Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення для тривимірної візуалізації фракталів»

08-54.БКР.004.00.000 ПЗ

Виконав: студент 4 курсу, групи 1СП-21б
спеціальності 123 — «Комп'ютерна інженерія»
освітня програма — «Системне програмування»
Бурдейний О.В.

Керівник к.т.н., доцент каф. ОТ
Черняк О.І.

Рецензент д.т.н., професор, завідувач кафедри ПЗ
Романюк О.Н.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«16» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо— кваліфікаційний рівень бакалавр
Галузь знань —12 Інформаційні технології
Спеціальність —123 Комп'ютерна інженерія
Освітня програма — Системне програмування

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н.проф._____Азаров О. Д.
«21» березня 2025 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

08–54.БКР.004.00.000 ПЗ

Бурдейному Олегу Володимировичу

1 Тема роботи: «Програмне забезпечення для тривимірної візуалізації фракталів», керівник роботи Черняк О.І. к.т.н., доц., доцент кафедри ОТ затверджені наказом вищого навчального закладу від «20» березня 2025 року № 97.

2 Термін подання студентом роботи 13.05.2025 р.

3 Вихідні дані до роботи: розробка програмного забезпечення з тривимірної візуалізації фракталів, використання методику тривимірного графічного подання Raymarching, побудова основного шейдер мовою, використання мову C# для побудови структури застосунку, забезпечення можливісті взаємодії користувача з основним шейдером через графічний інтерфейс.

4 Зміст розрахунково-пояснювальної записки: вступ, аналіз характеристик і способів побудови фракталів, огляд та вибір програмних технологій реалізації фракталів, розробка програмного забезпечення, висновки, перелік використаних джерел, додатки.

5 Перелік графічного матеріалу: діаграма алгоритму роботи програмного

забезпечення.

Перелік ілюстративного матеріалу: вигляд головного меню розробленого додатка.

6 Консультанти розділів роботи наведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Дата		Підпис
		завдання видав	завдання прийняв	
Спеціальна частина	к.т.н., доц., доцент кафедри ОТ Черняк О.І.	21.03.2025	13.05.2025	
Нормоконтроль	ас. каф. ОТ Швець С.І.	14.05.2025	30.05.2025	

7 Дата видачі завдання 21.03.2025 р. Календарний план для виконання етапів роботи наведено в таблиці 2.

Таблиця 2 — Каледарний план

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вибір, узгодження та затвердження теми БКР	21.03.2025	21.03.2025	виконано
2	Аналіз літературних джерел та попередня розробка основних розділів	21.03.2025	25.03.2025	виконано
3	Огяд існуючих програмних технологій	26.03.2025	29.03.2025	виконано
4	Розробка основної методики реалізації імітації тривимірного простору	30.03.2025	20.04.2022	виконано
5	Розробка архітекутри програмного забезпечення	21.04.2025	25.04.2025	виконано
6	Побуда графічного інтерфейсу користувача та проведення тестування продуктивності	26.04.2025	27.04.2025	виконано
7	Оформлення ПЗ та ілюстративного матеріалу	28.04.2025	11.05.2025	виконано
8	Попередній захист БКР, доопрацювання, рецензування БКР	13.05.2025	13.05.2025	виконано

Студент _____ Бурдейний О.В.

Керівник роботи _____ Черняк О.І.

АНОТАЦІЯ

УДК 004.04

Бурдейний О. В. Програмне забезпечення для тривимірної візуалізації фракталів. Бакалаврська кваліфікаційна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Системне програмування». Вінниця: ВНТУ, 2025. 117 с.

На укр. мові. Бібліогр.: 20, рис.: 74, табл.: 1.

У бакалавравській кваліфікаційній роботі розроблено програмне забезепення для тривимірної візуалізації фракталів з використанням сучасних графічних методик як Raymarching. Під час розробки було проведено аналіз існуючих технологій та аналогів програмного забезпечення, після чого обрано основний рушій, мову програмування застосунка та мову програмування шейдера зважаючи на поставлену мету оптимальної роботи на існуючих персональних комп'ютерах без використання значних системних та часових ресурсів. Під час розробки програмнрого забезпечення було створено інноваційну методику імітації тривимірного простору з поєднанням існуючих підходів та методик.

Ключові слова: фрактали, шейдер, HLSL, Unity, C#, Raymarching.

ABSTRACT

Burdeiniy O. V. Software for three-dimensional visualization of fractals. Bachelor's bachelor's qualification work in the specialty 123 “Computer Engineering”, educational program “System Programming”. Vinnytsia: VNTU, 2025. 117 p.

In Ukrainian. Bibliography: 20, figures: 74, tables: 1.

In the bachelor's qualification work, the software for three-dimensional visualization of fractals using modern graphical techniques such as Raymarching was developed. During the development, an analysis of existing technologies and software analogues was conducted, after which the main engine, application programming language and shader programming language were selected with the goal of optimal operation on existing personal computers without using significant system and time resources. During the development of the software, an innovative methodology for simulating three-dimensional space was created with a combination of existing approaches and techniques.

Keywords: fractals, shader, HLSL, Unity, C#, Raymarching.

ЗМІСТ

ВСТУП

Розвиток сучасних технологій та обчислювальних потужностей ком'ютерів приводить до появи нових можливостей у вирішенні більш складних математичних задач. Серед таких задач можна відзначити функції та геометричні об'єкти, що мають неперервність та повторювану самоподібність [1]. Такі об'єкти називаються фракталами. Вони визначені як самоподібні множини у двовимірному просторі, зокрема, наприклад, у комплексній площині. Найбільш часто фрактали застосовуються у графічних додатках з високодеталізованою реалізацією фонових сцен. В основному тут використовуються двовимірні фрактали. Розширення можливостей і якості графічних додатків породжує задачі реалізації фракталів у тривимірному просторі, що дозволяє розширити застосування цих додатків.

Актуальність даної роботи полягає в тому, що під час швидкісної динамічної зміни сцен у графічних додатках висуваються високі вимоги до їх деталізації. Це потребує значних апаратних і програмних ресурсів. Тому виникає необхідність зменшення цих вимог і отже, розширення сфери використання графічних додатків для існуючих персональних комп'ютерів.

Мета роботи полягає в наданні можливості динамічної візуалізації тривимірних фракталів зі зміною властивостей в реальному часі, яка не потребує значних апаратних і програмних ресурсів. Для цього потрібно розробити відповідні підходи щодо генерації та відображення, кожен з яких буде специфічним у реалізації відповідно до особливостей побудови того чи іншого фракталу.

Для реалізації вказаної мети потрібно вирішити такі **завдання**:

— провести аналіз характеристик і способів побудови фракталів;

— виконати огляд та вибір програмних технологій [2] реалізації фракталів;

— розробити метод імітації тривимірних фракталів;

— розглянути альтернативний метод реазіації візуалізації;

— побудувати архітектуру програмного забезпечення;

— налагодити зв’язок між користувачем та шейдером через графічний інтерфейс;

— провести тестування продуктивності.

Об’єктом дослідження даної бакалаврської дипломної роботи є фрактальні структури, а також їх властивості [3].

Предметом дослідження є методи візуалізації фракталів у тривимірному просторі для персональних комп’ютерів з використанням сучасних графічних технологій.

Наукова новизна полягає в тому, що запропоновано метод програмної тривимірної візуалізації фракталів в динамічному режимі, який на відміну від існуючих, виконує імітацію такої реалізації шляхом накладання двовимірної текстури на сферу, котра оточує камеру огляду. Це дозволяє виконувати високодеталізовану реалізацію рухомих сцен з використанням існуючих персональних комп’ютерів за рахунок зменшення ресурсів пам’ятті та часу.

Варіативність дослідження підтверджується аналізом альтернативного методу під час проведення розробки. Подальші дослідження показали переваги основного методу.

Практичне застосування. Розроблене програмне забезпечення запропонованого методу може бути використано при розробці ігрових, наукових, медицинських та інших проектів, що потребують якісного графічного оформлення.

Апробація. Результати дослідження за темою бакалавравської роботи "Програмне забезпечення для тривимірної візуалізації фракталів" були представлені у доповіді на LIV Всеукраїнській науково-технічній конференції ВНТУ 2025 року. За матеріалами доповіді здійснено **публікацію**, що доступна за посиланням: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/author/submission/23904 [4].

1 АНАЛІЗ ХАРАКТЕРИСТИК І СПОСОБІВ ПОБУДОВИ ФРАКТАЛІВ

1.1 Поняття фракталів

В основі побудови нашого Всесвіту потсійно трапляються об’єкти чи явища, котрі мають яскраво виражену самоподібність у кожній частині себе, незважаючи на природнє походження [5]. До прикладу крони дерев, будова гірських хребтів, форми хмар, сніжинок, кровоносних судин істот — є яскравим прикладом геометричних явищ із притаманними фрактальними ознаками (рисунок 1.1).

Рисунок 1.1 — Вигляд кристалу снігу під мікроскопом

У загальному, поняття фракталу можна інтерпретувати, як назва математичного об’єкту, що характеризується наявністю структурної подібності своїх частин до одного цілого на різних рівнях масштабу. Більшість існуючих фракталів, визначаються ітеративними та рекурсивними процесами, під час яких одна проста формула застосовується багаторазово до отриманого вихідного значення. До прикладу, Множина Мандельброта [6], яка є одним з найвідоміших фракталів, будується через ітеративні обчислення функції комплексної змінної, що має вигляд $z_{n+1}=z_n^2+c$.

Фрактали, змінюють звичайний спосіб осмислення побудови нашої реальності. Зокрема в нашому світі, що побудований на хаотичних і неструктурованих об’єктах, проявляються складні, однак зрозумілі закономірності. Тому зараз, фрактальні об’єкти намагаються використовувати для моделювання природніх рельєфів у графічних застосунках, обробки зображень та опису фізичних складних систем як дифуція чи турбулентність.

1.1.1 Перша згадка

Першим описом фракталу, ще до появи цього терміну, підлягали функцію, які були неперервними й недиференційованими на всій області визначення. Таку функцію було представлено Карлом Веєрштрассом [7]. Ще в 1975 році, в книзі «Фрактальні об'єкти: форма, випадковість і розмірність», французько—американський метематик, економіст, вчений та письменник єврейського походження Бенуа Мандельброта стверджував, що фрактали об’єднують математичну абстракцію та складність живу складність реального світу. У його роботі вперше було введено поняття фрактал, котре позначало розбитість та зламаність з латинської мови. Фрактал було використано як опис геометричних форм, що не можна було описати стандартною евклідовою геометрією.

Прикладом першого фрактала, що відповідав визначеним згодом фрактальним властивостям можна вважати Криву Коха, що була винайдена шведськми математиком Гельґе фон Кохом у 1904 році [8]. Хоча терміна фрактала тоді ще не було, однак це був яскравий зразок неперевної та постійно ламаної кривої, через що у жодній її точці не може існувати дотичної. Після чого на її основі було створено інші структури із закономірностями, як Килим Серпінського, Крива Пеано чи множина Жюліа. Вивчаючи ці множини, Бенуа Мандельброт, вивів свою одну з найвідоміших фрактальних множин з іменем у власну честь, яка мала відповідні властивості нескінченної складності та самоподібності.

1.1.2 Застосуваня під час вирішення парадоксів

Дослідження Бенуа Мандельброта під час вивчення фракталів та їх властивостей, дозволило вирішити деякі складні задачі математики. Зокрема, запропонований Льюїсом Річардсоном парадокс берегової лінії [9] довго був відомим як задача неможливого виконання, через те, що в залежності від масштабу вимірювання, берег буде мати різну довжину (рисунок 1.2).

Рисунок 1.2 — Візуалізація парадоксу берегової лінії

З рисунку вище, видно графічне зображення території Великої Британії, берегова лінія якого визначена трьома різними довжинами відрізків, а саме 200, 100 та 50 кліометрів відповідно. Зважаючи на вибраний підхід вимірювання, із зменшенням кроку виміру, збільшується загальний периметр. Тому, в статті 1967 року — “Яка довжина берегової лінії Британії? Статистична самоподібність і дробова розмірність” [10], Мандельброт зробив значний внесок у дослідження, обґрунтувавши фрактальну природу берега.

1.2 Способи представлення фракталів

1.2.1 Ручне рисування

Найпростішим та найпершим способом візуалізації фракталів було ручне малювання, котре використовувалось для першого відображення кривої Коха. Для цього нам потрібно намалювати відрізок, після чого розділити його на три рівні умовні частини, центральну частину замінити на з’єднанні два відрізка тієї ж довжини, однак під кутом 60 градусів, так, наче посеред відрізка було розміщено рівносторонній трикутник. Після чого, кожну з найменших частин утвореної кривої, що можна вважати відкрізками, потрібно просто замінити на зменшені копії отриманої кривої так, ніби ми застосували вище згаданий процес розміщення двох відрізків під кутом до кожного із існуючих сегментів. Результатом таких операцій стане нескінченно довга крива невизначеної розмірності, оскільки з кожною послідуючою ітерацією, довжина кривої складатиме добуток довжини початкового відрізку від числа 4/3 в степені номера ітерації (рисунок 1.3).

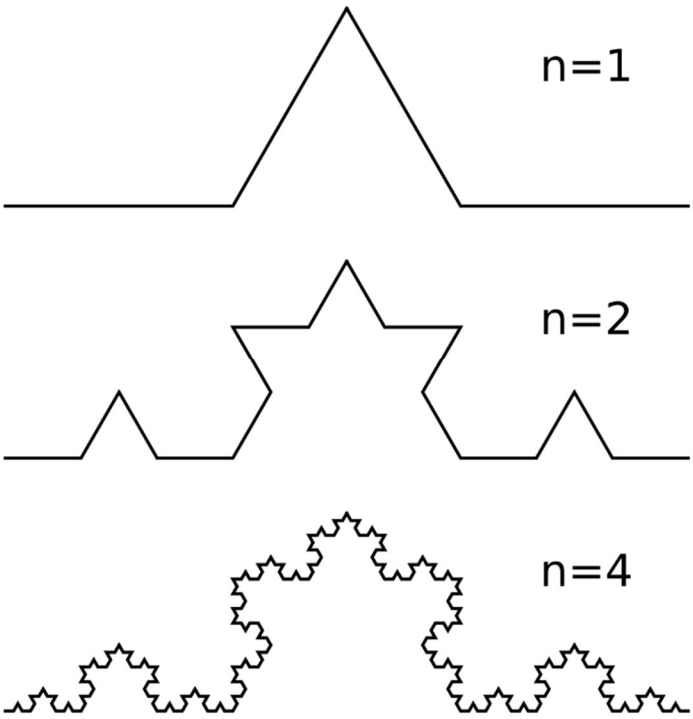


Рисунок 1.3 — Ітераційний вигляд Кривої Коха

1.2.2 Принтерна візуалізація

Згодом, із розвитком комп'ютерів та їх обчислювальних можливостей, у 1978 році було обчислене й надруковане перше в історії зображення набору Мандельброта [11], зображення фрагмента даного документа Роберта Брукса та Дж. Пітера Мательського показано на рисунку 1.4. Ці вчені математики використали основни комплесної динаміки, що згодом використовувались у Множині Мандельброта. Для обчислення фракталу використовувався десятковий комп'ютер IBM 1401, оскільки він був одним із перших, котрий міг обробляти великі обсяги даних і виконувати складні математичні обчислення. Написання коду для програмного обчислення, використовувало мову програмування асемблера, а для графічного виводу було використано лінійний принтер, який друкував порядок символів, що загалом утворювали контури зображення множини фракталу.

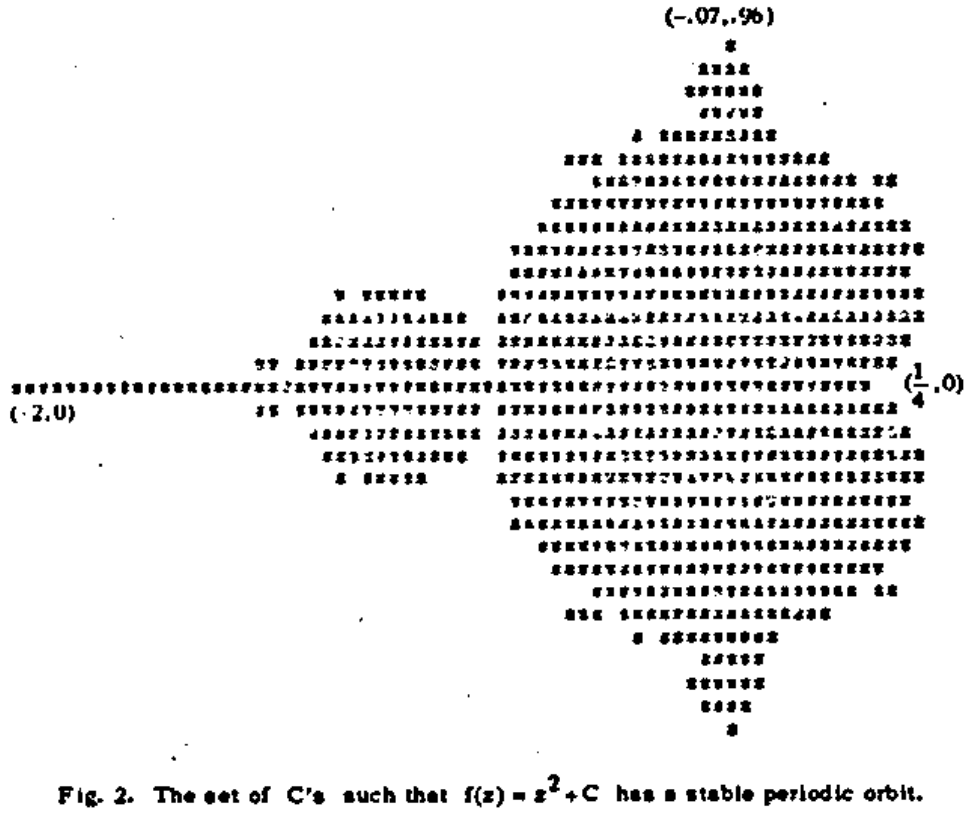


Fig. 2. The set of C's such that $f(z) = z^2 + C$ has a stable periodic orbit.

Рисунок 1.4 — Перша візуалізаціїя Фракталу Мандельброта

1.2.3 Виведення на екран монітора

Найпоширенішим у сьогоденні способом візуалізації фракталів є використання написаних для персональних комп’ютерів скриптів, що дозволять обчислити той чи інший фрактал у двовимірному просторі, та відповідно до вивести графічний результат на екран монітору.

Першим представленням фракталів у графічних моніторах, вважається робота Бенуа Мандельброта в 1980— их роках, під час якої, названий математик зміг відобразити [12] фрактальну Множину Мандельброта (рисунок 1.5). Для досягнення цієї мети використовувались комп’ютери IBM у дослідницькому центрі компанії IBM Research, яка знаходилась у Сполучених Штатах Америки. Для кожного пікселя на екрану, було обчислено та відображено результат розрахунків, від якого залежав колір. Щоб обмежити під час кожної ітерації точки на площині, було використатно алгоритм обчислення комплексних чисел.

