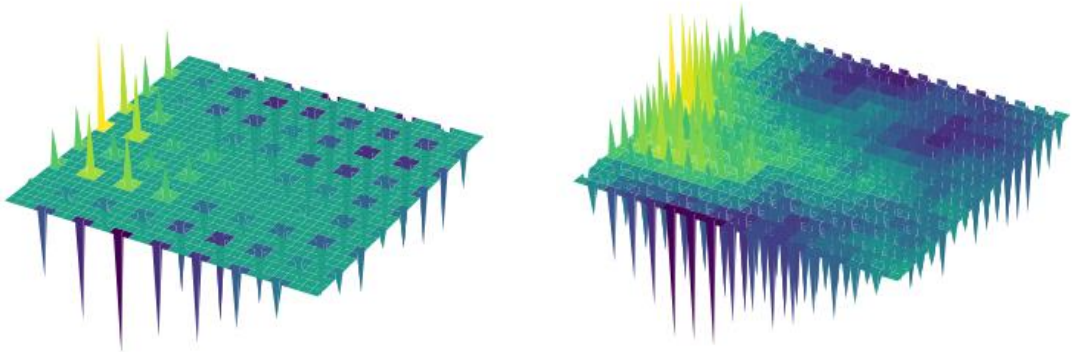


а) 1-ий крок б) 2-ий крок

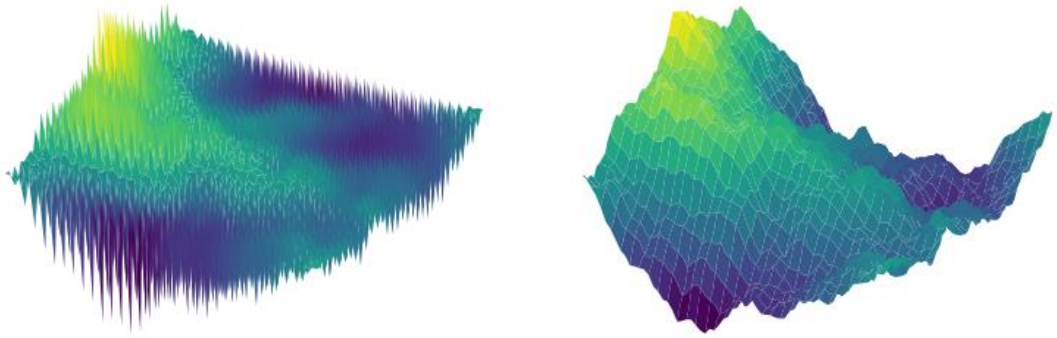
Рис. 2.5. Графічне відображення алгоритму 2D змщення середньої точки [21]



а) 1-ша ітерація б) 2-га ітерація



в) 3-тя ітерація г) 4-та ітерація



д) 5-та ітерація е) 6-та ітерація

Рис. 2.6. Процес генерації ландшафту з візуалізацією на кожному кроці

2.2.4. Алгоритм діамант-квадрат

Основним недоліком двовимірного алгоритму зміщення середньої точки є те, що хребти видно на кінцевому ландшафті. Це можна побачити ландшафтах, згенерованих цим методом. Ці хребти з'являються внаслідок зміщення середніх точок на ранніх ітераціях. Алгоритм діамант-квадрат запобігає утворенню гребнів на ландшафті, зміщуючи вертикальні координати в серединних точках вздовж країв квадрата. Починаючи з горизонтального квадрата на ітерації 0, алгоритм діамант-квадрат працює наступним чином [21, 23, 24]:

1. Діамантовий крок (Рис. 2.7 а) – обчислюються середні точки в центрі квадратів (сірі кружечки), використовуючи вершини (чорні кружечки).

$$z_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{4}\left(z_{i,j} + z_{i+1,j} + z_{i,j+1} + z_{i+1,j+1}\right) + r$$

2. Квадратний крок (Рис. 2.7 б) – обчислюються середні точки вздовж кутів квадратів(білі кружечки) використовуючи вершини і центральні середні точки квадратів, які є з обох боків, або з одного квадрату, і того, який знаходиться на протилежному кінці.

$$z_{i+\frac{1}{2},j} = \frac{1}{4}\left(z_{i,j} + z_{i+\frac{1}{2},j-\frac{1}{2}} + z_{i+1,j} + z_{i+\frac{1}{2},j+\frac{1}{2}}\right) + r$$

$$z_{i,j+\frac{1}{2}} = \frac{1}{4}\left(z_{i,j} + z_{i+\frac{1}{2},j+\frac{1}{2}} + z_{i,j+1} + z_{i-\frac{1}{2},j+\frac{1}{2}}\right) + r$$

Для вершин на лівому і нижньому ребрах, $z_{i+\frac{1}{2},1}$ і $z_{1,j+\frac{1}{2}}$, обчислюються:

$$z_{i+\frac{1}{2},1} = \frac{1}{4}\left(z_{i,1} + z_{i+\frac{1}{2},N_x-\frac{1}{2}} + z_{i+1,1} + z_{i+\frac{1}{2},j+\frac{1}{2}}\right) + r$$

$$z_{1,j+\frac{1}{2}} = \frac{1}{4}\left(z_{1,j} + z_{i+\frac{1}{2},j+\frac{1}{2}} + z_{1,j+1} + z_{N_y-\frac{1}{2},j+\frac{1}{2}}\right) + r$$

Вершина N це центр, який знаходиться в квадраті на протилежному кінці всієї матриці. Тобто, оскільки одна з точок буде виходити за межі матриці, вона береться з іншого кінця [9].

Для вершин на правому та верхньому ребрах, $z_{i+\frac{1}{2},N_y}$ і $z_{N_x,j+\frac{1}{2}}$, приймаються значення з лівого та нижнього країв, тобто [9, 21]:

$$z_{i+\frac{1}{2},N_y} = z_{i+\frac{1}{2},1}$$

$$z_{N_x,i+\frac{1}{2}} = z_{1,j+\frac{1}{2}}$$

3. Повторяються кроки 1 і 2 використовуючи нові обчислені середини ребер як вершини квадратів.