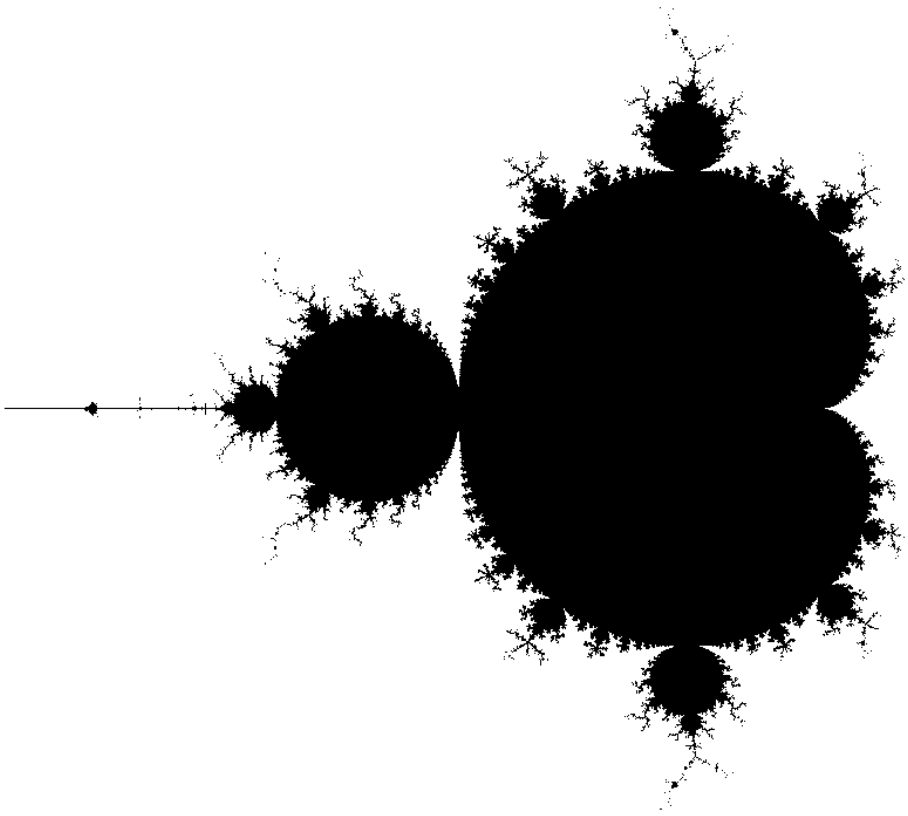


Рисунок 1.5 — Вигляд графічно візуалізованої Множини Мандельброта

1.3 Використання фракталів у інженерії

1.3.1 Електроніка

Фрактали іноді використовуються як основна форма під час виготовлення транзисторів, зокрема високошвидкісних. Оскільки тразистори це елементи, що використовуються в майже кожному електричному пристрої, то і коло застосувань є обширним, зокрема телевізори, телефони, мікрохвильові печі.

Підримка фрактальної форми дозволяє транзисторам працювати без втрат продуктивності в більшому діапазоні частот [13]. Такі напівпровідникові елементи мають абревіатуру HEMT (High Electron Mobility Transistor) і являють собою польовий транзистор, що використовує різні ширини забороненої зони для двох контактів навіпровідникових матеріалів. Головною перевагою використання такого підходу до створення транзисторів, є те, що існуюча самоподібність (рисунок 1.6) дозволяє передбачувати певні властивості під час виготовлення напівпровідникових компонентів, а звивиста форма затворів — знижувати втрати потужності та покращувати швидкодію, управляючи рухом електронів.

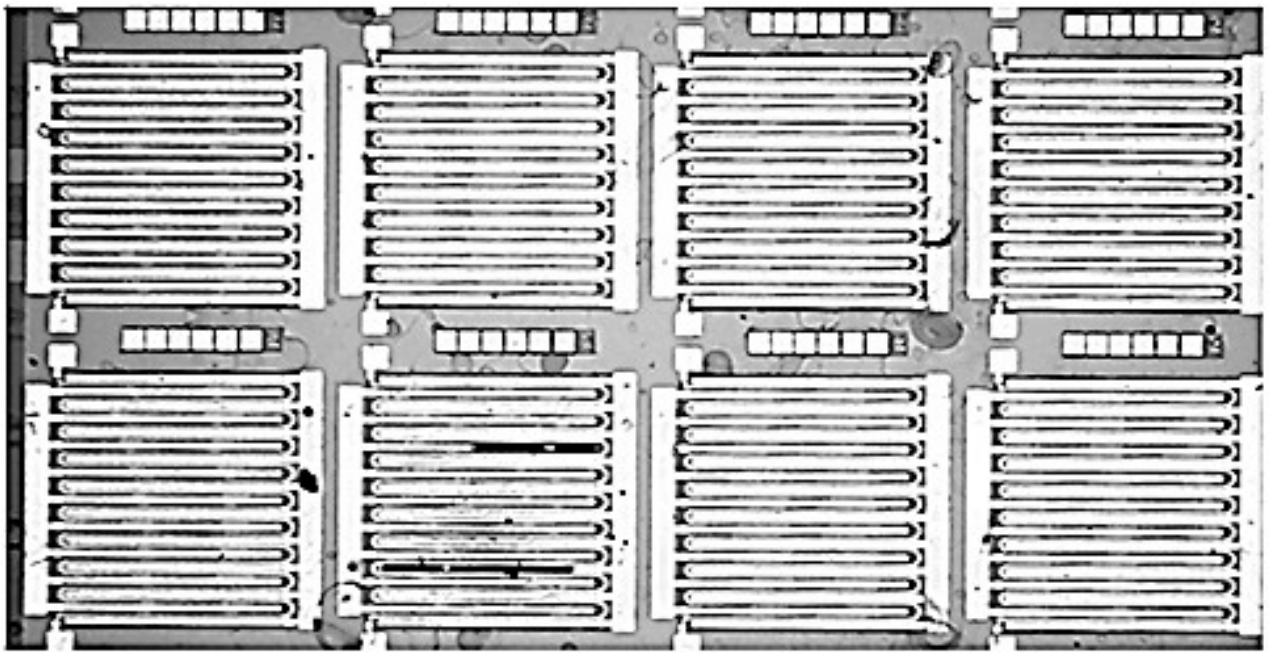


Рисунок 1.6 — Оптичне зображення затворів HEMT транзисторів

1.3.2 Побудова антен

Одним із найпопулярніших застосувань фракталів сьогодення в інженерії є використання фрактальних форм під час побудови антен для передачі та отримання даних. Самоподібність форми в будові антени забезпечує кращу якість сигналів та доступність до ширного діапазону частот.

Прикладом використання фрактальних форм у побудові антен є антена Vicsek (рисунок 1.7), що має форму квадрата, побудованого з менших квадратів, а ті в свою чергу — із ще менших квадратів [14]. Така властивість пристрою дозволяє однакого взаємодіяти з різними частотами сигналу, як для 100МГц так і для 10000МГц, відповідно маючи для цього визначені розміри квадратів.

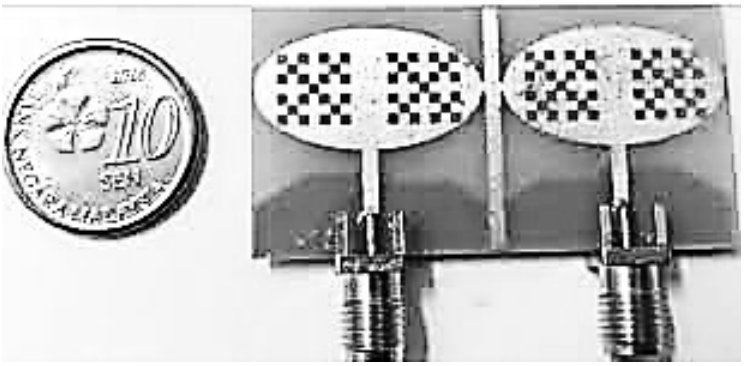


Рисунок 1.7 — Вигляд антени Vicsek

Перші телефони та інші пристрої мали специфічну висувну антенту, котра мала відповідно визначену робочу довжину для сприйняття конкретних частот. Однак за цим же принципом використання фрактальних форм, допомогло уникнути використання громістких видовжених антен на подібних пристроях, а також підвищити швидкість зв’язку.

1.3.3 Пристрої теплообміну

Теплообмінниками вважаються пристрої, які використовують для передачі тепла до зон пониженої температури, аби відвести зайве тепло від джерела, що його створює. Прикладами таких пристроїв теплообміну можуть бути радіатори на сучасних графічних чи центральних процесорах персональних комп’ютерів.

Форма й розміри теплообмінників будуть залежати від потреб щодо відведення тепла. Основною перевагою теплообмнінників у інженерії є підтримка робочих тепературр електронних пристроїв, що дозволятиме працювати та підтримувати продуктивність на визначеному розробником рівні.

Сніжинка Коха, відомий фрактал, який виористовувався у теплообмінниках літаків також. Пристрої теплообміну, що мають фрактальну форму, дозволяють зекономити матеріал виробництва теплообмінника в порівнянні із звичайнимим формами [15], при цьому збільшити кількість відведеного тепла (рисунок 1.8).

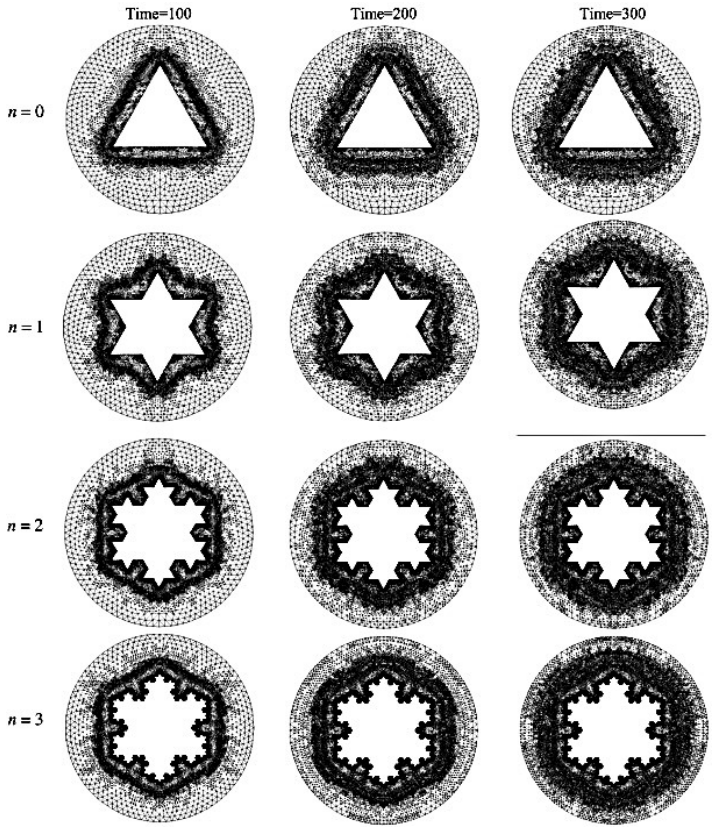


Рисунок 1.8 — Фрактальний теплообмінник у формі Сніжинки Коха в перерізі

Значним мінусом таких теплообмінників є лише більша ціна через складність виготовлення, фрактальних форм. Однак, як згадувалось вище, зменшена площа теплообмінника за рахунок конкретного візерунку — зменшить кількість витраченого матеріалу, а також зменшуючи вагу — точно повпливає на зниження ціни.

1.3.4 Обробка сигналів

Технології сьогодення тісно пов’язані з обробками сигналів різноманітних типів. Всі методи обробки мають на меті схожі завдання, а саме аналіз певних даних, зібраних за конкретний проміжок часу, а потім вивід важливих закономірностей у результаті обробки [16]. Прикладами самоподібності сигналів можна виділити октави звукових хвиль. Оскільки нота кожна нота в одній октаві, буде самоподібним звуком до цієї ж ноти у інші октаві.

Бенуа Мандельброт займався інтентсивним вивченням цінових графіків як фрактальних сигналів [17], адже цінові графіки також мають видиму самоподібність даних на різних інтервалах. До прикладу розглянемо курс долару в Україні за період 5 років та загальний (рисунок 1.9).

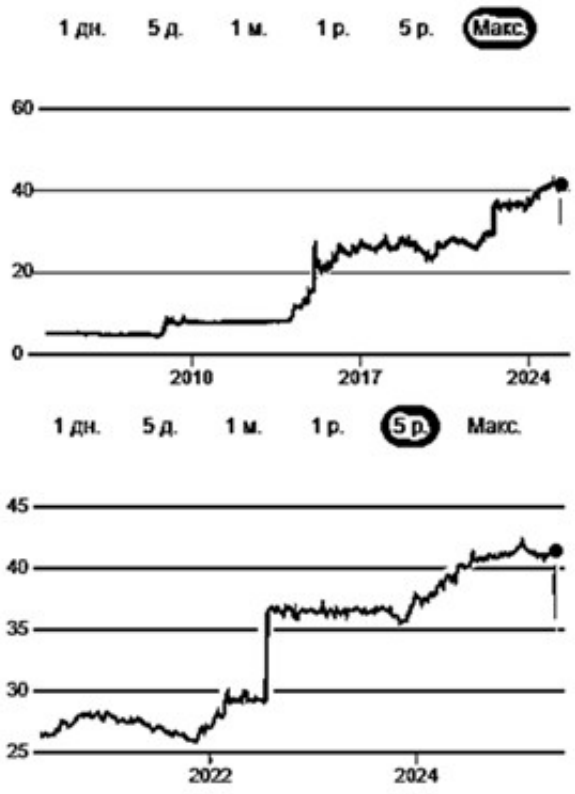


Рисунок 1.9 — Порівнняння графіків курсу валюти на різних проміжках

З рисунку вище, бачимо, що якби графіки не мали позначення часових шкал, то значення було б важко розрізнити. Тому існують методи аналізу обробки даних у часі, на основі фрактальних сигналів, які характеризуються

самоподібною поведіною в ріщних часових масштабах. Серед таких методів є функція фрактального аналізу, що полягає у визначенні того, які характеристики фракталів існують в даних сигналу, і згодом використовувати ці властивості для подальшого порівняння сишналу з іншими частинами. До прикладу частоту скорочень серця можна порівняти з— зі звуковими сигналами.

1.4 Аналоги програмного забезпечення з візуалізації фракталів

Оскільки сучасні обчислювальні потужності пристроїв значно зросли — з’явилось більше можливостей до виконання складних завдань. Тому, зараз існує багато застосунків [18], що дозволяють відображати складні структури як фрактали, зокрема й у вигляді тривимірних моделей, однак з обмеженими можливостями щодо цілковитої деталізації в реальному часі.

1.4.1 Використання в графічному мистецтві

З метою художнього використання, найпопулярнішим платним застосунком є Ulta Fractal. Це потужний комерційний фрактальний візуалізатор, орієнтований на художників та дизайнерів. Ulta Fractal дозволяє створювати складні фрактальні зображення з використанням шарів, масок та анімацій. Через зручний і гнучний інтерфейс (рисунок 1.10), даний застосунок дозволяє писати власні формули для генерації фракталів з використанням власного рушія. Також цей додаток дозволяє обробляти кольори генерованого фрактального зображення з використанням власних математичних формул, однак, для звичайний завдань його функціонал занадто великий.

Chaotica та JWildfire вважаються не менш популярними фрактальними редакторами для рендеру та створення анімацій з високою роздільною здатністю. Вони мають зручний схожий інтерфейс, показаний на рисунку 1.11 та 1.12 відповідно, а також можливість коригування кольорових каналів для створення різноманітних кольорових ефектів на основі фрактальної геометрії.