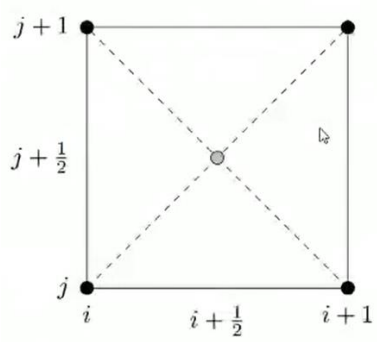


Рис. 2.7. а) Діамантовий крок DS [21]

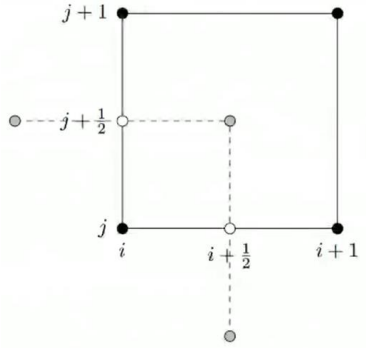


Рис. 2.7. б) Квадратний крок DS [21]

2.2.5. Удосконалення ландшафту

Вигляд ландшафту можна покращити за допомогою наступних дій:

- Додавання кольору – загальний колір ландшафту залежить від його вертикальної висоти. Можна додати воду, застосувавши кольорову карту, де частини з найнижчими вертикальними висотами замальовані голубим. Низько розташовані ділянки, як правило, мають пишну рослинність, яку можна змоделювати, зафарбувавши наступний шар зеленим кольором. Гори, де висота над рівнем моря надто високі для рослинності, можна зафарбувати сірим кольором, і, нарешті, сніг на вершинах гір можна показати, зафарбувавши найвищі вершини ландшафту білим кольором. Фрактальний ландшафт з кольоровою картою на основі вертикального рельєфу показано на Рис. 2.7.
- Атмосферні ефекти – природні явища, такі як небо, хмари і туман, можуть бути додані до ландшафту можна додати природні явища, такі як небо, хмари, туман і річки.
- Освітлення – можна додати використання світла і тіні.
- Простір під землею – наприклад в Minecraft, практично весь цей простір заповнений різноманітними печерами та тунелями.
- Флора і фауна – наповнити деревами, рослинами, камінням і т.д.

РОЗДІЛ 3. РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ДОСЛІДЖЕННЯ ФРАКТАЛЬНОГО ЛАНДШАФТУ

3.1. Опис архітектури застосунку

Головним етапом побудови ландшафту – це визначення висоти для кожної з точок поверхні, тобто створення карти висот. Найбанальніша ідея це пройтись по кожній точці і присвоїти рандомне значення в якихось межах, але як не дивно, це не дасть нормальних результатів, тому треба використовувати певні алгоритми. В цьому застосунку буде застосовуватися алгоритм діамант квадрат.

Реалізація генерації ландшафту буде в классі ‘GenerateTerrain’, він складається повністю із ‘class’ методів.

```
def generate_terrain(width, height, smoothness=1):
    size = smallest_power_of_two_after(max(width,height))
    square_terrain = generate_square_terrain(size, smoothness)
    terrain = [row[:width + 1] for row in square_terrain[:height + 1]]
    return terrain
```

Рис. 3.1. Функція ‘generate_terrain()’

Метод ‘generate_terrain()’ (Рис. 3.1), є головним, і генерує КВ. Розроблятиметься застосунок де можна задати розміри ширини та довжини. Метод приймає параметри ширини(width), довжини(height) та гладкості(smoothness). ‘Width’ і ‘height’ відповідають за розмір ландшафту, параметр ‘smoothness’ контролює рівень гладкості чи різкості, яка застосовується до ландшафту. За замовчуванням параметр smoothness матиме значення 1.

```
def smallest_power_of_two_after(cls, n):
    ret = 1
    while ret < n:
        ret <<= 1
    return ret
```

Рис. 3.2. Функція ‘smallest_power_of_two()’

Метод ‘smallest_power_of_two_after()’ (Рис. 3.2) рахує найменшу ступінь для двійки, результат якого трохи більше, чи дорівнює параметру ‘n’. Він використовує операцію побітового зсуву, що рівнозначно множенню на 2.

```
def generate_square_terrain(cls, size, smoothness):
    if size & (size - 1):
        raise ValueError('Expected terrain size to be a power of 2, received ' + str(size) + ' instead.')
    mat = cls.generate_matrix(size + 1)
    cls.iterate(mat, smoothness)
    return mat
```

Рис. 3.3. Функція ‘generate_square_terrain()’

Ця функція (Рис. 3.3) генерує ‘square_terrain’ – КВ, із зазначеним розміром і гладкістю. Спочатку відбувається перевірка чи є ‘size’ числом від двійки в степені. Функція застосовує

перевірку на спільні біти між ‘size’ та ‘(size – 1)’. Якщо ‘size’ є степенем 2, результатом буде 0. Якщо розмір не є степенем 2, то між ними буде принаймні один спільний біт, і результатом буде ненульове значення, і підніметься помилка ‘ValueError’.

Потім викликає ‘generate_matrix()’ для створення початкової матриці з нулями. Є варіант створення матриці з нулями із застосуванням бібліотеки ‘NumPy’. З точки зору продуктивності, другий спосіб з використанням ‘NumPy’, як очікується, буде швидшим для великих матриць. ‘NumPy’ забезпечує оптимізацію операцій та ефективне використання пам’яті для обчислень з масивами. З іншого боку, перший спосіб, що включає ‘list comperhension’, може бути більш придатним для менших матриць або коли потрібна вкладена структура списків. Так як в цьому застосунку не використовуються великі списки, можна обійтись першим варіантом, та за висновками проведених в цій роботі тестів, він працює швидше.

Після, викликає функцію ‘iterate()’ (Рис. 3.4), щоб застосувати до матриці алгоритм DS, який генерує рельєф із заданим рівнем гладкості.

```
def iterate(cls, matrix, smoothness):
    counter = 0
    num_iteration = math.log(len(matrix) - 1) / math.log(2)
    while counter < num_iteration:
        cls.diamond(matrix, counter + 1, smoothness)
        cls.square(matrix, counter + 1, smoothness)
        counter += 1
```

Рис. 3.4. Функція ‘iterate()’

Функція ‘iterate()’ ітераційно застосовує алгоритм DS до матриці, який в коді розбитий на два окремих методи. Функція приймає матрицю і гладкість і виконує задану кількість ітерацій(‘n’), яка залежить від розміру матриці($2^n + 1$). На кожній ітерації функція викликає diamond(діамантовий крок DS) і square(квадратний крок DS) методи.

Метод ‘diamond’ виконує діамантовий крок DS. Він використовує матрицю наповнену нулями, глибину(ітерація) та гладкість. Він обчислює необхідні змінні для діамантового кроку DS і відповідно оновлює матрицю.

Метод ‘square’ виконує квадратний крок DS. Він приймає матрицю, глибину(ітерація) та гладкість. Він обчислює необхідні змінні для квадратного кроку DS і відповідно оновлює матрицю.

Логіку для цих кроків було описано в попередньому розділі, а їхня реалізація займає певного місця, тому їхню реалізацію можна переглянути в додатках цієї роботи, де можна з ними ознайомитись, також там надруковано наочну реалізацію із застосуванням матриць.

Загалом, клас ‘GenerateTerrain’ інкапсулює логіку генерації квадратних рельєфів за допомогою DS. Він містить методи для генерації рельєфу, ітерації над матрицею та виконання

кожного етапу DS окремо. Метод ‘generate_terrain()’ слугує точкою входу для генерації рельєфу, в той час як інші методи підтримують процес генерації.

Метод ‘square_helper()’ – це допоміжний метод, який використовується у квадратному кроці. Він приймає матрицю, глибину, гладкість і конкретні координати. Він обчислює середнє значення сусідніх висот і додає значення зсуву, щоб оновити матрицю в цільових координатах.

Метод ‘get_h()’ обчислює значення зсуву(в попередніх розділах використовувалось як ‘r’), що використовується для diamond та square методів. Він приймає гладкість і глибину як параметри і повертає розраховане значення зсуву.

Метод ‘average()’ обчислює середнє значення списку чисел. Він отримує список чисел як параметр і повертає розраховане середнє значення.

Класс ‘GenerateTerrain’ забезпечує структурований і модульний підхід для генерації квадратних ландшафтів за допомогою алгоритму діамант-квадрат. Він розділяє різні кроки алгоритму на окремі методи, що полегшує читання, розуміння та модифікацію коду. Методи класу забезпечують легкий доступ до процесу генерації ландшафту без необхідності створювати екземпляр класу.

Щоб згенерувати рельєф, можна скористатися методом класу ‘GenerateTerrain.generate_terrain(sizer, smoothness)’, де sizer представляє бажаний розмір рельєфу, він застосовується як для довжини, так і ширини (пояснення заміни буде далі), а smoothness контролює рівень згладжування, що застосовується. Метод повертає КВ рельєфу.

Використовуючи класс GenerateTerrain, можна створювати карти висот з різними розмірами і рівнями згладжування за допомогою алгоритму діамант-квадрат у структурований і гнучкий спосіб.

В окремому файлі який відповідатиме за візуалізацію буде функція під назвою ‘visualize_terrain_3d()’, вона буде приймати КВ. Для візуалізації використовуються бібліотеки ‘NumPy’ і ‘Matplotlib’.

Спочатку КВ конвертується на двовимірний масив за допомогою функції ‘np.array()’. Далі створюється сітка координат за допомогою функцій ‘np.arange’ і ‘np.meshgrid’. Змінні ‘x’ і ‘y’ містять масиви значень для відповідних осей, а ‘X’ і ‘Y’ містять матриці з усіма можливими парами координат. Потім створюється тривимірний графік за допомогою функції ‘figure.add_subplot(111, projection=’3d’)’. Цей графік буде відображатись на вказаному полотні.

Ландшафт відображається за допомогою функції ‘ax.plot_surface(X, Y, terrain_array, cmap=’terrain’)’. Функція приймає матриці координат ‘X’ і ‘Y’ і матрицю КВ ‘terrain_array’ і створює поверхню ландшафту. Встановлюються мітки осей за допомогою методів

‘ax.set_xlabel’, ‘ax.set_ylabel’, ‘ax.set_zlabel’. Наостанок створюється об'єкт ‘figure’ для відображення графіка, а в функції вкінці викликається ‘plt.show()’ щоб відображати графік.

Тепер для того щоб візуалізувати КВ, можна імпортувати цей файл у виконуваний, звідки буде передаватись наша КВ.

Так як при великій розбіжності ‘width’ і ‘height’, ландшафти генеруються не досить коректно, до прикладу, величини у відношенні 1:8 дають такий результат (Рис. 3.5):

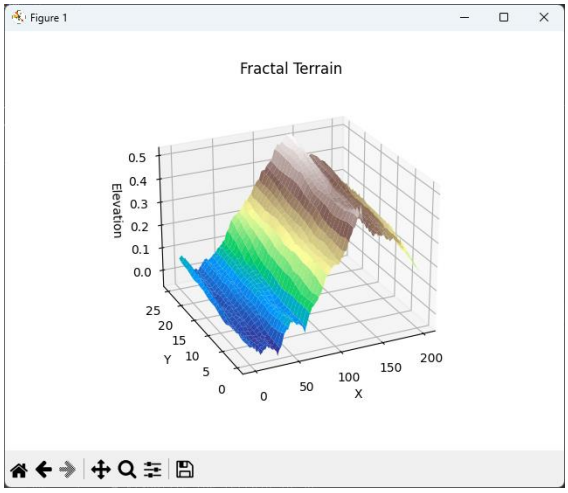


Рис. 3.5. Візуалізація ландшафту із заданим співвідношенням сторін 1:8

Для реалізації повзунків (Рис. 3.6), які дозволяють міняти параметри ландшафту в реальному часі, було використано бібліотеку tkinter. В окремому файлі tk_gui.py, будуть реалізовані повзунки. Імпортується сама бібліотеку ‘tkinter’ для створення головного вікна, і ‘ttk’ модуль з цієї бібліотеки щоб створити повзунки за допомогою ‘ttk.Scale()’ та кнопку для оновлення ландшафту з ‘ttk.Button()’, куди буде передаватись функція ‘update_terrain()’ в параметр ‘command’. Ця функція в свою чергу буде витягувати значення з повзунків за допомогою одноіменної функції ‘get()’.