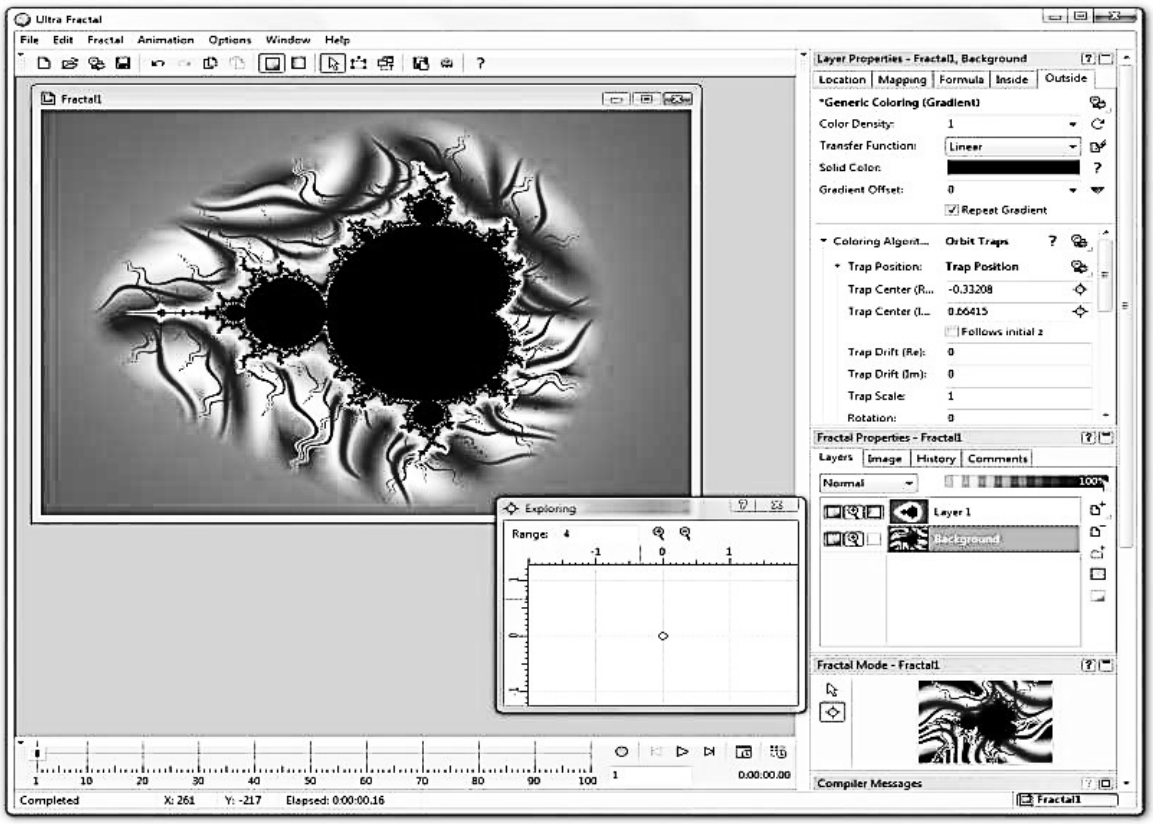


Рисунок 1.10 — Графічний інтерфейс користувача застосунку Ulta Fractal

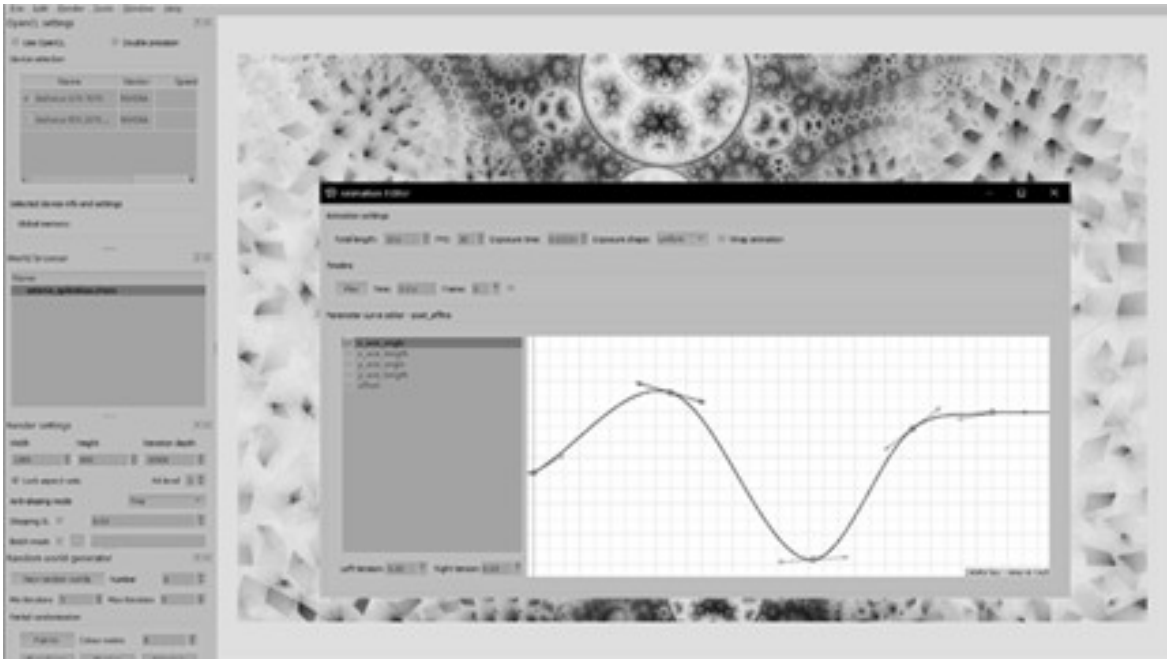


Рисунок 1.11 — Візуальне представлення програми Chaotica

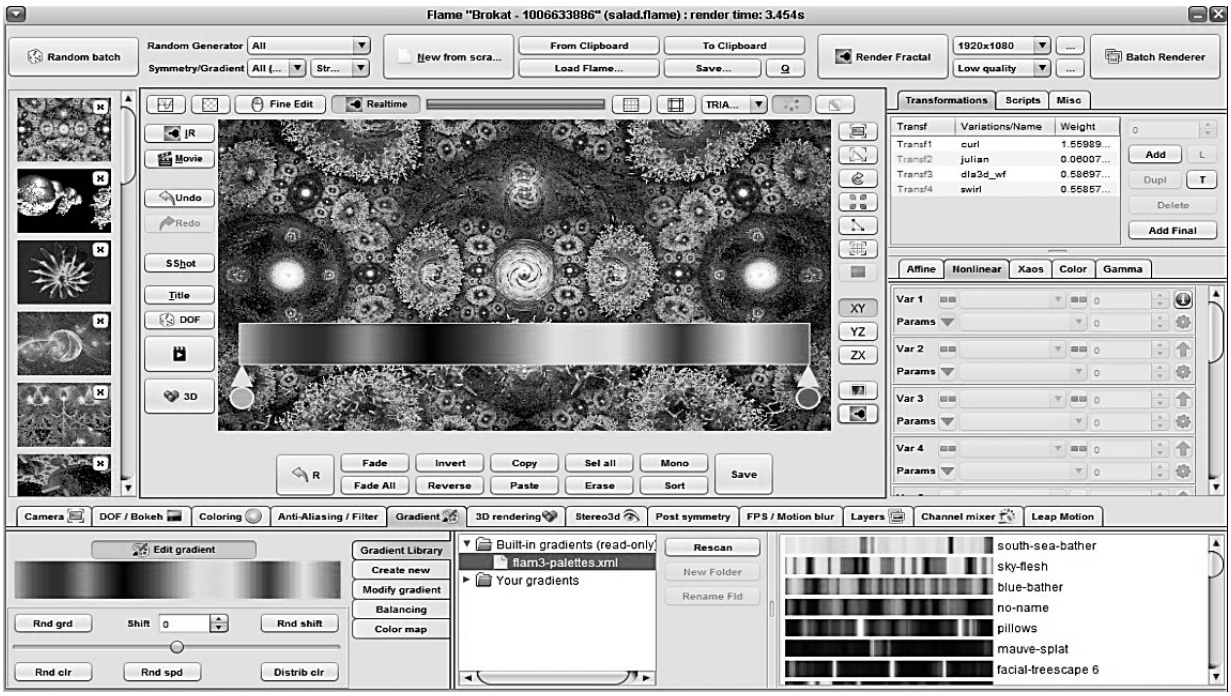


Рисунок 1.12 — Інтерефейс фрактального редактора JWildfire

Перевагою Jwildfire перед Chaotica є підтримка скриптів, які можуть автоматизувати процес налаштування та створення бажаних фракталів. Також, Jwildfire підтримує розширену систему параметричних модифікаторів, що дозволяє користувачам змінювати структуру двовимірних фракталів у реальному часі та легко експериментувати з складними комбінаціями ефектів.

1.4.2 Використання для наукових досліджень та математичної візуалізації

Візуалізація складних фрактальних структур також має важливу ролю в понятті визначенні самоподібностей та їх закономірностей у таких геометричних об’єктах. Тому існують специфічні застосунки, що дозволяють аналізувати складні структури математикам, фізикам та іншим дослідникам, зокрема й у напрямку обчислювальної графіки. Такими програмами є Fractint, Fragmentarium (рисунок 1.13) та Mandelbulb 3D.

Це не пересічні додатки, тому вони потребують наявності науково—математичного досвіду, аби коректно та в повній мірі користуватися ним. Насамперед через гнучку можливість редактувати зображення двовимірних та інколи тривимірних фракталів за своїми власними формулами.

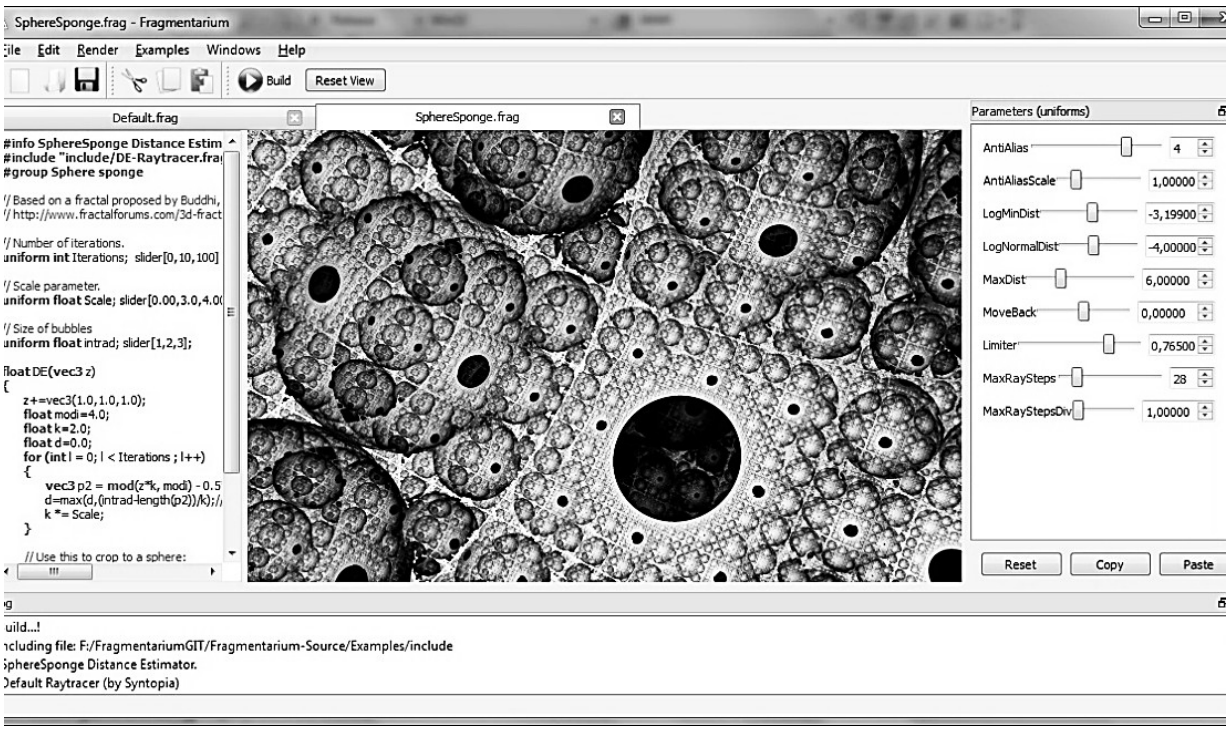


Рисунок 1.13 — Зовнішій вигляд інтерфейсу програми Fragmentarium

1.4.3 Використання для освітніх та розважальних цілей

Зважаючи на той факт, що фрактальна графіка поширена не лише в наукових напрямках, а й у розважальних та пізнавальних цілях, існують відповідні програми для цього. Найвідомішими застосунками даного напрямку є XaoS, Electric Sheep і Kalles Fraktale, які дозволяють через простий інтерфейс користувача, візуалізувати складні математичні об’єкти, що спрозує розуміння складних абстрактих термінів та понять.

До прикладу, програмне забезпечення XaoS (рисунок 1.14), головна задумка якого є найшвидше інтерактивне збільшення масштабу геометрії фракталів — дозволяє пересічному користувачу не знати відповідні формули чи теорію. Все що потрібно це керувати процесом позиції та навігації за допомогою відповідних клавіш у вже підготовлених наперед прикладах фракталів, а також вміти користуватись інтуїтивно зрозумілим інтерфейсом.

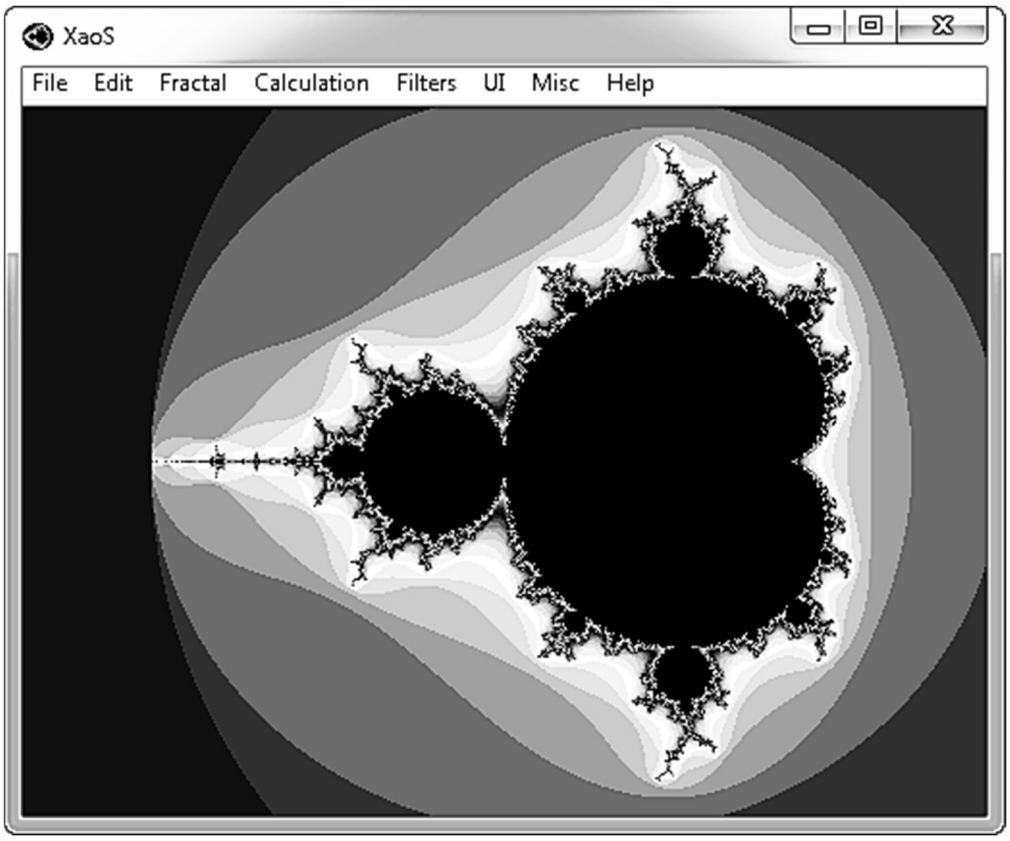


Рисунок 1.14 — Приклад вікна застосунку XaoS

1.4.4 Нелоліки існуючих аналогів програмного забезпечення

Фрактали, перш за все, не мають конкретної межі деталізації, тому їх візуалізація повинна мати чітку межу обрану програмно чи вручну, адже вона наближається до нескінченності. Більшість сучасних застосунків обчислюють деталізацію фрактальних структур із точністю до одного пікселя монітора, проте це займає дуже багато часу. Це підтверджує актуальність даної розробки, яка повинна дозволити скоротити час очікування через інших підхід до реалізації візуалізації та обмежень її точності, що дозволить створювати достатньо деталізовану візуалізацію, без значного використання системних ресурсів.

2 ОГЛЯД ТА ВИБІР ПРОГРАМНИХ ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ ФРАКТАЛІВ

Оскільки розробка запропонованого диплмною роботою, програмного забезпечення буде опиратись на оптимізацію з використанням сучасних засобів технологій, важливим є вибір основних технологій щодо його реалізації.

2.1 Рушії та основні мови програмування

Основою для кожного професійно— спеціалізованого програмного забезпечення повинен бути рушій, що надасть зручні інструменти для швидкої роботи з інтерфейсом, графікою, фізикою, та взаємодією скриптів. Оскільки розробка власного рушія це дуже трудомісткий процес, що потребує значних людських ресурсів та часу, для виконання бакалавравської кваліфікаційної роботи, було прийнято рішення вибрати найбільш оптимальний рушій серед існуючих у вільному доступі [19]. Також потрібно враховувати, що низький поріг до апаратних вимог та можливість графічної складової, граютимуть ключову роль для побудови обраного програмного забезпечення. Через потребу в графічній складовій та складної структури застосунку, найкращим вибором буде рушій із категорії ігрових.

2.1.1 Рушій CryEngine

Багатофункціональним та потужним варіантом серед огляду рушіїв є CryEngine (рисунок 2.1). Він дозволяє має досить складну, однак точну та якісну систему рендерингу графіки, включаючи підтримку глобального освітлення, на додачу до якої присутня велика бібліотека вбудованих ефектів.

Оскільки основною мовою програмування рушія є C++, він має досить високий поріг використання, зважаючи менш інтуїтивні інструменти, ніж у інших аналогів. Також CryEngine вимагає великі системні ресурси для роботи й тестування застосунків, що робить його непрактичним вибором для розробки

полегшених наукових застосунків, орієнтованих на потужності повсякденних персональних комп’ютерів.

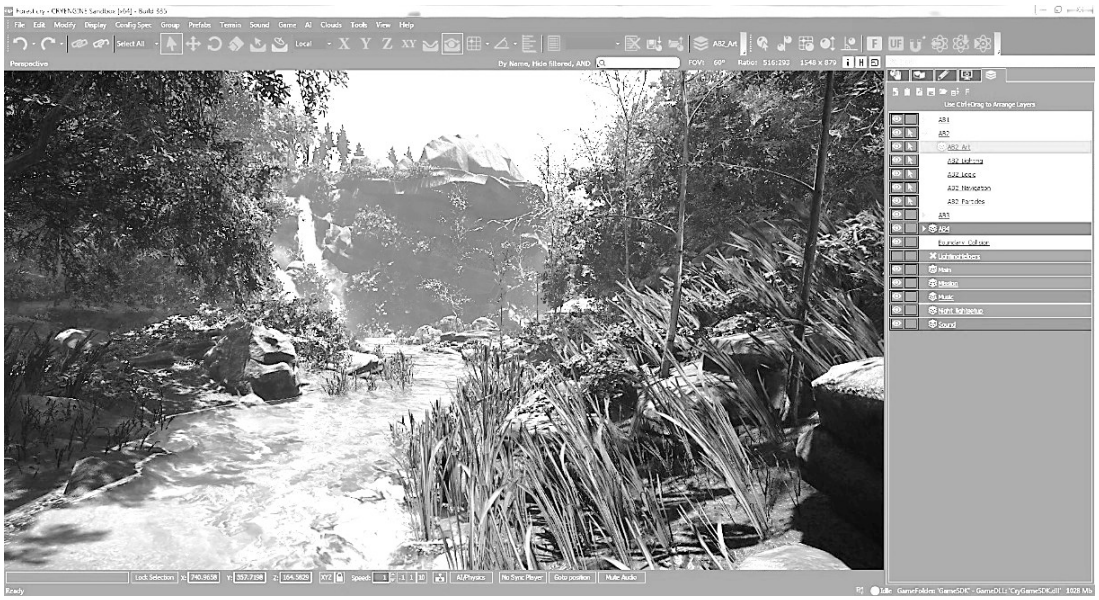


Рисунок 2.1 — Зовнішній вигляд редактора рушія CryEngine

2.1.2 Рушій Godot Engine

Інтуїтивно зрозумілий для початківців у сфері розробки програмного забезпечення, є безкоштовний рушій з відкритим кодом Godot Engine. Його основна перевага полягає в модульності й простоті інтерфейсу та побудові застосунків (рисунок 2.2). Даний рушій має власну мову для написання скриптів на основі Python і має назву GDScript.

Можливість підтримки C# дозволяє рушію створювати продуктивні компоненти для вирішення складних ресурсомістких задач. Також Godot Engine має підтримку 2D та 3D сцен, що надає ширший вибір під час розробки поставленого в роботі програмного забезпечення та його функціоналу.

Значна оптимізація робить рушій придатним для малих за обсягами застосунків, зокрема орієнтованих на мобільні телефони або ж просто для експериментів. Основним недоліком даного рушія є погано реалізована тривимірна графіка в порівнянні з іншими, а також фізична відсутність підтримки обчислювальних шейдерів з іншими засобами прямої роботи з

графічним процесором. Такі недоліки є критичними для розробки застосунків з графічної візуалізації складних фракталів, тому Godot Engine — є поганим вибором для поставленої роботою задачі.