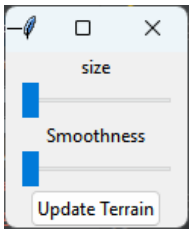


Рис. 3.6. Повзунки для зміни значень для змінних ‘size’ та ‘smoothness’

На даний момент модуль ‘terrain’, який відповідає за генерацію ландшафту не пристосований оновляти чи то змінювати існуючий ландшафт. На даний момент головна функція генерування ландшафту не може змінювати наявний ландшафт, не приймає існуючого та повертає зрізаний ‘square_terrain’. Тепер ця функція буде повертати повний ‘square_terrain’, який в подальшому буде оброблятись, наприклад для візуалізації (Рис. 3.7).

```
def generate_terrain(sizer, smoothness=1):
    size = smallest_power_of_two_after(sizer)
    square_terrain = generate_square_terrain(size, smoothness)

    return square_terrain
```

Рис. 3.7. Модифікована функція ‘generate_terrain()’

Реалізований класс ‘Terrain’ (Рис. 3.8) котрий і буде мати цей ‘square_terrain’, та буде мати метод візуалізації ‘visualize()’, який приймає параметр size, куди буде передавати цю зрізану змінну, яку назвемо ‘terrain’. Візуалізація буде відбуватись за допомогою раніше створеного методу ‘visualize_terrain_3d()’.

```
class Terrain:
    def __init__(self, square_terrain):
        self.st = square_terrain

    def visualize(self, size):
        terrain = [row[:int(size) + 1] for row in self.st[:int(size) + 1]]
        visualize_terrain_3d(terrain)
```

Рис. 3.8. Класс ‘Terrain’

Тож в файлі tk_gui, буде генеруватись ландшафт з максимальним розміром, який буде дозволений в слайдері (Рис. 3.9), і надалі візуалізуватиме його з розміром size = 100

```
size_slider = ttk.Scale(window, from_=1, to=MAX_VAL, value=size, orient=tk.HORIZONTAL)
```

Рис. 3.9. Реалізація повзунка, із максимальним розміром значення змінної ‘size’

При зміні повзунків вийде помилка так як код не пристосований обробляти не цілі значення, в той час як слайдер давав як цілі значення так і float. Щоб це виправити, виконується перетворення значення на ціле за допомогою одноіменної функції ‘int()’, це виконується в отриманні значення від повзунків в функції яка запускається після натискання кнопки для оновлення.

Реалізація функції, яка буде зменшувати КВ шляхом урізання полігональності, для візуалізації того ж ландшафту, але з іншою розмірністю матриці, показана в коді, на Рис. 3.10. Зменшення розміру за довжиною рядків і стовпців видалено.

```python
class Terrain:
    def __init__(self, square_terrain, smoothness=1):
        self.st = square_terrain
        self.smoothness = smoothness
        self.size = len(square_terrain)
        self.power = int(math.log(len(square_terrain) - 1, 2))

    def visualize(self, sizer: int):
        terrain = copy.deepcopy(self.st)
        counter = self.power - sizer
        span = pow(2, counter)
        half = span // 2
        while counter > 0:
            for i in range(0, self.size - 1, span):
                for col in terrain:
                    col[i + half] = None
                terrain[i + half] = [None] * self.size  # 0.09389901161193848
            counter -= 1
            span = half
            half //= 2
        terrain = [list(filter(lambda el: el is not None, row)) for row in terrain if
                   any(el is not None for el in row)]  # 0052, 0046
        # terrain = [row[:int(size) + 1] for row in self.st[:int(size) + 1]]
        visualize_terrain_3d(terrain)
```

Рис. 3.10. Реалізація зменшення полігональності ландшафту

Так як програма створюється для спостерігання фрактального ландшафту, його дослідження та його змінення в реальному часі, передбачувано, що ці зміни можуть застосовуватися повторно. Отож в змінні об’єкта збережено величини ‘size’ – розмір КВ, і ‘power’ – степінь двійки для ‘size’.

У функції ‘visualize’, насамперед створюється копія КВ, щоб не змінювати її, так як потрібно буде не раз звертатись за нею щоб змінити розмір. Для створення копії, через синтаксис Python, не можна просто присвоїти значення одної змінної іншій через знак ‘=’, як наприклад в мові C/C++, тому додатково інстальовано бібліотеку ‘copy’. Створюється копія КВ за допомогою ‘copy.deepcopy()’, яка створює копію його об’єктів і якщо об’єкти теж мають посилання на об’єкти, вона теж створює їх копії, на відміну від ‘copy.copy()’ чи вбудованого рішення від классу ‘list’, які роблять просто поверхневу копію.

Для урізання полігональності, використано спрощений метод, але цей метод працює нічим не гірше в цьому застосунку. Так як в DS при кожній ітерації створюються нові рядки та стовпці значень, вони просто будуть видалятись. В циклі while функція проходиться по ітераціям, тут неважливо, починати з кінця ітерацій, чи з початку, так як тут просто присвоюється статичне значення, яке не обраховується в залежності від сусідів. В циклі for заміняються значення цілих визначених рядків чи стовпців значеннями ‘None’, в майбутньому вони будуть просто видалені. В цьому випадку цикл працює швидше ніж LC, а на етапі видалення(terrain = […]), не значно, але повільніше ніж LC.

Щоб змінювати гладкість, при тому щоб мотив ландшафту був таким самим, потрібно рандомні значення які видає функція get_h при генеруванні ландшафту зберігати. Зміниться функція get_h, куди буде передаватись координата, для якої рахується зсув, і рандомне

значення із його координатами буде зберігатись в список ‘random_values[]’ на цю ж позицію. Адаптований класс генерації, тепер буде створюватись об’єкт цього класу, тобто мати магічний метод ініціалізації, і його функції більше не будуть мати декоратор ‘@classmethod’, але деякі з останніх будуть статичними методами(‘@staticmethod’). Тепер точкою входу для генерації ландшафту буде метод ‘iterate’, так як при ініціалізації створюється пуста матрицю з потрібним розміром і привласнюється вона змінній ‘mat’. Також класс ‘Terrain’ піддався змінам, він буде мати композицію, тепер він не приймає КВ в якості аргументу, а одразу ініціалізує об’єкт классу ‘GenerateTerrain’ в методі ініціалізації, а він в свою чергу має КВ. При першому запуску генератора, так само буде створюватися КВ, але якщо потрібно буде змінити параметр ‘smoothness’, буде викликатись функція ‘set_smoothness()’ (Рис. 3.11), яка збереже цю змінну в стані екземпляру, і функція ‘get_h()’ (Рис. 3.12) тепер буде посилатись на нову функцію ‘new_h()’, яка замість генерації нових рандомних значень, буде застосовувати старі, збережені в ‘random_values[]’.

```
def set_smoothness(self, sm):
    self.smoothness = sm
    self.get_h = self.new_h
```

Рис. 3.11. Функція ‘set_smoothness()’

```
def new_h(self, depth, el):
    h = self.h(depth, self.smoothness)
    rand = self.random_values[el[1]][el[0]]
    return (1 - 2 * rand) * h  # Ignore emphasis
```

Рис. 3.12. Функція ‘new_h()’

Після застосування високого рівня гладкості при деяких генераціях ландшафту появляються помітні виступи. З такою проблемою стикався не тільки я, з тих хто користується генерацією ландшафту. Це пов’язано з тим що на першій чи перших ітераціях коефіцієнт зсуву ще не сильно зменшився. І в такому масштабі осі ‘z’ це дуже помітно (Рис. 3.13).

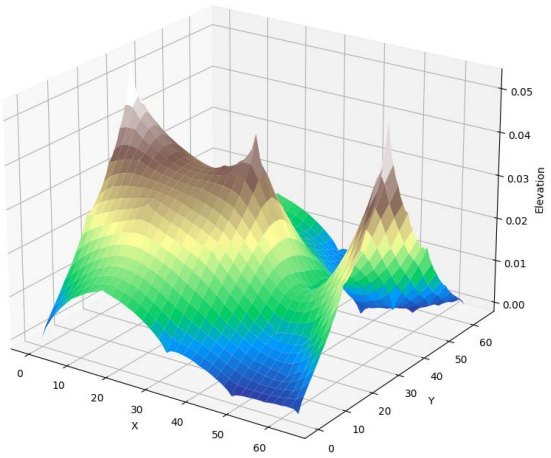


Рис. 3.13. Сильний зсув деяких з вершин при великому масштабуванні осі ‘z’

Це показує що функцію візалізації потрібно адаптувати, так як границі для осей встановлюються автоматично, від крайніх значень КВ, тому тут (Рис. 3.13) співвідношення вертикальної осі до горизонтальної є дуже суттєве, що дає такий виразний зсув в деяких місцях. Щоб цього уникнути робляться сталими границі по горизонталі за допомогою функції ‘set_zlim(-0.5, 0.5)’. Також зміниться співвідношення між горизонтальними та вертикальною осями за допомогою ‘set_box_aspect(2, 2, 1)’ який приймає значення для 3 осей, і буде передано значення 2 для горизонтальних осей, 1 для вертикальної осі. Також встановлено його позицію, щоб він займав все місце полотна, за допомогою ax.set_position([0, 0, 1, 1])

3.2. Реалізація GUI

Matplotlib – це бібліотека для графічних зображень, бібліотека GUI, яка використовує ‘tkinter’ як ‘TkAgg’ в якості backend за замовчуванням. Отож бібліотеку ‘tkinter’ вже було використано для додавання повзунків для взаємодіїї з ландшафтом. Потрібно вбудувати графік в застосунок ‘tkinter’. Бібліотека ‘matplotlib’ має backend під назвою ‘FigureCanvasTkAgg’, який можна використовувати для інтеграції керування і ‘matplotlib’.

```
# Create a blank figure and canvas for Matplotlib plot
canvas = FigureCanvasTkAgg(figure, master=window)
canvas.get_tk_widget().grid(row=2, columnspan=4, sticky="nsew")
```

Рис. 3.14. Виклик функції для відображення фігури в GUI від ‘tkinter’

Тепер для візуалізації в вікні ‘window’, яке раніше було створено, потрібно замість ‘plt.show()’ викликати ‘canvas.draw()’ (Рис. 3.14). Для цього в візуалізацію буде передаватись ‘canvas’ як посилання, звідки він вже буде викликати цю функцію.

Додано відображення значень, які має в даний момент часу повзунок в окреме поле за допомогою такої конструкції (Рис. 3.15). Так само виконано реалізацію для повзунка smoothness.

```
def update_size_label(value):
    size_label_value.config(text=str(int(float(value))) + ' pow of 2 = ' + str(2 ** int(float(value))))
```

Рис. 3.15. а) Функція ‘update_size_label()’, яка викликається слайдером ‘size_slider’

```
size_slider.config(command=lambda value: [update_terrain(value), update_size_label(value)])
```

Рис. 3.15. б) Реалізація виклику функції ‘update_size_label()’ в слайдері ‘size_slider’

Під час цього рішення було помічено, що можливо одразу передавати значення слайдеру в функцію візуалізації, так що кнопка для оновлення ландшафту більше не знадобиться. Присутній невеличкий баг модулю, так як щоб ця конструкція коректно працювала, функція куди передається, обов’язково повинна приймати якийсь аргумент, навіть якщо вона не буде його ніяк застосовувати.

Елементи розташовуються за допомогою ‘grid()’. Біля слайдеру буде висвітлюватись значення слайдеру, для слайдеру ‘size’ буде висвітлюватись степінь двійки, з якої побудовано КВ, і її розмір.