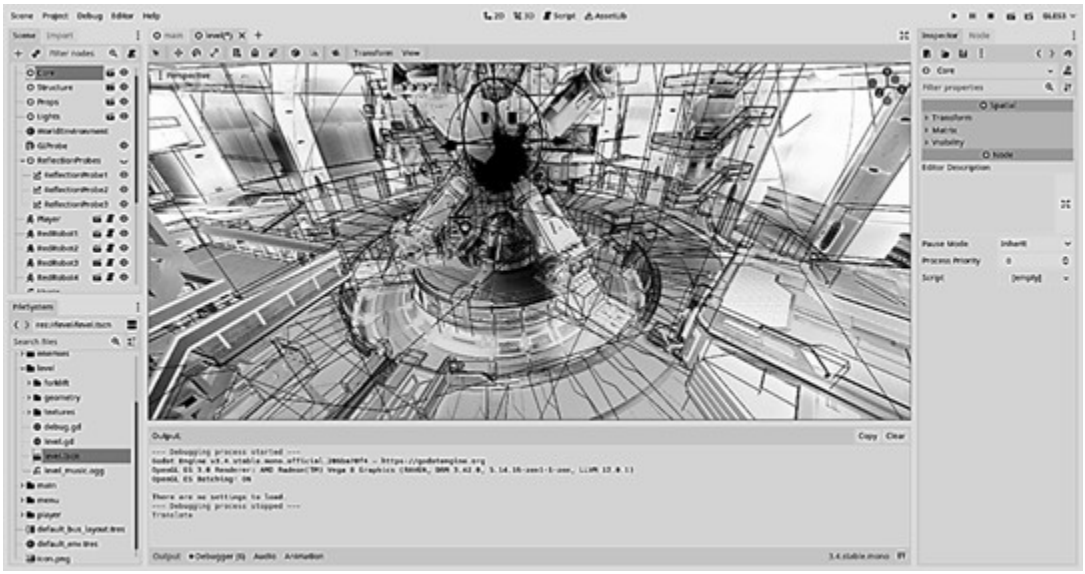


Рисунок 2.2 — Візуальне представлення вікна редактора Godot Engine

2.1.3 Ігровий рушій Unreal Engine

Фундаментром стандарту фотореалістичних застосунків сьогоднення, вважається Unreal Engine (рисунок 2.3). Він має потужні інструменти рендерингу, точну симуляцію фізики та одну з найкращих продуктивностей масштабних сцен застосунків.

Рушій містить новітні технології візуалізації геометрії як Nanite, а також глобального освітлення в реальному часі як Lumen. Такі аспекти дозволяють створювати масштабні реалістичні сцени. Основною мовою програмування вважається C++, хоча значну частину побудови застосунків рушія, проводять з використанням візуальної мови вузлів Blueprints, що дозволяє мінімізувати використання коду під час побудови логіки застосунку.

Незважаючи на продвинуті інструменти створення й конфігурування гріфчної складової застосунків, Unreal Engine не придатний для слабких персональних комп’ютерів. Для побудови оптимізованого додатка з

використанням даного рушія, потрібне глибоке технічне тестування та розуміння середовища, котре все одно навіть під час перебування в редакторі — споживає дуже багато ресурсів апаратних засобів. Перевага Unreal Engine полягає в можливості підтримувати масштабні сцени в застосунках, однак для побудови простих додатків з мінімальною кількістю об’єктів на сцені, цей рушій є дуже неефективним рішенням.

Рисунок 2.3 — Редактор сцени Unreal Engine

2.1.4 Рушій Unity

Одним із найпопулярніших рушіїв сучасності є Unity, оскільки він пропонує широкий спект можливостей створення застосунків з відповідним попереднім налаштуванням до типу побудови, що дозволить наприклад не використовувати тривимірну складову в двовимірних застосунках, оскільки це не потрібно. Unity тримає високу планку перш за все через модульність компонентних складових під час створення власного застосунку, а також через повний контроль над структурою власної сцени, логікою взаємодіїї її об’єктів та подій (рисунок 2.4).

Серед значних переваг, даний рушій підтримує як 2D, так і 3D графіку, а також має вбудовану систему фізики, анімацій, ефектів частинок, освітлення та

керування камери огляду. Досить значимим є підримка мов ShaderLab та HLSL, що дозволяють писати власні шейдери, котрі безпосередньо працюватимуть на рівні графічного процесора комп’ютера. Звідси, усі складні математичні обчислення можна переназначати з центрального процесора для розвантаження системи, на графічний процесор, після чого оперуючи кількістю обробки даних аби додаток продовжував функціонувати під час великих навантажень. Особливо, враховуючи той факт, що візуалізація фракталів у режимі реального часу потребуватиме багато однотипних обчислень кожен часовий такт програми.

Рисунок 2.4 — Зовнішній вигляд редактора Unity

Оскільки Unity має основною мовою програмування C# — це гарантує значну продуктивність то зручність підтримки проєкту під час розробки. Зокрема, можливість обирати між різними графічними системами обчислень (URP та HDRP) для пристроїв з низькою та великою продуктивністю, робить цей рушій універсальним для більшості сучасних персональних комп’ютерів.

Зважаючи на наведені вище переваги, а також на велику спільноту, що налічує документації, плагіни та розширення — рушій Unity є найкращим

вибором для реалізації, поставленого індивідуальним завданням, програмного забезпечення.

2.1.5 Вибір основної мови програмування

Не менш важливим під час створення власного програмного забезпечення є вибір основної мови програмування, за допомогою якої буде налаштовуватись взаємодія компонентів програми для цілісної роботи під час використання. Також обрана мова, на пряму буде впливати на ефективність розробки застосунку, зручність його подальшої підтримки та продуктивність роботи на пристроях. Однак варто зауважити, що ефективність обраної мови проявляється лише в тому випадку, коли вона використовується за своїм призначенням [20].

Серед найбільш популярних мов програмування сьогодення, можна виділити C++, Python, TypeScript, Rust та C#. Кращим вибором для системного програмного забезпечення без сумініву є C++, вона ж зазвичай використовується для важких проєктів як от ігор класу «ААА». Ця мова є компіляторною мовою, і поєднує процедурний (поділ коду на логічно завершені блоки у вигляді функцій) та об’єктно— орієнтований підходи. Важливим недоліком C++ є відстутність вбудованого автоматичного збору сміття, тому програмісту потрібно власноруч керувати пам’яттю, а би досягти найкращої продуктивності, ця особливість підвищує можливість появи критичних помилок як витоки пам’ятті. Однак оскільки дана мова є складною в освоєнні, а також у подальшій підтримці, даний вибір не підходить для виконання застосунку з візуалізації фракталів у тривимірному просторі.

Широко використовувана під час роботи із шутчним інтелектом та науці мова програмування Python є інтерпретованою мовою високого рівня. Ця мова має автоматичну збірку сміття, що дозволяє пришвидшити процес розробки. Також, оскільки Python виконується інтерпритатором, а не компілюється в машинний код — він значно повільніший за вищезгаданий C++, що робить його непридатним до активного графічного відображення в додатках без участі проміжних доповнень.

TypeScript є надбудовою JavaScript з додаванням статичної типізації, тому він успадковує характеристики js, як інтерпретованість мови. Він має автоматичний збір сміття, та часто використовується для побудови веб— додатків чи розробки веб— інтерфейсів. Оскільки дана мова має власні обмеження доступу до системних ресурсів як от оперативної пам’ятті чи відеокарти, вона не підходить для виконання завдань, котрі потребують високої обчислювальної потужності.

Сучасною компіляторною мовою системного програвування, що містить безпечне управління пам’ятті без використання збірника сміття, вважається Rust. Ця мова компілюється у високопродуктивний машинний код з використанянм власного компілятора rustc, а також гарантує відсутність виктоків пам’яті та подвійного звільнення ресурсів під час компіляції. Вона має продуктивність C++, однак із забезпеченням кращої безпеки, тому її використовують під час написання криптовалютних клієнтів. Проблемою Rust є недостатня інтеграція мови на графічні рушії, тому її підтримка в розробці тривимірних додатків досить обмежена.

Найбільш популярною, та відносно новою компіляторною мовою програмування є C#, яка поєднує об’єктно— орієнтований та функціональний підходи. Також, дана мова програмування, на відміну від, наприклад, Rust, має автоматичне управління пам’ятті за з використанням середовища виконання CLR, яке містить вбудований збір сміття, та дозволяє зосередитися безпосередньо на розробці, а не на керуванні пам’яттю. C# має жорстку типізацію типів даних, через що на етапі кмпіляції легше виявити помилки, що в свою чергу покращить надійність коду. Для розробників доступна бібліотека .NET, котра надає доступ до тисяч класів для специфічних задач як робота з файлами, інтерфейсом, багатопотоковістю чи мережею. Оскільки вона була створена компанією Microsoft, вона постійно розвивається й отримує покращення, а також має стабільну підтримку та спільноту. Перевагами C# також можна виділити кросплатформенність, підтримку середовищем розробки Visual Studio, а також

офіційно визнаною мовою програмування для рушія Unity, який був обраним для побудови бажаного програмного забезпечення.

2.2 Аналіз шейдерів HLSL

Аби створити менш вибагливий метод імітації, потрібно зрозуміти головні аспекти скриптових шейдерів, для прикладу розглянемо HLSL шейдери, обраного рушія Unity. Загалом, вони дозволяють визначити те, як на екрані відображатимуться об’єкти, за рахунок обробки певних взідних даних, та розрахунку результатів у вигляді формування кольорів чи складних ефектів як деформація чи відзеркалля відображення.

2.2.1 Типи шейдерів

Шейдери у Unity, по суті є спеціалізованими програмами, які виконуються на графічному процесорі пристрою, а також створюються мовою HLSL. Через щільний зв’язок шейдеру з відеокартою, який має назву — графічний конвеєр, потрібно оглянути основні етапи обробки цієї структурованої послідовності, щоб потім можна було реалізувати відповідний метод реалізації.

Усі шейдери написані HLSL, мають зазвичай поділ на два основних типи:

— вершинний шейдер, який обробляє всі вершини моделі, з можливістю їх зміни чи зчитування положення;

— фрагментний шейдер, котрий працює на рівні пікселів і визначає їх остаточний колір.

Ці два типи тісно пов’язані в програмі шейдера, оскільки текстурована модель, проходить увесь етап взаємодії під час якого зчитуються усі наявні вершини, отримані та індексовані дані передаються до вершинного шейдеру для обробки, після чого відбувається модифікація геометрії об’єктів (якщо така прописана), результат обчисленої геометрії перетворюється у пікселі на екрані (растеризація), після чого отримані дані геометрії прив’язаної до конкретних пікселів передаються до фрагментного шейдеру, який в свою чергу застосовує відповідні кольори, а останнім етапом є фінальна обробка пікселів перед

відображенням на екрані. До фінальної обробки належать процеси тесту глибини, альфа змішування, колекція кольору чи яскравості, котрі вже не пов’язані з фрагментним шейдером напряму, однак є частиною графічного конвеєра. Описані вище етапи графічного конвеєра шейдерів Unity HLSL, наведено у вигляді ілюстрації порядку взаємодій на рисунку 2.5.

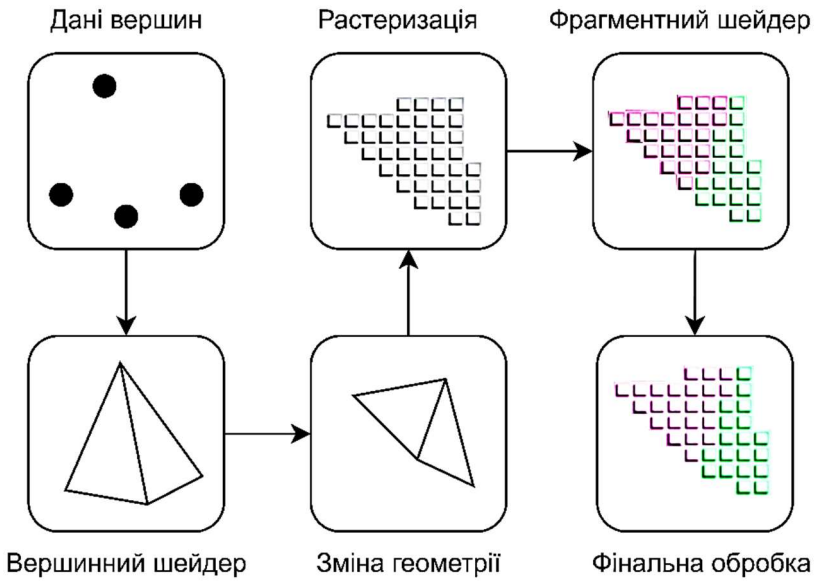


Рисунок 2.5 — Основні етапи графічного конвеєра шейдеру HLSL у Unity

Дії над вершинами, у вершинному шейдері, є досить трудомісткими, тому потрібно обмежитись використанням фрагментного шейдера. Цей шейдер виконується для кожного пікселя моделі окремо, й відопвідає в результаті лише за накладений колір. Однак, щоб зрозуміти де саме накладається піксель на модель, потрібно пояснити текстуринг моделей.

2.2.2 UV— розгортка

Кожна вершина моделі, містить призначення до UV кординат, котрі визначаються вручну під час моделювання або ж процесу UV— розгортання. По суті, UV це відповідність двовимірним кординатам накладуваної на модель текстури до її кординат поверхні, де U — це горизонтальна вісь текстури, а V — вертикальна вісь. Кожен піксель отримує нормалізовані значення відносно самої

текстури, котрі позначатимуть позицію. Яскравим поясненням цього, можуть слугувати вузли інструменту Shader Graphs від Unity (рисунок 2.6), де UV візуалізовано як нормалізовану текстуру де кординати абсцис та ординат відповідають за відповідний нормалізований відтінок червоного та зеленого в системі RGB. Однак відокремивши значення позицій, та об’єднавши їх вузлом Multiple, можемо побачити загальний вигляд розташування кординатних значень UV.

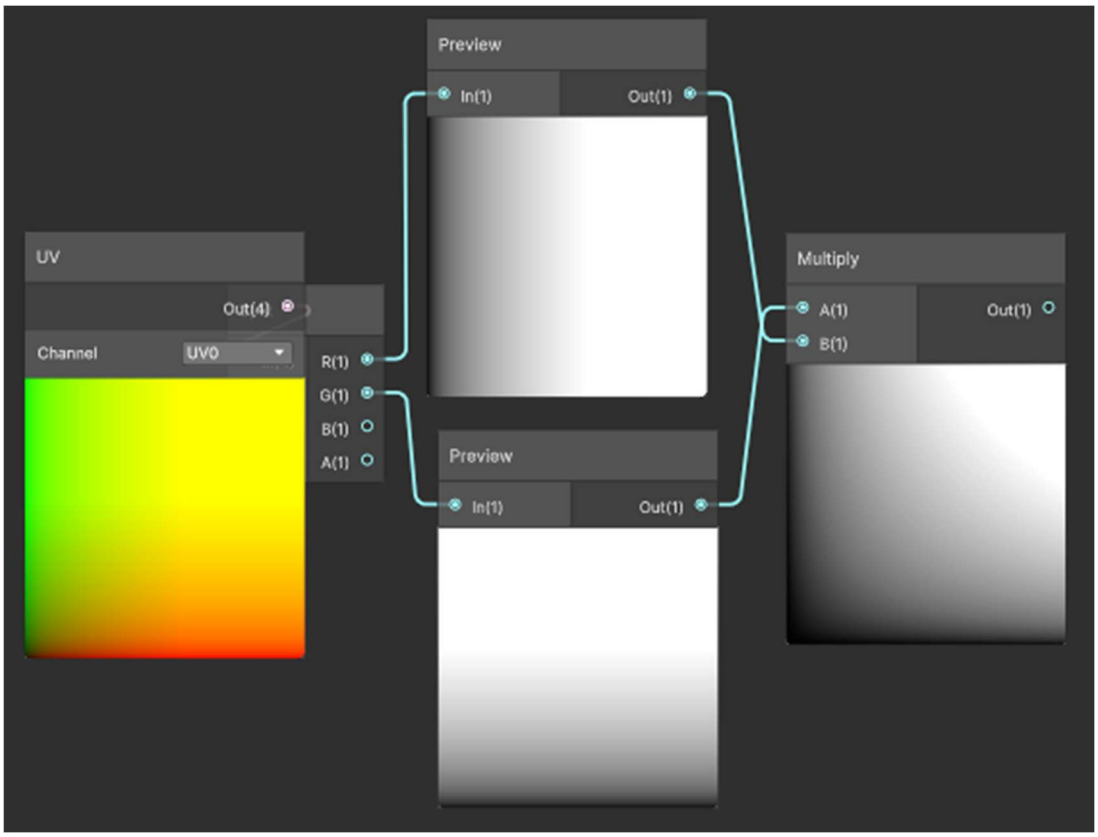


Рисунок 2.6 — Візуалізація осей звичайної UV розгортки

Хоча ці кординати нормалізовані, їх числове представлення дозволяє нам виконувати відповідні дії, які в результаті змінюватимуться. Тобо віднявши або додаши певні значення до осей UV, ми фактично зсунемо двовимірні кординати у відповідному напрямку. Аналогічно цьому, дії ділення та множення — будуть збільшувати та зменшувати масштаб накладання текстури на модель (рисунок 2.7).

Знаючи можливості оперування UV, ми можемо створювати складні результати накладання фінальної текстури на модель без значних затрат продуктивності.

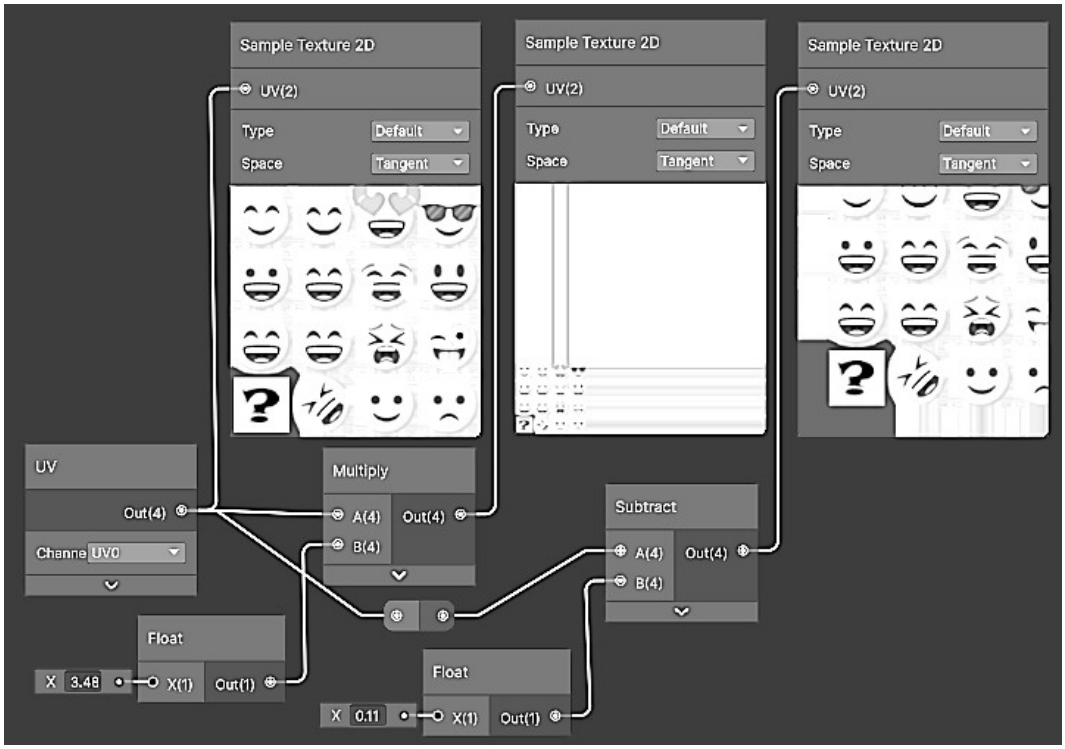


Рисунок 2.7 — Приклад простих операцій над UV кординатами

2.2.3 Матеріали шейдерів

Усі написані моою HLSL шейдери, реалізують візуалізацію тієї чи іншої моделі, зокрема попри колір, він може обчислювати тіні, відблиски, гладкість поверхні, динамічність візерунку, форми моделі та їх динамічної зміни абощо. Зважаючи на таку велику кількість різномантіних параметрів характеристики візуалізації поверхні моделі, результат обчислень шейдера має тип не текстури, а своєрідного кластера під назвою матеріал.

Ігровий рушій Unity, підтримує систему матеріалів, яка фактично визначає вигляд поверхні моделі, що також враховує і текстуру. Всього матеріали можуть бути таких типів:

— звичайного типу з базовими налаштуванням властивостей до візуального сприйняття освітлення, та монотонним забарвленням;

— текстурні, що являють собою зображення, котре накладається з використанням UV— розгортки;

— шейдер, який може створювати власні текстури та способи їх візуальних представлень на моделі.

Незважаючи на офіційний поділ матеріалів, усі ці типи є прикладами саме шейдерів, які в свою чергу і є програмами які визначатимуть ту чи іншу поведінку матеріалу в просторі та на моделі.

Тому, маючи UV— розгортку певної моделі, ми можемо присвоїти відповідним її полігонам матеріал шейдера, який в свою чергу може змінювати ряд візуальних характеристик і форм обраного об’єкта

2.3 Способи обчислення та візуалізації фракталів

Оскільки всі фрактали мають різну природу походження та побудови, спосособи їх обчислення та візуалізації відрізняються. Наприклад, певні типи базуються на простих геометричних перетворення, а інші на властивостях комплексних чисел.

2.2.1 Ітеративні функціональні системи

Метод обчислень фракталів з використанням системи ітераційних функцій дозволяє створбвати самоподібну структуру об’єкта, через його багаторазове перетворення математичними функціями. Основним принципом такого методу є ітерація, під час якої повторно застосовуються функції до початкового стану об’єкта, після чого формується аналогія геометричного фракталу. Зазвичай функції в таких системах, створюють самоподібні частини об’єкта, що кратні ітераціям збільшують свою кількість та зменшуються у загальному масштабі аби можна було побачити загальний прогрес, а також додають певне перетворення, як наприклад оберт на певний кут. Прикладом виконання такого підходу до реалізації фракталів може бути Папороть Бернслі (рисунок 2.8).

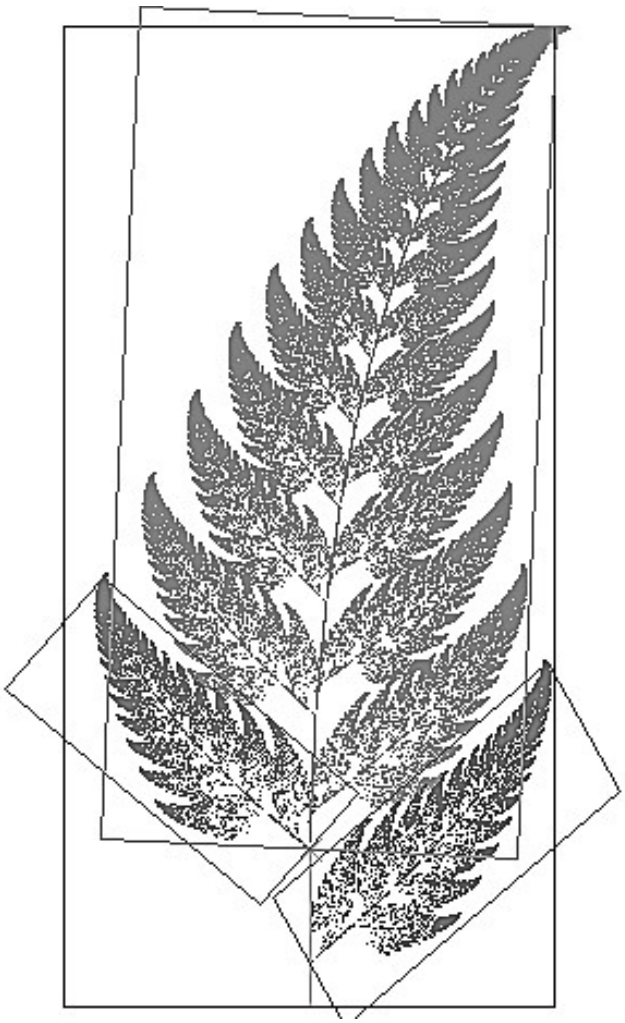


Рисунок 2.8 — Вигляд фракталу Папороть Бернслі

2.2.2 Рекурсивне відтворення

Даний спосіб обчислення базується на відтворенні фігури за певним шаблоном. Інакше кажучи, почавши з простої фігури, всередині неї, ми малюємо точну, однак зменшену копію самої себе, після чого цей процес повторюється бажану кількість разів. Попри ускладнення фігури з кожною ітерацією обчислення, загальна конструкція зберігає початкову форму.

Серед яскравих прикладів використання такого підходу, можна виділити реалізацію обчислень фракталу Трикутник Серпінського (рисунок 2.9), який базується на побудові рівностороннього трикутника, після чого, в кожній ітерації обчислення додаючи в невикористаний простір якого вдвічі менші за розміром рівносторонні трикутники.

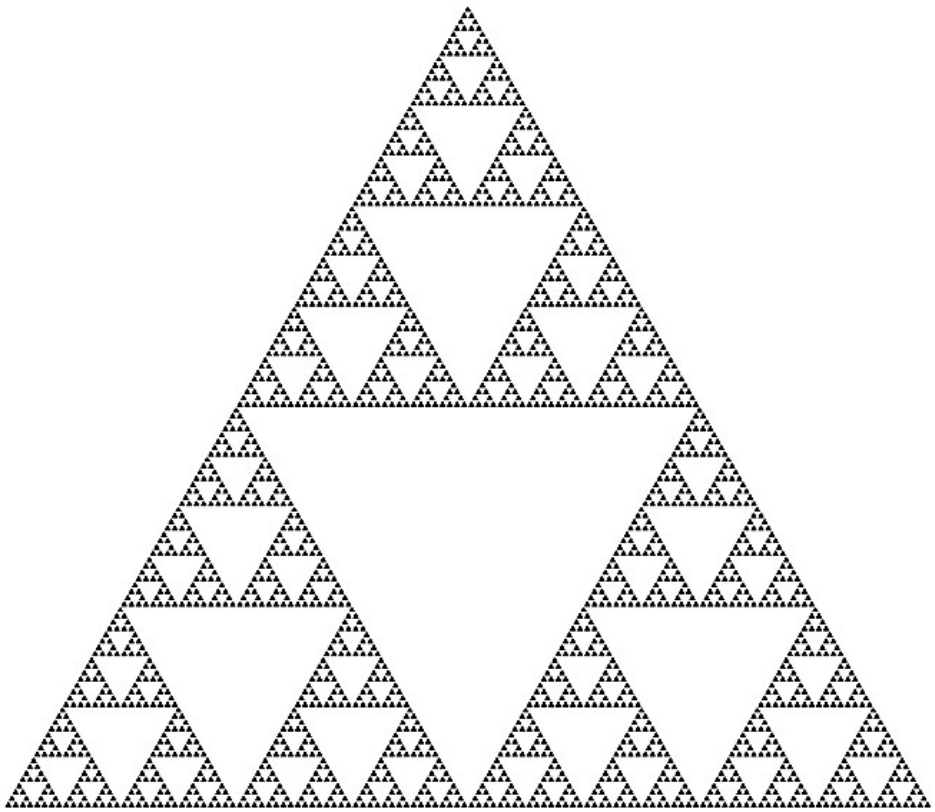


Рисунок 2.9 — Вигляд фракталу Трикутника Серпінського

2.2.3 Використання властивостей комплексних чисел

Фрактали з роду Множини Джулія та Множини Мандельброта — побудовані на властивостях комплексних числе. Їх рисунки визначаються за рахунок дії над комплексними числами, а саме над кожним відповідним число точки площини. Зазвичай формула в такий випадках має загальні схожості: обрати число з площини, піднести до степеня квадрату й додати інше число. Даний процес розрахунків продовжується доти, доки значення чисел не почне різко рости, що позначатиме нестабільність точки, а отже неналежність її до множини фракталу. Для відповідного забарвлення фракталів аби побачити залежності від обчислень в певних його точках на кожній ітерації, в залежності від ітерацій враховується колір точки. Для прикладу показано росфарбування таким способом Набору Джулії на рисуноку 2.10.

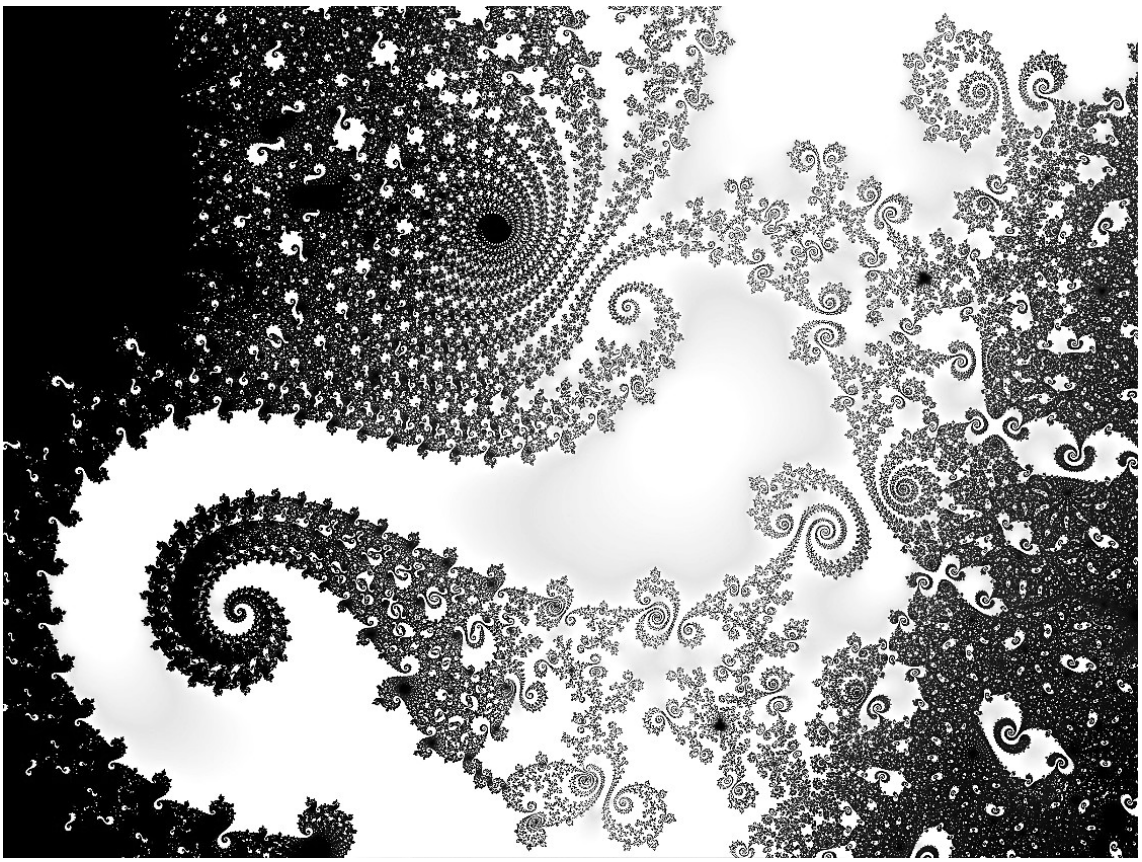


Рисунок 2.10 — Зовнішній вигляд візуалізації Набору Джулії з кольоровим фарбуванням в залежності від кількості ітерацій

Головна ж властивість комплексних чисел, що дозволяє створювати такі дивовижні структури (див. рисунок 2.10) полягає в тому, що під час піднесення його до квадрату, ми отримуємо не просту зміну числа, а складну динаміку, згідно якої, точка обертається навколо центрру, виходить далеко за межі видимого діапазону, або ж залишається незмінною в одному місці. Це можна охарактеризувана як змінну точки за відстанню та напрямком під час кожної ітерації.

2.2.4 L–системи

Такий підхід обчислення фракталів, має на меті покрокову рослиноподібну побудову знідно наперед визначених простих правил, котрі повторюються й доповнюють самі себе, власними копіями на кожній ітерації. На відміну від інших існуючих методів, де основа обчислень спирається на результати формул,

даний підхід зазвичай працює з рядками символів, які згодом програмно розпізнаються як інструкції щодо малювання.

До прикладу, в рамках кваліфікаційної роботи, було реалізовано Фрактал Слова Фібоначчі, що базується на такому типі побудови (рисунок 2.11).

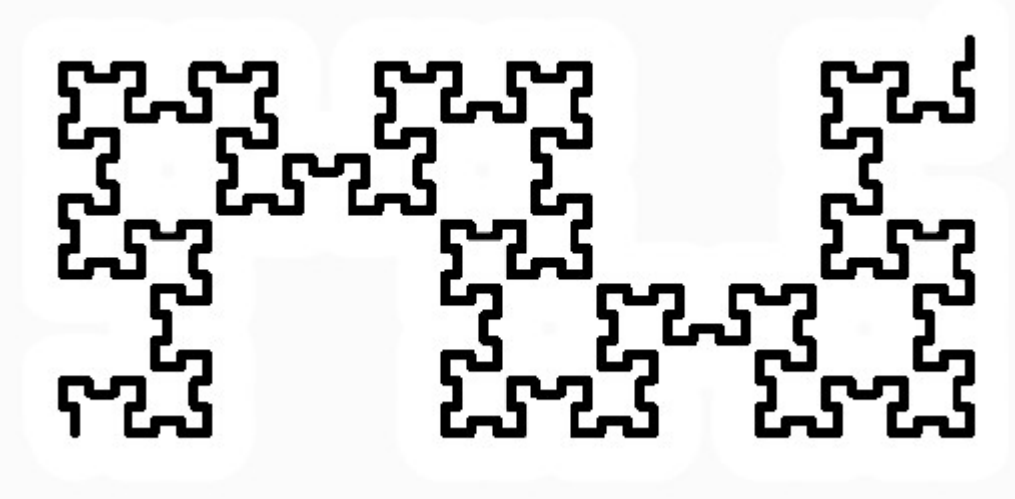


Рисунок 2.11 — Шейдерна візуалізація фрактала Слова Фібоначчі

Для початку побудови такого фракталу, було обчислено символьну послідовність інструкцій, котра базувалась на двійковому слові Фібоначчі. Перше слово має значення «0», друге «01», третє слово формується через дописування попереднього слова перед останньо визначеним. Слідуючі слова з кожною ітерацією обчислюються за відповідним принципом, після чого збережене з певною кількістю ітерацій слово зберігаєтсья як майбутня інструкція. Потім, у відповідному програмному забезпеченні, наприклад в шейдерному доповненні, зчитуючи попорядку символи зі слова Фібоначчі малюємо лінію (сегмент) наперед визначеної довжини за такою ісінтрукцією:

— якщо символ рівний «1», то відмалльовуємо прямий відрізок;

— якщо символ рівний «0» і його індекс непарний, то обертаємо відрізок на 90 градусів вправо, і малюємо його з початку кінця попередньо намальованого;

— якщо символ 0 і його індекс позиції парний, то повертаємо відрізок на 90 градусів вліво, і малюємо його з початку кінця попередньо намальованого.

Основною проблематикою подібних обчислень фракталів, є потреба в збереженні попередніх результатів та попередньо визначений обчислений порядок інструкцій, що робить такий підхід майже неможливим до реалізації в шейдерах орієнтованих на роботу графічного процесора, оскільки доступна пам’ять буде шивдко вичерпуватись.

2.4 Техніка обчислення шляху променів

Для виконання основного завдання з візуалізації фракталів, використовувалась технологія комп’ютерної графіки під назвою Raymarching. Основна ідея в тому аби відтворювати тривимірні сцени, без використання полігональних моделей, а з використанням умовного променю проведеного від центру позиції камери в напрямку погляду до конкретного пікселя екрану. В порівнянні з вибагливим до системних ресурсів raytrasing методом, що одразу прямує до поверхні враховуючи негайне зіткнення, то raymarching використовує техніку крокування:

— кожен промінь, що було запущено, має мінімально обмежену довжину кроку;

— під час кожного з крокування променя, перевіряється за допомогою функцій відстані чи знаходиться перетин променя всередині об’єкта;

— за умови відсутності перетину, промінь продовжується на додаткові кроки;

— обчислення випускання променів з кожного пікселя продовжується доти, доки промінь умовно не перетне об’єкт або не вийде за встановлений максимальний діапазон кроків.

Схематичне зображеня роботи Raymarching показано на рисунку 2.12.

Для перевірки так званого «перетину» з об’єктом, використовується поняття функцій відстані, які дозволяють для відповідної кожної точки простору вказати на скільки далеко від неї знаходиться поверхня. Такі функції видають числовий результат, який враховує три стани: додатній результат вказує на розташування поза об’єктом, нульовий — точне розташування на поверхні а

від’ємний на розташування всередині об’єкта. Інакше кажучи, якщо ми до прикладу візьмемо стандартне рівняння еліпса $x^2/a^2 + y^2/b^2 = 1$, та змінимо знак рівності на знак менше, то отримаємо функцію відстанні, де від’ємні значення будуть належати еліпсу, а додатні — ні.