Було помічено що при закритті вікна, програма продовжує працювати. Це малий недолік, який можна виправити наступним чином (Рис. 3.16):

```
def quit_me():
    window.quit()
    window.destroy()
```

Рис. 3.16. а) Функція ‘quit_me()’, яка викликається при закритті вікна застосунку

```
window.protocol("WM_DELETE_WINDOW", quit_me)
```

Рис. 3.16. б) Виклик функції ‘quit_me()’ при закритті вікна застосунку

На даний момент вся робота відбувається з одним ландшафтом, а що якщо потрібно буде змінити його повністю, тобто разом з мотивом ландшафту. Для цього реалізовано кнопку для створення нового ландшафту. Кнопка буде відповідати за наступну команду (Рис. 3.17):

```
def new_terrain():
    global terrain
    terrain = Terrain(MAX_VAL)
    visualize_terrain_3d(terrain.square_terrain.mat, button_var.get(), canvas)
```

Рис. 3.17. Функція ‘new_terrain()’

У візуалізатор тепер передається ще один параметр. Цей параметр отримується від кнопки типу ON/OFF, який відповідає за включення осей координат або ж за виключення. Його реалізація (Рис. 3.18):

```
button_var = tk.BooleanVar()
button = tk.Button(window, text="OFF", command=toggle_button)
button.grid(row=1, column=3, sticky="nsew")
```

Рис. 3.18. а) Реалізація кнопки активації/деактивації осей

```
def toggle_button():
    if button_var.get():
        button_var.set(False)
        button.config(text="OFF")
        update_terrain(False)
    else:
        button_var.set(True)
        button.config(text="ON")
        update_terrain(True)
```

Рис. 3.18. б) Функція ‘toggle_button()’

В свою чергу цей параметр передається в візуалізатор, де він його застосовує (Рис. 3.19):

```
ax.set_axis_off()
if var:
    ax.set_axis_on()
```

Рис. 3.19. Активація/деактивація осей

Для того щоб можливо було змінювати розмір вікна, щоб всі елементи могли адаптуватись, потрібно надати їм вагу (Рис. 3.20):

```
for i in range(4):
    window.grid_columnconfigure(i, weight=1)
window.grid_rowconfigure(0, weight=1)
window.grid_rowconfigure(1, weight=1)
window.grid_rowconfigure(2, weight=10)
```

Рис. 3.20. Визначення ваги кожного зі стовпців та рядків

Де в 2-му рядку розташовується тільки ландшафт.

Щоб додати стандартний toolbar від matplotlib, його потрібно окремо імпортувати з бібліотеки “matplotlib”, його назва “NavigationToolbar2Tk”. Всі елементи в цьому застосунку розташовуються за допомогою методу ‘grid()’, в той час як toolbar можна розташувати лише за допомогою методу ‘pack()’. Тому, щоб його розташувати, спочатку потрібно запакувати у ‘Frame()’, а вже потім його розташовувати за допомогою ‘grid()’ (Рис. 3.21).

```
toolbarFrame = tk.Frame(master=window)
toolbarFrame.grid(row=3, columnspan=3, sticky="nsew")
toolbar = NavigationToolbar2Tk(canvas, toolbarFrame)
toolbar.update()
toolbar.pack(side=tk.LEFT)  # (anchor='w') as an alternative
```

Рис. 3.21. Пакування панелі інструментів у робоче вікно застосунку

Анімація відбуватиметься за допомогою функції ‘FuncAnimation()’ з бібліотеки matplotlib. Буде відбуватись візуалізація покрокової генерації ФЛ за допомогою функції ‘visualize_first_terrain_3d()’, при генерації першого та нових ФЛ. Для цього також буде потрібно додати зберігання стану ФЛ на кожній ітерації. Також буде присутня статична анімація обертання яка реалізована в функції ‘animation_rotate()’. Для виклику двох анімацій, друга анімація(animation_rotate) буде продовженням першої (Рис. 3.22).

```
def visualize_first_terrain_3d(terrain, var, canvas, ani_var):
    terrain_frames = terrain.square_terrain.frames

    def update_figure(i):
        if i == len(terrain_frames):
            if ani_var:
                visualize_terrain_3d(terrain_frames[-1], var, canvas, ani_var)
        else:
            terrain_list = terrain_frames[i]
            visualize_terrain_3d(terrain_list, var, canvas, False)

    animation = FuncAnimation(figure, update_figure, frames=len(terrain_frames) + 1, interval=200, repeat=False)
```

Рис. 3.22. Функція ‘visualize_first_terrain_3d()’

Функція зберігання в toolbar зберігає лиш зображення, для зберігання самого ландшафту як об'єкту 3D прописана функція ‘export_plot()’, яка за допомогою методів mesh бібліотеки numpy-stl зберігає останній візуалізований об’єкт під назвою ‘terrain’ із розширенням ‘stl’ в директорію застосунку. Викликатись ця функція буде при закритті застосунку.

Для створення самодостатнього виконуваного файлу використано бібліотеку ‘PyInstaller’. Прописавши в директорії проекту команду ‘pyinstaller tk_gui.py’, що створить виконуваний файл у директорії ‘dist’. Також перед назвою файлу можна вказати шлях до неї. PyInstaller автоматично аналізує код та створить виконуваний файл разом з необхідними залежностями.

3.3. Шляхи оптимізації

Щоб пришвидшити роботу застосунку, можна перенести реалізацію на швидшу мову програмування, як наприклад C/C++. Хоча також є варіанти компіляції python файлу у виконуваний, який буде мати швидкість виконання не довше ніж в C/C++, але цих методів в цій роботі застосовано не було.

Цей застосунок слід розпаралелити, але через GIL, в цьому випадку, із застосуванням python це зробити неможливо. Для цього потрібно переносити рішення для генерації ландшафтів наприклад на мову C/C++, та вже після вбудовувати це рішення в цю програму, яка написана на python.

Розглядаючи це рішення, слід зазначити, що полотно, на якому малюється графік, заповнює не всю область, яка на неї відведена, тобто в мірках вікна tkinter, це другий рядок.