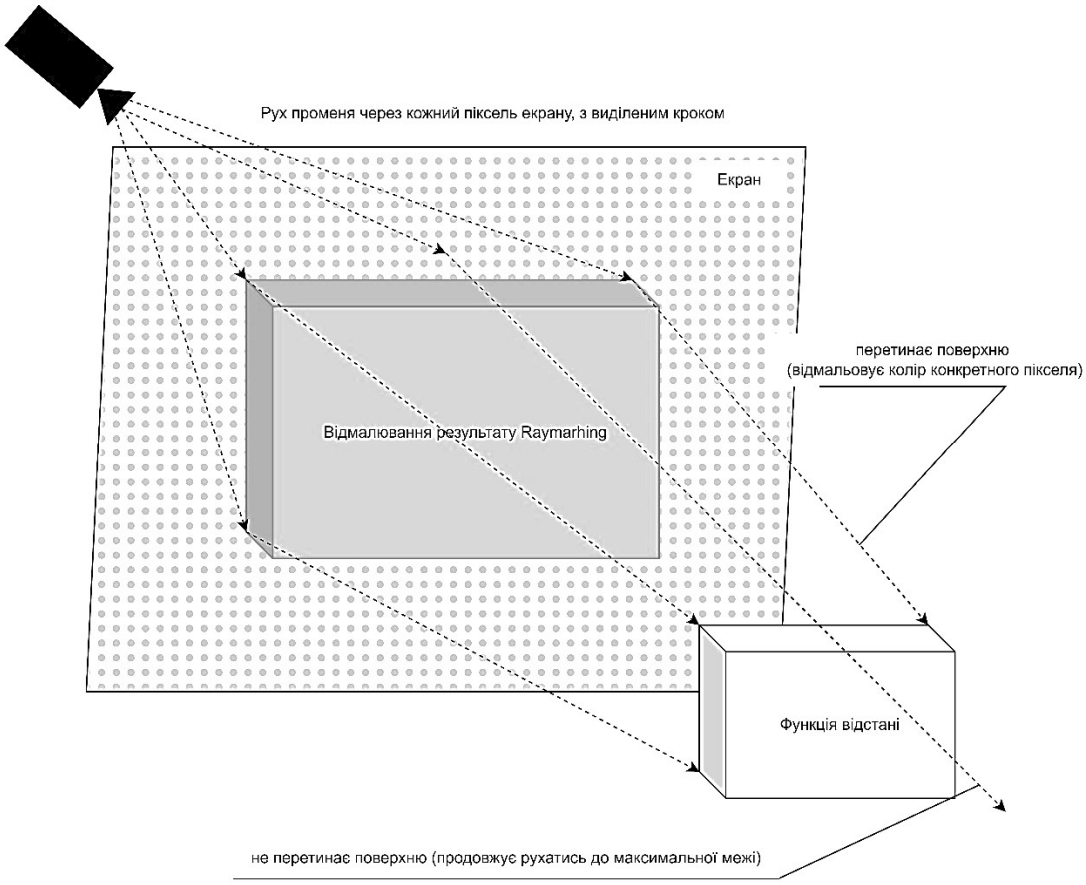


Рисунок 2.12 — Схематичне зображення роботи Raymarching підходу

Аби показово відобразити цю залежність, скористаємось мовою шейдерів рушія Unity під назвою ShaderLab з використанням її візуального представлення Shader Graphs (рисунок 2.13). Аналізуючи наведений рисунок бачимо, що площину декартової системи кординат відіграє вузол Screen Position, розділивши кординати кожної з його точок на осі ординат та обсцис ми застосували рівняння еліпса, де значення ширини «a» приймає число 0.46, а значення висоти «b» — 0.96. Після чого кординати та розміри було піднесено до квадрат за допомогою вузлів multiple помноживши їх самих на себе, і виконавши попарне ділення, результат якого об’єднати вузлом Add. Після чого отриманий результат у вигляді

графічного представлення усіх кольорів як занчення відстанні, можна вважати функцією відстанні, де значення від 0 до 1 приймають колір від чорного до білого відповідно, а все що переважає цей результат, колір встановленої межі. Тому, для того, аби побачити наш фактичний еліпс, скористаємося вузлом Replace Color де всі від'ємні та нулеві значення, що приймають чорний колір, виділимо яскраво червоним.

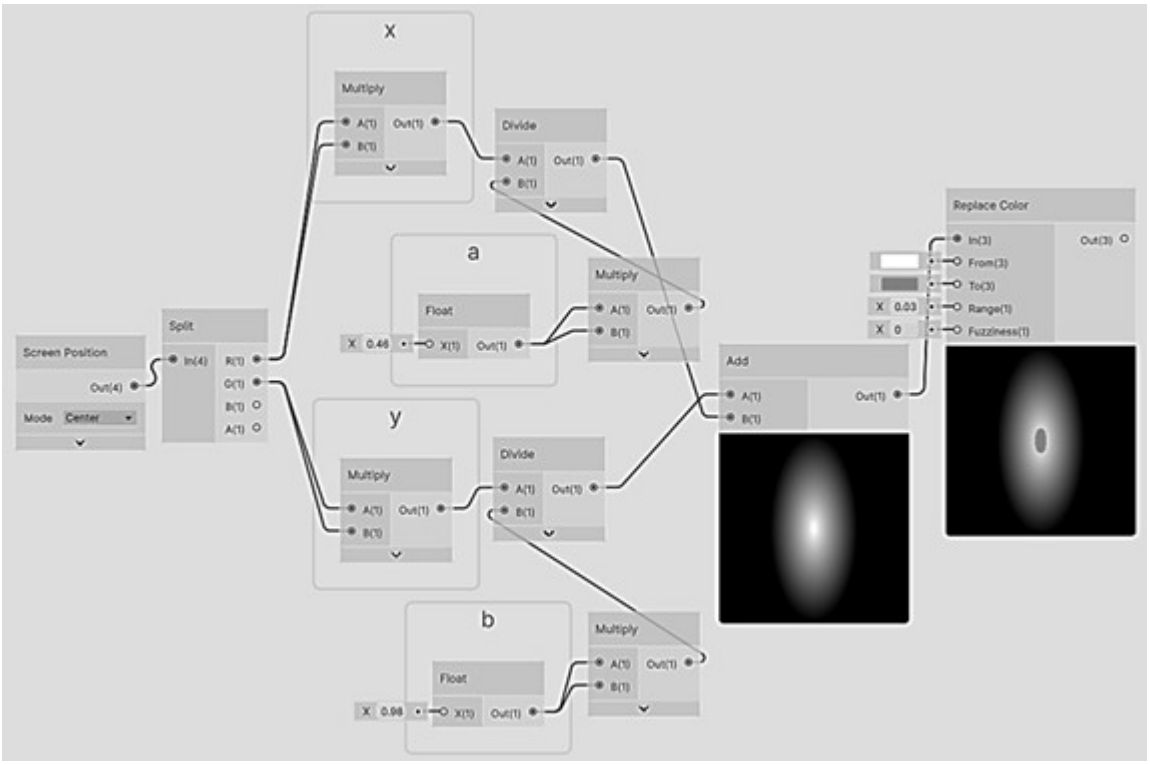


Рисунок 2.13 — Демонстрація простої функції відстанні еліпса з використанням Shader Graphs інструменту

Використання Raymarching відкриває ряд переваг в реалізації поставленого роботою завданні, а саме процедурність графіки, що дозволятиме генерувати нескінченно детальні фрактальні структури згідно встановлених обмежень, компатність написання коду, що дозволить обмежити увесь процес визначення об'єктів математичними формулами, а не фактичною геометрією, а також можливість створення складних форм через використання операцій об'єднання, перетину та віднімання (рисунок 2.14).

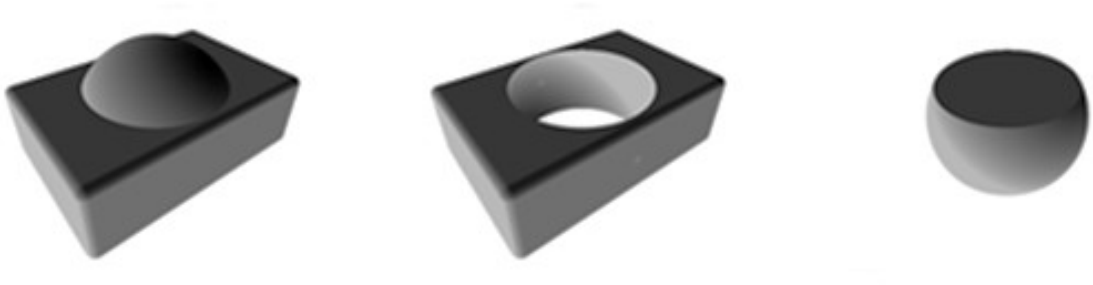


Рисунок 2.14 — Візуальнгий приклад операцій над сферею та зкругленим палалелепіпедом

Як згадувалось вище, навіть для нескінченної генерації, варто встановлювати обмеження, зокрема на обчислювальні затрати для більш складних та деталізованих фракталів. Також, варто зазначити що в тривимірному просторі майже не можливо ідеально визначити перетин з поверхнею за допомогою Raymarching, оскільки завжди буде певна мінімальна похибка, яку бажано встановлювати вручну, аби не отримати певних неочікуваних результатів при діях з нулевим значенням поверхні. Також недоліком є те, що моделі створені технологією Raymarching набагато важче анімувати ніж звичайні полігональні моделі, зокрема тому, що візуалізовані об’єкти представляють собою результати виконання наперед відомих функцій, що зазвичай залежать повністю від кординат простору, та певних параметрів фігури, тому, якщо ми хочемо застосувати певні зміни для таких об’єктів, потрібно змінювати саме початкові дані як от систему кординат у тривимірному просторі.

Оскільки даний метод візуалізації найкраще підходить для сторення тривимірного показу фракталів, зокрема й на посередніх за кількістю обчислювальної потужності, персональних комп’ютерах, під час виконання дипломної роботи, було обрано саме його.

3 РОЗРОБКА МЕТОДУ ІМІТАЦІЇ ТРИВИМІРНИХ ФРАКТАЛІВ

Основою продуктивності під час побудови програмного забезпечення, є реалізація методу візуалізації тривимірних фракталів. Оскільки не має єдиного достовірно коректного варіанту, а фактична візуалізація в тривимірному просторі вимагає багато системних ресурсів, було розроблено найбільш продуктивний метод імітаці показу тривимірної візуалізації.

Вагомою часткою методу є використання шейдеру, який обчислює та відмальовує фрактальні структури з використанням підходу Raymarching. Основним поняттям обраного графічного підходу є відстань, кожен об’єкт це представлення відстанні, а оскільки більшість існуючих фракталів мають лише двовимірну візуалізацію, спочаку було реалізовано перетворення із звичайних чітко визначених меж фрактальних структур, до представлення їх відстанней зважаючи на інформацію подану в підрозділі 2.3. Для цього було реалізовано стандартну функцію фізуалізації Набору Мандельброта, після чого додано реалізацію апроксимації відстані від точки до межі множини, а не конкретний контур (рисунок 3.1). Інакше кажучи, до існуючої формули ітеративної функції $z_{n+1}=z_n^2+C$, було додано обчислення похідної та аналітично оцінено відстань до фрактальної межі з використанням відповідної логарифмічної формули відстанні.

Такий підхід перетворення результатів фрактальних візерунків до оцінки відстані, проводимо з усіма доступними в майбутній програмі двовимірними фракталами. Це потрібно для того, щоб під час використання Raymarhing, геометрія об’єктів не мала обривів на конктурах, та містила згладжену поверхню.

3.1 Візуалізація із додаванням товщини контура

Першим з реалізації тривимірних фракталів було реалізовано метод їх відтворення на основі двовимірної форми та додаванням товщини по контуру фігури. Для цього використовувалась властивість Raymarhing, коли для виводу й обчислення числово результату певної відстанні тривимірно об’єкта, вносились

лише залежності даних двох світових осей (на вибір Z та X), що призводило до нескінченної візуалізації поверхні в напрямку невикористаної осі (рисунок 3.2).

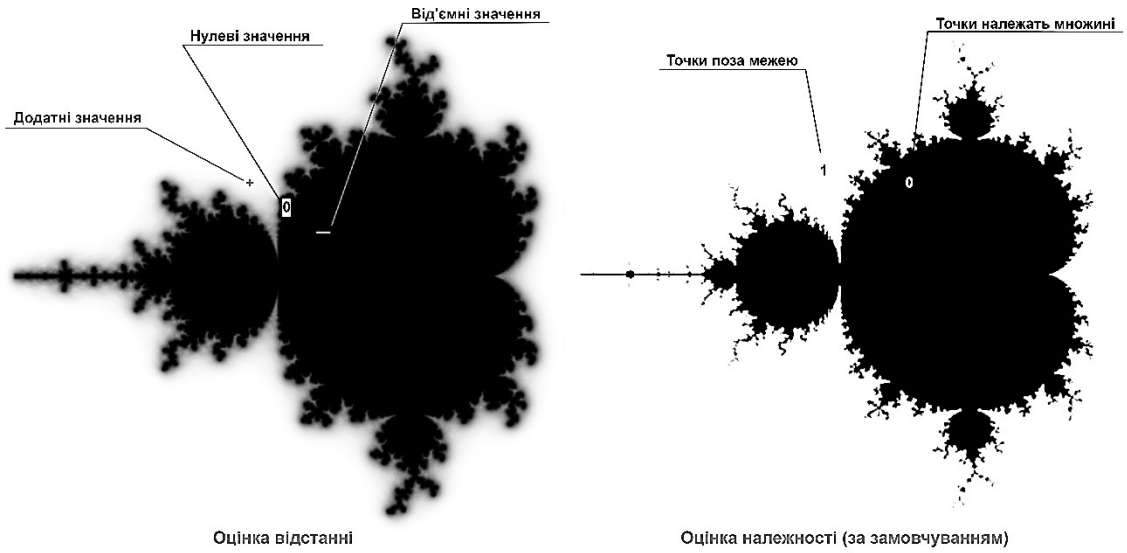


Рисунок 3.1 — Відображення фракталу Набору Мандельброта з використанням оцінки відстані та без

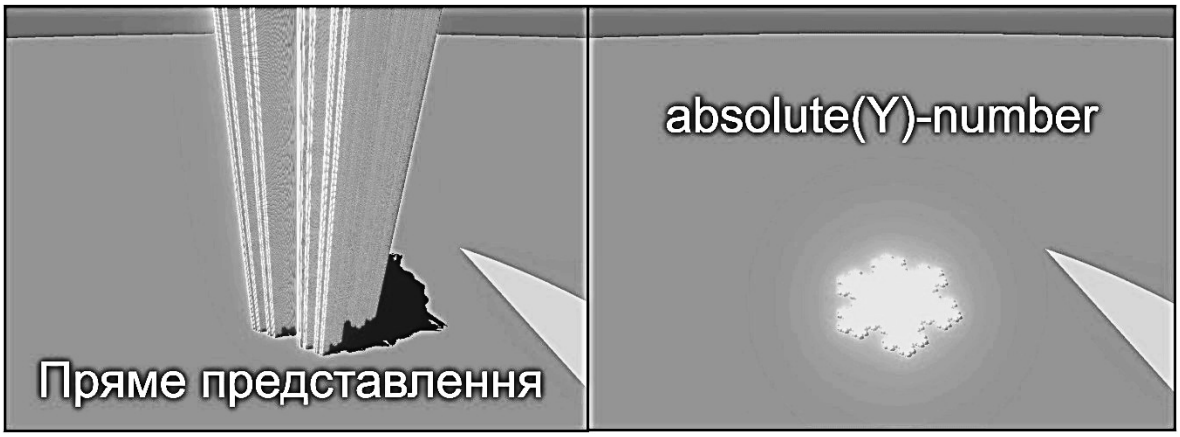


Рисунок 3.2 — Raymarching візуалізація двовимірного фракталу Сніжинки Коха

Згідно рисунку 3.2, бачимо, що оскільки стандартна формула фракталу призначена для двовимірного простору, то під час безпоседенього перенесення її в тривимірний простір, утвориться нескінченна візуалізація поверхні вздовж ігнорованої осі. Тому, оптимальним рішенням обмеження такої візуалізації було використання згаданої в пункті 2.2 роботи, операції перетину (max), згідно якої,

спочатку було створено обмежену по висоті площину за формулою абсолютного значення осі Y мінус обраного методом тестування числа. Як згадувалось раніше, при упущені осей, ми отримуємо нескінченну візуалізацію поверхні, тому оскільки згадана лише висота що було обмежена по одній осі, ми отримали вічно генеровану поверхну. Виконавши математичну дію пошуку максимального значення, між нескінченно генерованою вздовж осі Y Сніжинки Коха, та нескінченною площино, було отримано обмежено фіксовану візуалізацію вищезгаданого фракталу. Загальний принцип отримання обмеженого результату візуалізації двовимірних фракталів у тривимірному просторі таким методом показано на рисунку 3.3.

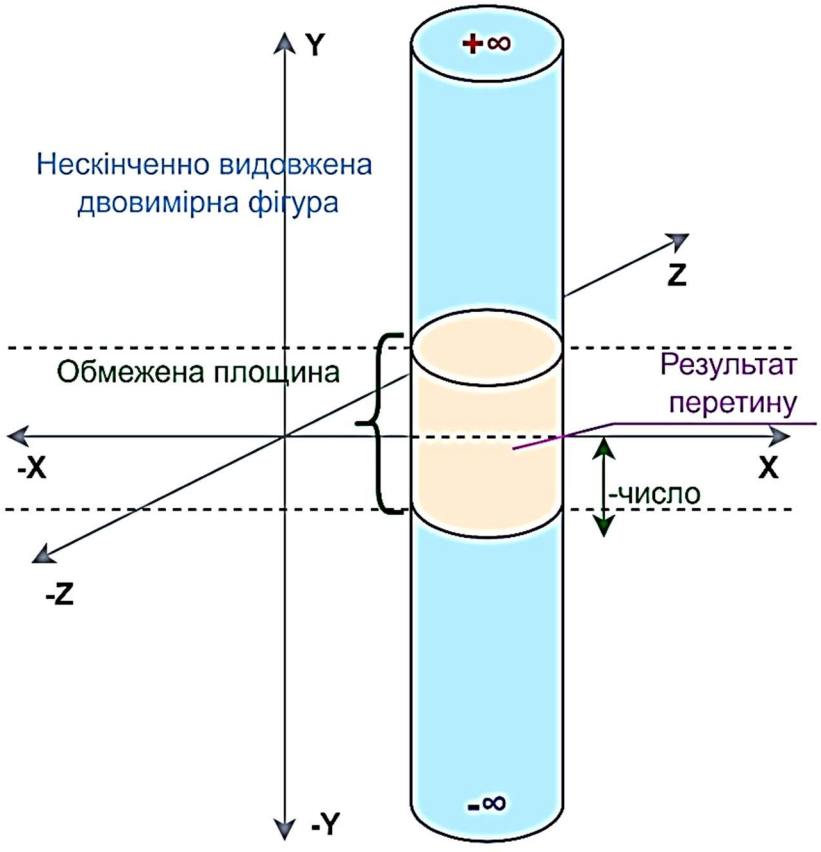


Рисунок 3.3 — Принцип обмеження візуалізації двовимірних фігур в тривимірному просторі Raymarching

Однак проблема полягає в тому, що такий підхід до візуалізації двовимірних фракталів у тривимірному просторі можна вважати технічно

коректним, та не математично. Головною ознакою фракталів є самоподібність, а отже при переході до нового виміру, ця характеристика повинна залишатись, проте з видимими ознаками в усіх трьох вимірах. Звідси, таку реалізацію було збережену в фінальному застосунку, однак через спірність з наукової точки зору, було позначено як два з половиною виміри, оскільки третій вимір відповідав лише за фактичне потовщення контуру, отриманих на кординатній площині результатів.