Проблема застосунку в тому, що при зміні параметрів фігури, вона постійно перебудовується. Можливо знайти інше рішення, для зміни об’єкту в реальному часі. Методи для візуалізації об’єкта можуть викликатись безперервно, в момент рухання повзунка, тому краще реалізувати цю функцію без умов типу ‘if’.

Є сенс розглянути інші бібліотеки для реалізації візуалізації чи GUI, так як при візуалізації великих об’єктів, ‘matplotlib’ починає довго грузити, і для інших задач окрім візуалізації ця бібліотека не підійде. Для візуалізації можна розглянути такі бібліотеки як ‘OpenGL’, та ‘mayavi’, яка побудована на першій. Цікавий факт, на сайті бібліотеки ‘OpenGL’ є різні зображення, деякі з них це фрактали. Їх можна використовувати для побудови із взаємодією з навколишнім світом, наприклад ігор чи систем з автоматизованим проектуванням. Для реалізації складних gui можна розглянути ‘PyQt5’, який ‘matplotlib’ також може використовувати в якості бекенду.

Слід зазначити, що робота з КВ ландшафту відбувається через списки, але варто розглянути спосіб реалізації за допомогою масивів, так як тут використовується тільки читання та видалення за індексом, та не додаються нові значення в кінець списку.

3.4. Демонстрація роботи застосунку

Застосунок запускається при запуску головного файлу під назвою ‘tk_gui.py’ або ж виконуваного файлу. Фінальна версія програми виглядає наступним чином (Рис. 3.23):

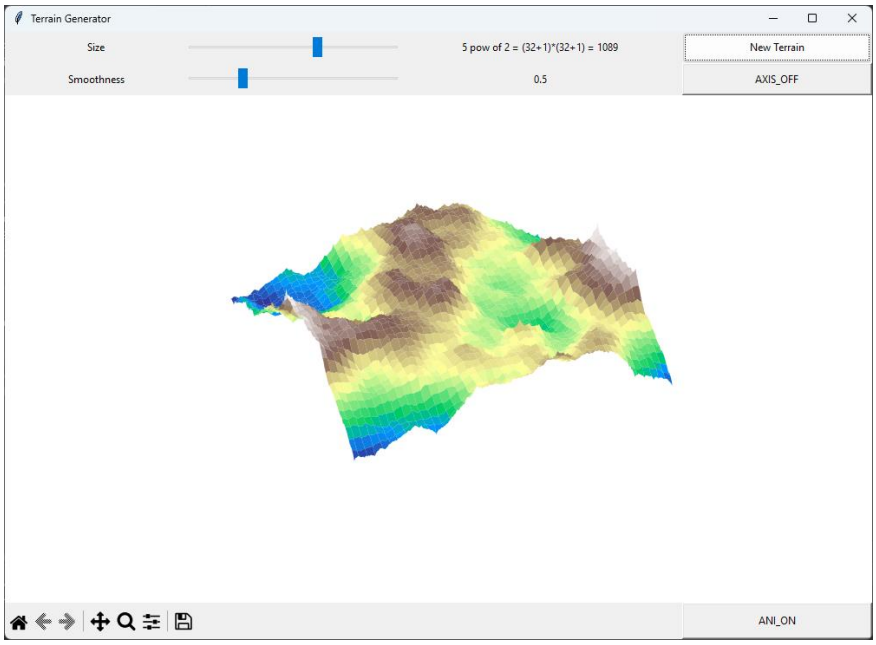


Рис. 3.23. Робоче вікно застосунку 'Terrain Generator'

Це вікно, яке відкривається з заданим розміром 1000x700 пікселів, яке можна масштабувати та розгортати на повний екран. Воно містить фрактальний ландшафт, побудований за допомогою DS алгоритму. Цей ФЛ заданий максимальною величиною 8, який відображається із початку із зменшеним значенням 5, а полігонів (чи точок вершин) в ньому $(2^5 + 1)^2 = 1089$.

Вікно має в собі 10 основних елементів, а саме:

- Повзунки та їхні найменування(2 sliders, 2 labels).
- Значення повзунків.
- Toolbar – панель інструментів, що поставляється стандартно при відображенні за допомогою бібліотеки ‘matplotlib’.
- Три кнопки, одна з яких відповідає за статичну функцію(New Terrain), а інша налаштовує відображенням осей, яка за замовчуванням їх вимикає(AXIS_OFF), і третя кнопка в кутку зправа знизу яка відповідає за включення/виключення анімації обертання, яка за замовчуванням увімкнена(ANI_ON).

Рухаючи повзунки можна динамічно змінювати ландшафт.

- За допомогою слайдеру ‘Size’ можна змінювати кількість полігонів, з яких він будується, ця кількість рахується як $(2^{Size} + 1)^2$. Повзунок видає тільки цілі числа, це пов’язано з специфікою коду.
- За допомогою слайдера ‘Smoothness’, можна змінювати гладкість ландшафту, чим менше цей параметр, тим більш ландшафт буде різкий, чим більше(до 2-ох), тим він рівніше і гладкіше.

Є дві кнопки:

- Кнопка ‘New Terrain’, дає новий згенерований ФЛ, який можна далі так само досліджувати.
- Кнопка ‘AXIS_OFF’, відповідає за відображення осей, нажавши на неї можна переключити її на значення ‘AXIS_ON’, що включить їх відображення, і навпаки.

Також можна керувати ландшафтом за допомогою миші:

- Зажимаючи ліву кнопку миші можна рухати ландшафт.
- Зажимаючи праву кнопку миші і рухаючи вперед чи назад, можна його масштабувати.

На панелі інструментів, яка надається бібліотекою ‘matplotlib’ є функціональні кнопки:

- 1-ша кнопка відповідає за початковий стан налаштувань.
- 2 і 3 кнопки не знадобляться, так як вони відповідають за переключення між різними фігурами, які відображаються водночас, в цьому застосунку відображається лиш одна фігура в момент часу.
- За допомогою 4 і 5 кнопок можна рухати та наближати відповідно, ці ж дії можна виконуати двома кнопками миші, що описувалось раніше.
- За допомогою 6-ої кнопки можна налаштувати розташування фігури.
- За допомогою 7 кнопки можна зберегти ФЛ у вигляді простого фото

При закритті застосунку, останній стан візуалізованого ФЛ буде збережено у форматі ‘stl’(розширення для 3D об’єктів) поряд із виконуваним файлом застосунку(tk_gui.py).

ВИСНОВКИ

В процесі виконання бакалаврської роботи, було зроблено:

- Літературний огляд за темою ‘використання концепції фракталів у комп’ютерній графіці’.
- Досліджено поняття фракталу, його різновиди та історію виникнення.
- Ознайомлено з теорією фракталів, ознайомлено з фрактальною розмірністю.
- Оглянуто деякі з основних застосунків з використанням концепції фракталів в комп’ютерній графіці, зокрема для генерації ландшафтів, хмар, дерев та інших природних об’єктів. Використання фракталів у цих контекстах дозволяє отримати високоякісні та реалістичні зображення, які схожі на природні об’єкти.
- Ознайомлено з основними алгоритмами для генерації фрактальних ландшафтів.
- Розроблено застосунок для генерації фрактального ландшафту та проведення його дослідження. Цей застосунок дозволяє створювати різноманітні фрактальні ландшафти з різними параметрами та відтворювати їх з високою точністю якими можна керувати за допомогою графічного інтерфейсу користувача.

Можна зробити висновок, що в хаосі, який оточує світ, справді присутні ідеальні форми, людство зустрічається із фракталами буквально у всіх сферах життя, тому що концепція фракталів є дуже простою та допомагає в різних галузях науки, як наприклад виконання високоефективних, низькопрофільних та багатодіапазонних антен, та разом з цим зменшувати її розмір, а також полегшує виконання певних задач в ІТ, таких як: візуалізація різних природних об’єктів, явищ чи руху, можна легко виконувати стиснення зображень і їх розпаковуванням, із мінімальною втратою їх якості.

Відкриття фракталів змінило багато традиційних уявлень про геометрію, а в історії розвитку математики стало переломним моментом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mandelbrot, Benoit B. The fractal geometry of nature, W.H. Freeman and company, 1983 p., 976 c.
2. Mandelbrot, Benoit B. Fractals: Form, chance, and dimension, W.H. Freeman and company, 1977 p., 527 c.
3. Mandelbrot, Benoit B. How long is the coast of Britain?, Science Journal, 1967 p., 157 c.
4. Пікавер К.А.. The Math Book From Pythaogoras to the 57th Dimension, 250 Milestones in the History of Mathematics. Sterling Publishing Co, 2009 p., 527c
5. Банах Т.О., Ардан Р.В., Радул Т.М.. Детерміністичні фрактали. ЛНУ ім. І.Франка. 1997 р., 20 с.
6. Поняття фрактала та історія появи фрактальної графіки, Физика, математика ТОЕ. Лекции, задачи, учебники, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: http://fismat.ru/wincom/osnov_info53.html.
7. Barnsley, M. F., Fractals Everywhere, Academic Press, 1993, 560 c.
8. Fractal Software – застосунки для дослідження фракталів, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: https://fractalfoundation.org/resources/fractal-software/
9. Fractal Terrain Generator, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: http://qiao.github.io/fractal-terrain-generator/demo/
10. Julia set color mapping, stack overflow, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: https://stackoverflow.com/questions/49699326/julia-set-color-mapping
11. Біографія Вейєрштрасса, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: https://mathshistory.st-andrews.ac.uk/Biographies/Weierstrass/
12. Fractals and Scaling: Fractal landscapes, Complexity Explorer, [Електронний ресурс]: [Youtube] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=4_nDNb1DN88&t=1814s
13. Фрактали – Наука чи краса? Державний університет телекомунікацій, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: https://dut.edu.ua/ua/news-1-562-7157-fraktali-%E2%80%93-nauka-chi-krasa_kafedra-vischoi-matematiki-matematichnogo-modelyuvannya-ta-fiziki
14. Ercan,A.B., Fractals in science, Springer Berlin, 1994 p., 300 c.
15. Fractal Analysis. Applications in Physics, Engineering and Technology. Edited by Fernando Brambila, InTechOpen, 2017 p., 294 c.
16. Terragen™, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: https://planetside.co.uk/free-downloads/terragen-4-free-download/

17. Gulick D., Scott J., The beauty of Fractals, The Mathematical Association of America, 2010 р., 106 с.
18. Fractals and the Fractals Dimension – Фрактальна розмірність, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: https://web.archive.org/web/20080513224548/http://www.vanderbilt.edu/AnS/psychology/cogsci/chaos/workshop/Fractals.html
19. Gasket Online - Веб-сайт для генерації “Гри хаосу” [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: http://www.shodor.org/gasket/
20. Fractals, Mathematics of Computer Graphics and Virtual Environments, [Електронний ресурс]: [Youtube] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=bEXW7V9mATU
21. Fractal landscapes, Mathematics of Computer Graphics and Virtual Environments, [Електронний ресурс]: [Youtube] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=4_nDNb1DN88&t=1814s
22. Landscape generation using midpoint displacement, bites of code, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: https://bitesofcode.wordpress.com/2016/12/23/landscape-generation-using-midpoint-displacement/
23. Fournier A., Fussell D., Loren A. Graphics and Image Processing, Computer Rendering of Stochastic Models, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: https://dl.acm.org/doi/pdf/10.1145/358523.358553 -
24. Алгоритм diamond-square для побудови фрактальних ландшафтів, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: https://habr.com/ru/articles/111538/
25. Falconer K., Fractal Geometry: Mathematical Foundations and Applications, John Wiley & Sons, Ltd, 2003 р., 400 с.
26. The Weierstrass function, ResearchGate, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: https://www.researchgate.net/figure/The-Weierstrass-function-W-x-Top-b-1-and-ab-1-regular-curve-Middle-b-2-and_fig1_330552507
27. Ігри хаосу. Трикутник Серпінського. Фрактали, механіко-математичний факультет ЛНУ ім. І. Франка, [Електронний ресурс]: [Веб-сайт] – Режим доступу до ресурсу: http://mmf.lnu.edu.ua/ar/1966