3.2 Візуалізація на основі структурної відповідності

Зважаючи на спірність першого методу тривимірного представлення фракталів, було реалізовано ряд специфічних підходів до візуалізації різних типів фракталів за природою, і всі ці підходи є частиною загального методу візуалізації запропонованого програмного забезпечення.

3.2.1 Комплексна залежність

Комплексні фрактали, які залажать від властивостей комплексних чисел, як на наприклад фрактал Множина Джулія, для тривимірної візуалізації потребують перетворення в гіперкомплексне узагальнення. Згдіно цього поняття комплексні числа можуть розширюватись на більшу кількість вимірів і одним із способів цього досягти, це перетворення до стандартної формули фрактала з представлення кординатної площини до сферичних кординат. Основними аспектами такого перетворення є відстань від початку кординат до обраної точки, що позначається як радіус; кута між вертикальною віссю та зенітного кута (в якому положені знаходиться точка відносно вертикальної віссі), які вважаються полярним кутом «тета»; кут між проєкцією на горизонтальну площину, та віссю X, що називажться азимутальний кут «фі». Графічне відображення цих величин, та їх зв’язку із декартовими кординатами у вигляді відповідних формул, вказане на рисунку 3.4.

Для визначення відстанні до об’єкта фрактала в тривимірному просторі використовувалсь видозміненний вираз пошуку точок для кожної відповідної

множини комплексних фракталів, з використанням залежностей переходу до сферичних кординат (див. рисунок 3.4). Таким чином, було отримано реалізації функцій комплексних фракталів, що наближено обчислюють відстань до точки у просторі до поверхні фрактала з використанням Raymarhing та сферичних кординат.

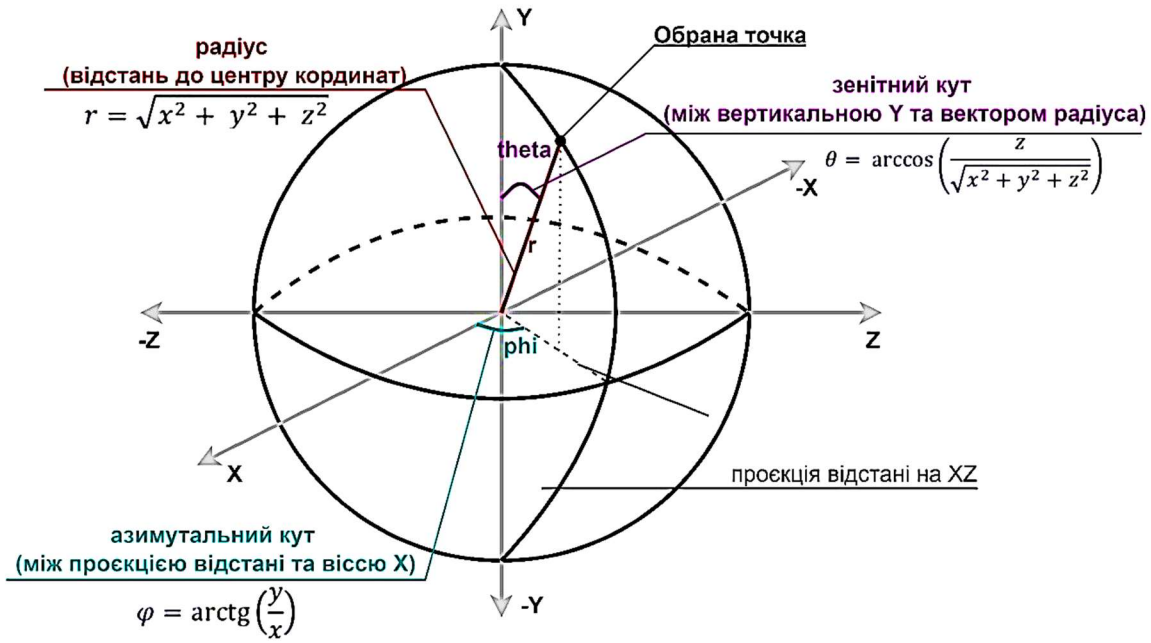


Рисунок 3.4 — Залежності декартових кординат від сферичних

Код шапки функції відображення відстанні Фракатала Мандельброта з використанням сферичних кординат матимеме вигляд: float Mandelbrot3D(float3 p, int iters, float t = 0), де p — параметр що позначає кординату точки в тривимірному просторі, iters — кількість ітерацій, t для анімації зміщення. В свою чергу в тілі функції конвертуються відносно властивостей шейдеру осі точки простору: p = p.zyx * 0.08, після чого визначається радіус від точки початку кординат простору — float r = length(p). Згодом в основному циклі for (int i = 0; i < iters; i++) на кожній ітерації перевизначається радіус точки, який позначає відстань до центру кординатного простору — r = length(z). Після чого перевіряється оснровна умова належності до множини фракталу й тільки за її коректності, обчислюються подальші величини як кут «тета» — theta = 2 *

acos(z.y / r), кут «фі» — phi = 2 * atan2(z.z, z.x) та похідна від основного значення формули мандельброта — dr = r * 2 * dr + 1. Отримавши відповідні сферичні параметри, обчислюємо значення Множини Мандельброта — z = r * r * float3(sin(theta) * cos(phi), cos(theta), sin(phi) * sin(theta)) + p + t. По виході з функції враховуємо оцінку відстанні в залежності від обчислених даних у циклі: return log(r) * r / dr.

3.2.2 Геометрична залежність

У свою чергу фрактали геометричного типу, потребують інакшого підоходу, оскільки вони базуються на базових двовимірних фігурах як квадрат чи трикутник, то під час їх реалізації візуалізації в Raymathing методиці було використано відповідні аналоги тривимірних функцій фігур як куб чи піраміда. Також, на відміну від комплексних фракталів, для геометричних недостатньо буде просто змінити всі існуючі формули за одним принципом переведення до сферичних кординат і видомзіненням формули для реалізації фігури в тривимірнрому просторі.

На прикладі фрактала Сніжинки Коха, двовимірна версія якої пояснена в підпукнкті 1.2.1, ми не можемо створити тривимірну модель на основі складання шести обернених площин стандартної структури із нескінченною поверхнею відносно шести напрямків різни осей світового простору. Оскілький результат таких дій не відповідатиме оскновній закономірності самоподібності на всіх рівнях масштабності. Для візуального розуміння цього, було розроблено тестовий підхід, що слідував умовам зазначеним на початку цього абзацу (рисунок 3.5), тобто методом операції перетину нескінченних площин.

Хоча такий метод дає значно більшу оптимізацію згідно самих обчислень, оскільки використовує базові операції Raymarching min— max, навідміну від математичних дій по типу добутку векторів. З використанням дублювання об’єктів під час збільшення операцій, було визначено, що метод із використанням операцій min— max в двічі оптимізованіше ніж метод з використанням математичних операцій. Однак результатом все одно не є фактична візуалізація

фракатала, що теоретично могла успадковувати ознаки двовимірного оригіналу. Оскільки основою Сніжинки Коха на кординатній площині є відрізки що повторюють контури двох сторі рівностороннього трикутника, то логічно припустити, що в тривимірному просторі аналогією будуть трикутні та чотирикутні піраміди або ж правильний тетраедр.

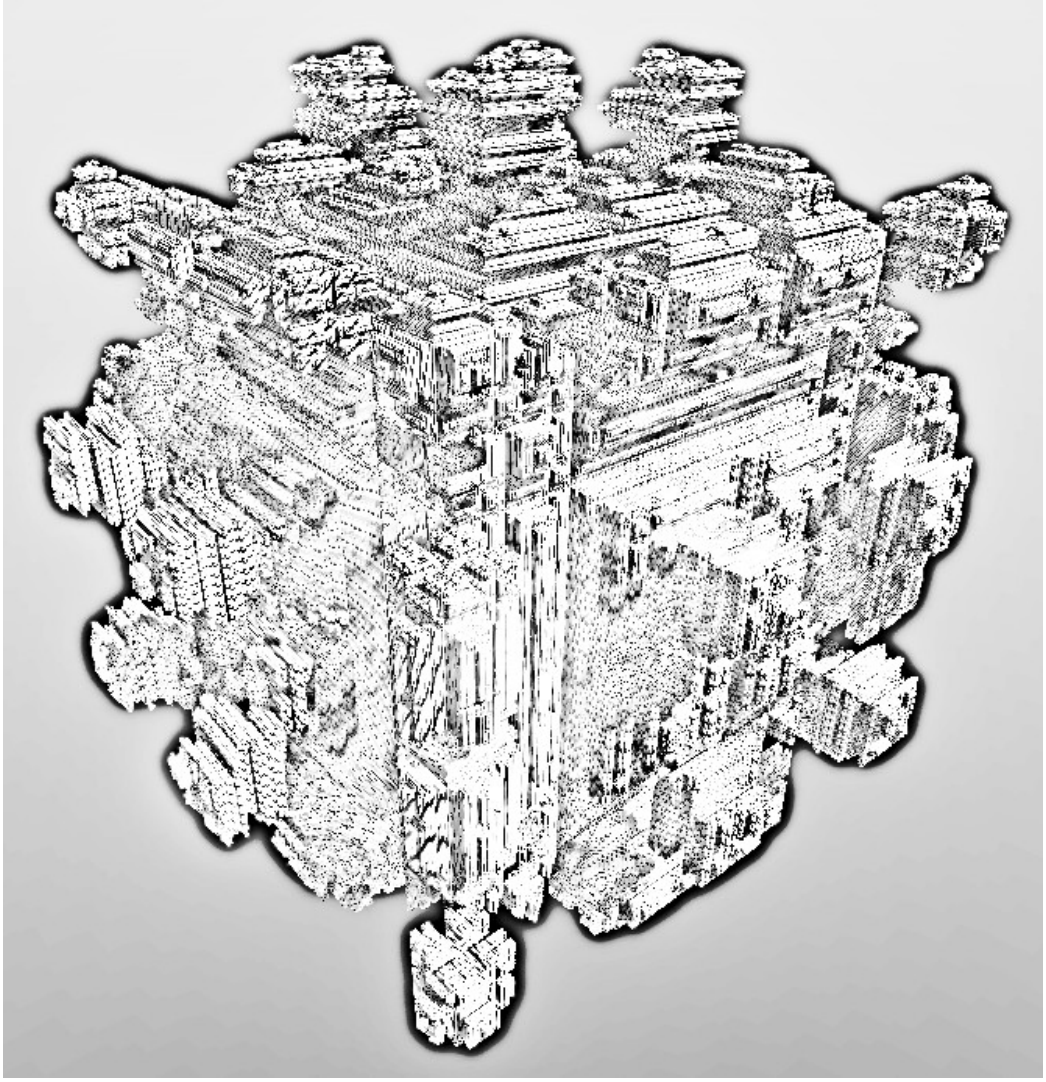


Рисунок 3.5 — Тривимірна візуалізація Сніжки Коха

Для забезпечення геометрії, за початкову основу було обрано й подальшу деталізацію було обрано рівносторонній тетраедр, що реалізовано з використанням базових операцій Raymarching. Однак через математичну складність обчислень, було обрано спрощений варіант, згідно яким використовувався створений октаедр за принципом перетину двовимірних

поверхонь нескінченних в різних площинах рівносторонніх трикутників в тривимірному просторі, та обчислення їх абсолютного значення вдіносно осей, що дублюватиме фігуру відносно горизонтальної площини (рисунок 3.6).

Оскільки під час перетину, бічні грані вже не є рівносторонніми трикутниками, юуло проведено масштабування висоти за принципом добутку на значення кореню з трьох поділених на два. Після чого у відповідному методі реалізації Сніжинки Коха в тривимірному представлені, було визначено умовні позиції вершин, центру вписаного в бічну сторону кола з використанянм відповідної формули та центри ребер. Ці всі дані обчислювались лише для однієї з просторових сторін, що позначала додатню частину кординатного простору, після чого було використано метод абсолютних значень, що в методиці Raymarching по суті дублює додатні значення у всіх напрямках кординатного простору (рисунок 3.7).

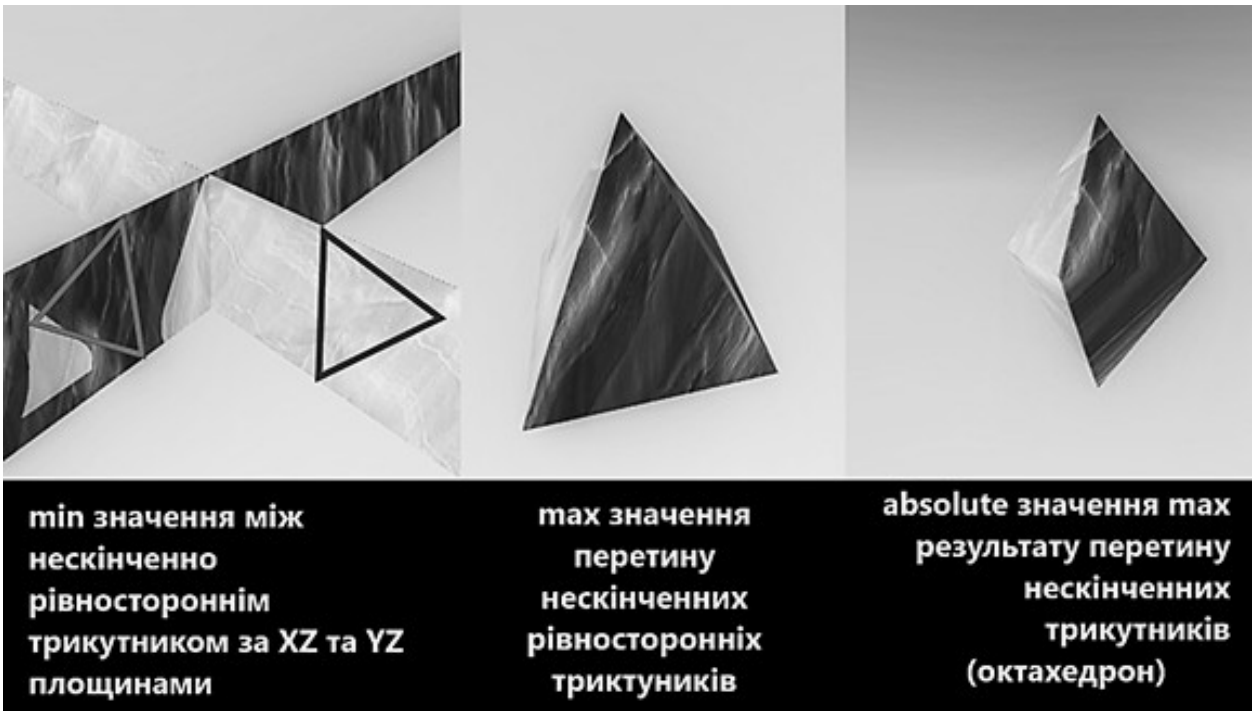


Рисунок 3.6 — Створення октаедра на основі проєкцій рівносторінніх трикутників

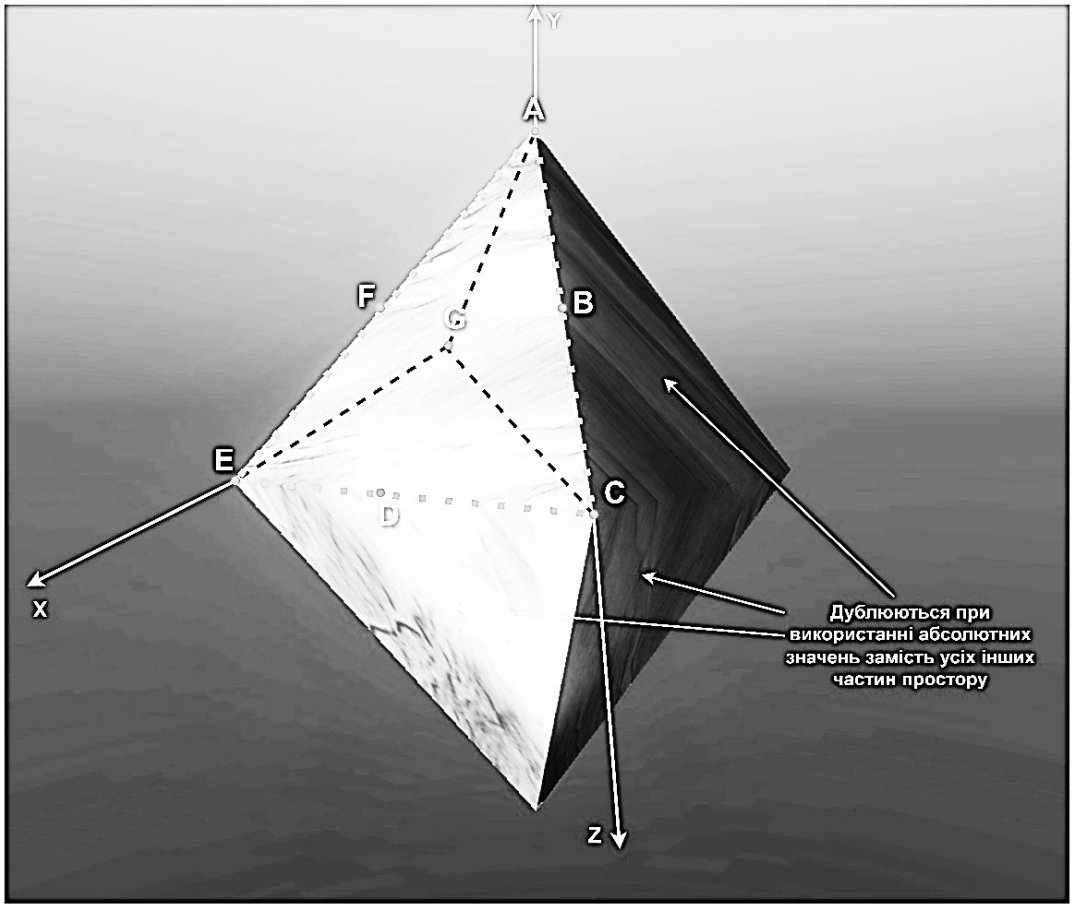


Рисунок 3.7 — Методика обрання точок для ітераційного розміщення менших частин фракталу

Саме таке розміщення мених копій початкового октаедра, дозволить реалізувати Сніжинку Коха в тривимірному просторі, що візуально буде мати властивості та виглядатиме як двовимірний оригінал, незалежно від ракурсу огляду камери на генеровану структуру. Відповідно в функції, створено цикл обробки котрий реалізує розміщення відносно цих кординат, зменшуючи поітеративно розміри самоподібного елемента октаедра та ділячи простір використовуючи абсолютні значення, що дозволяє уникнути поосібно звертання до кожної з нових деталізованих зменшених фігур. Результато такої візуалізації є фігура зображена на рисунку 3.8, що побудована з котаедра, на відповідних точках геометрії якого розміщені зменшені вдвічі копії октаедрів, на відповідних точках яких також присутні октаедри, а ця самоподібність проглядається стільки, скількип визанчено згідно кількості ітерацій.

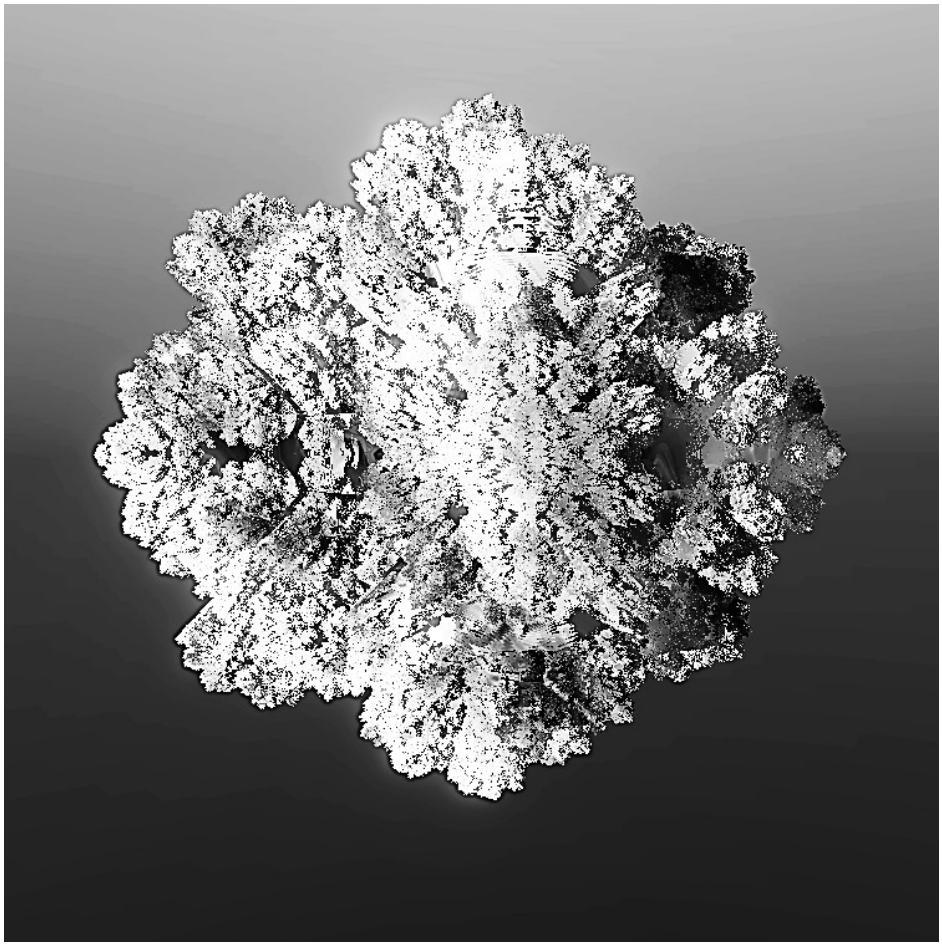


Рисунок 3.8 — Візуалізація фракталу Сніжинка Коха в тривимірному просторі

Аналогічно цього методу, для кожного іншого розглянутого в запропонованому програмному забезпечені, геометричного фракталу було розроблено власну реалізацію, однак всіх їх об’єднувало використання абслютних значень для дублювання самоподібності та добуток для масштабування під час кожної ітерації.

Після чого готовий матеріал шейдера з візуалізацією тривимірних фракталів, було накладено на модель інвертованої сфери, всередині якої, в тривимірному просторі сцени Unity, розташована керована користувачем камера огляду. Зокрема камера має фіксовану позицію, однак можливість обертання навколо власної осі, котра керується користувачем через використання відповідно написаного скрипту. Загальну вигляд роботи запропонованого методу наведено на рисунку 3.9.

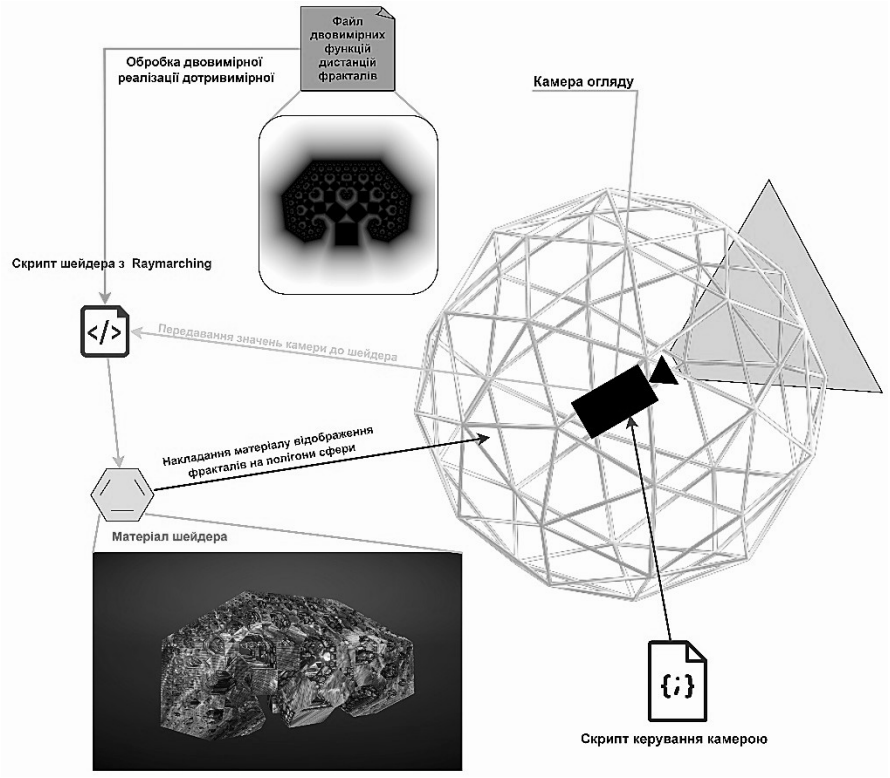


Рисунок 3.9 — Діаграма основних зв’язків та об’єктів запропонованого методу

Більш детальний опис основних аспектів відтворення методу, та розуміння основ його функціональності, наведено у наступних пунктах даного підрозділу.

3.3 Накладання матеріалу шейдера

Накладання матеріалів можливе на відповідно виділені під час моделювання полігони, та призначенні їм відповідних слотів для майбутніх матеріалів. Оскільки згідно винайденої методики, нам протрібно накласти на один шейдерний матеріал на інвертовану сферу, перейдемо до її створення. Для побудови сфери використовувалось програмне забезпечення для моделювання — Blender. Після відкриття даного середовища тривимірної графіки, було очищено усі об’єкти створені за замовчуванням. Після цього через комбінацію клавіш Shift+A, було відкрито меню створення тривимірної сітки, та обрано ікосферу з четвертим рівнем поділу геометрії (рисунок 3.10).

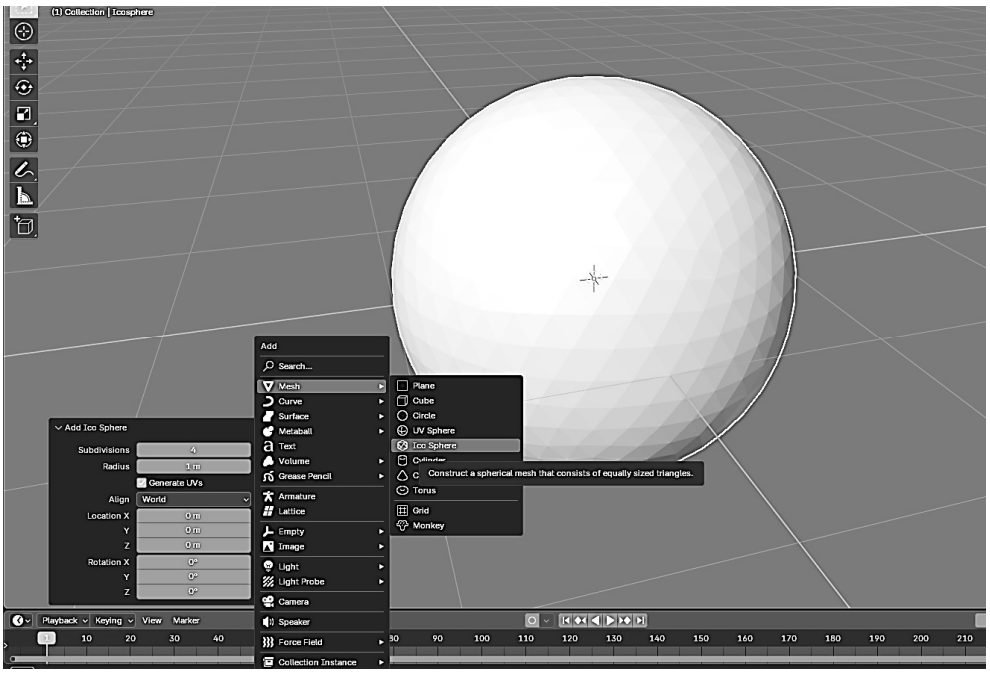


Рисунок 3.10 — Створення ікосфери в середовищі Blender

Для фігури накладання шейдеру, було обрано саме сферу через те, що в цьому об’єкті, всі полігони мають найбільш згладженні (тупі) кути між собою. Це дозволяє зменшити можливі похибки обчислень на контурах, про які детальніше розказано в пунткі 3.1.3. З цієї ж причини було обрано саме ікосферу а не UV— сферу, хоч остання має найбільш візуально кращу розгортку, проте текстура не рівномірно розподілена по поверхні моделі, що утворює надмірну деталізацію на полюсах і мінімальну деталізацію на екваторі (рисунок 3.11).

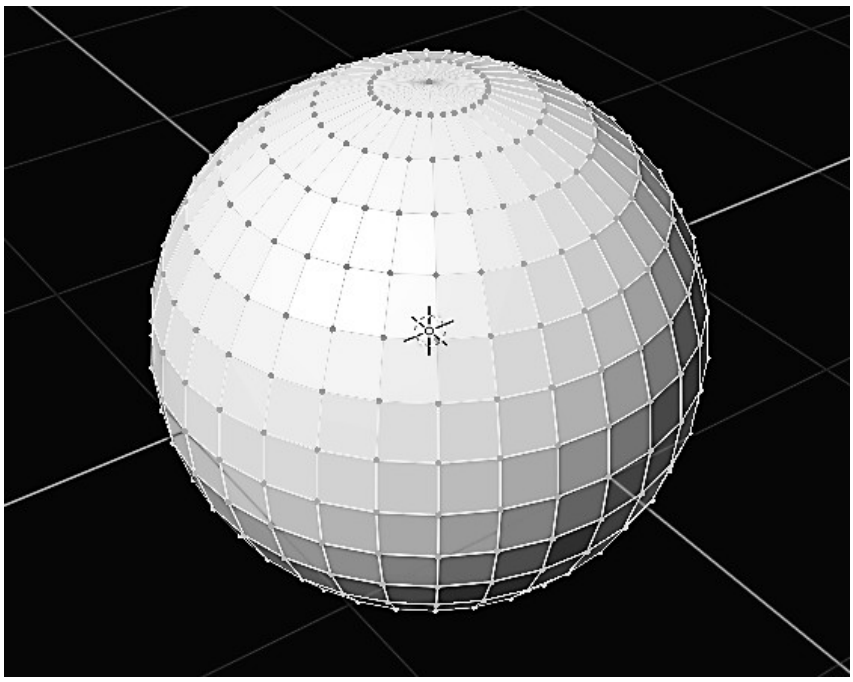


Рисунок 3.11 — Вигляд полігональної сітки UV— сфери

Даний тип фігури побудований з чотирикутників різної форми, а також трикутників на полюсах, що відповідно нерівномірно розподіляє вершини на моделі в цих напрямках. На відміну від цього, ікосфера не має таких проблем, тому результати обчислень у всіх напрямках погляду на сферу матимуть однакові результати, зокрема без спотвореності на полюсах, та зайвих нерозподілених навантаженнях через переважаючу кількість вершин.

Щоб створену сферу зробити інвертованою, потрібно змінити напрямок її нормалей, аби накладння текстури через UV— розгорту відбувалось на внутрішню сторону фігури. Для цього в середовищі Blender потрібно перейти в режим редагування через клавішу Tab, та виділивши всі грані ікосфери, здійснити перевертання нормалей з використанням сполучення клавіш Alt+N (рисунок 3.12). Для контрасту, няпрямків спочатку було інвертовано одну половину нормалей, слідуючи чому, під час візуалізації орієнтації нормалей частина була пофарбована в синій колір, що позначає спрямованість назовні, а інша частина в червоний — позначає внутрішню орієнтацію.

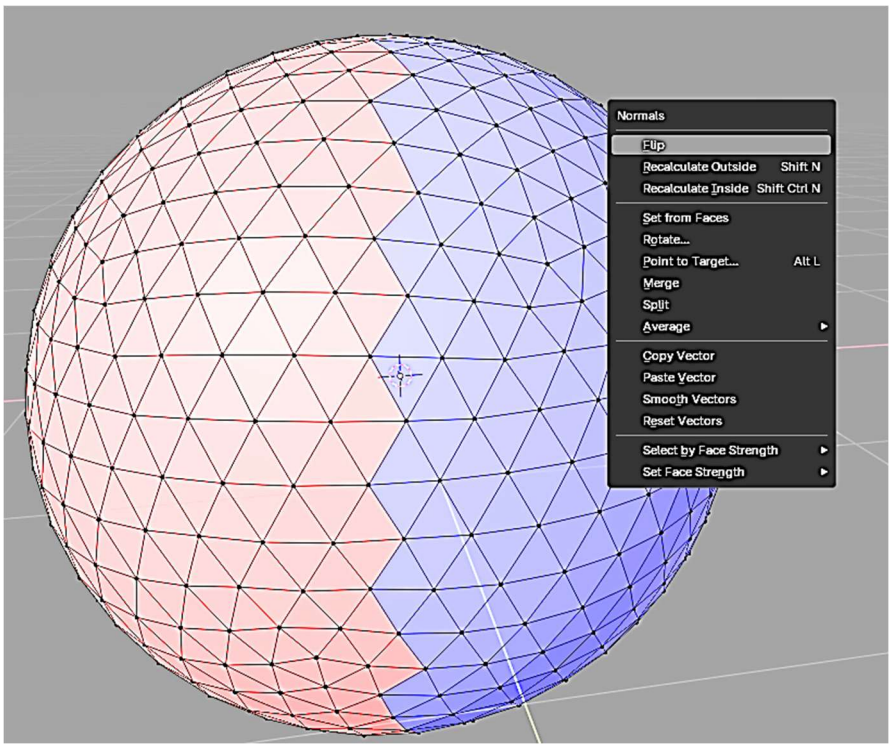
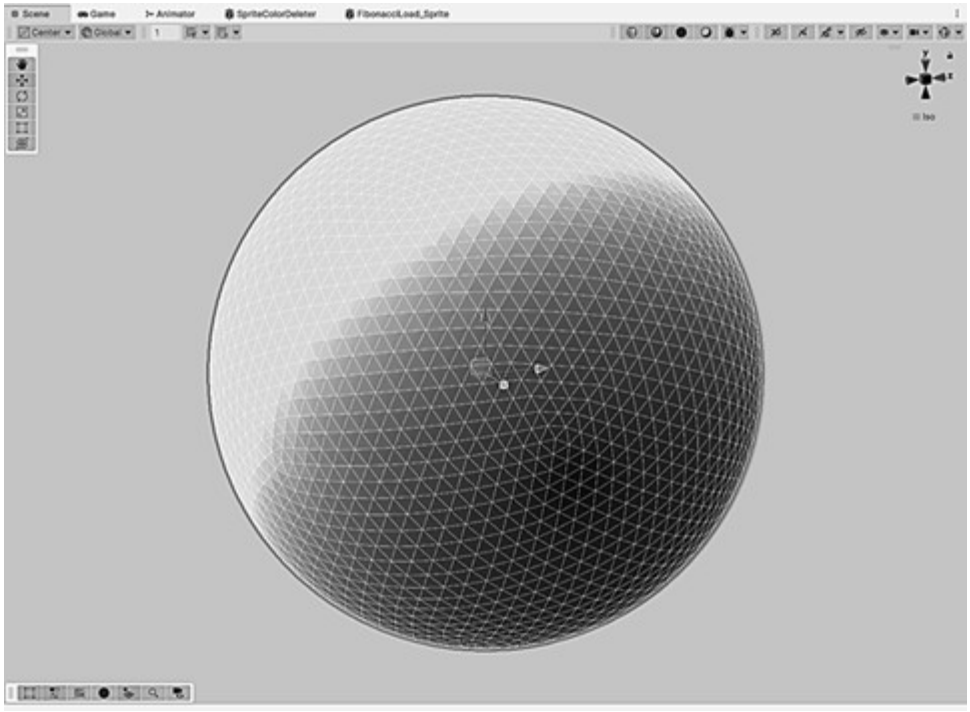


Рисунок 3.12 — Інвертування напрямку нормалей ікосфери

Завершивши основну побудову моделі, потрібно створити відповідний слот для майбутнього матеріалу шейдера, тому в інтерфейсі Blender обираємо вкладку Material та натискаємо клавішу New. Після чого в робочій області середовища, обираємо усі полігони фігури, та повернувшись до попередньо згаднаного меню й натискаємо Assign для призначення усій моделі конкретного матеріалу. Тепер файл моделі потрібно експортувати у локальне розташування файлів на власному комп’ютері з використанням послідовності меню File, Export, FBX. Додавший файл формату .fbx до вікна інспектора ресурсів Unity, перетягуємо модель на сцену. Для зручності встановлюємо позицію початку кординатного простору (нулі), а також перевіряємо інвертованість встановивши стандартний матеріал білого кольору в об’єкті моделі та ввімкнувши режим показу полігональної сітки з перекриттям матеріалу. Під час обертання моделі видно що сітка покриває матеріал білого кольору а не навпаки, тому візуалізується внутрішня сторона поверхні ікосфери, а не зовнішня (рисунок 3.13).



Рисуно 3.13 — Розміщення ікосфери на сцені Unity

3.4 Взаємодія з камерою

Для імітації простору використовується сфера з відповідною текстурою, однак для імітації тривимірності точки огляду, потрібно реалізувати обертання головної камери через управління користувачем, а також врахувати кут огляду відносно поверхні моделі, на якій візуалізується шейдерна текстура, аби врахувати корректний показ обчислень методу Raymarching.

Щоб це реалізувати, переходимо в ієрархію сцени, відкритого проєкту Unity та правим кліком миші викликаємо меню створення й обираємо об’єкт головної камери Main Camera. Налаштовуємо камеру так, аби її позиція збігалась із позицією сфери з майбутнім шейдером, тобто початок кординат простору сцени. Після чого встановлюємо тип проєкції на перспективу, щоб залишити природнє викривлення об’єктів через відстань та властивості розширення кута огляду. В залежності від розміру створеної моделі сфери, визначаємо межі відстанні видимих об’єктів камери в Clipping planes, так, аби Far параметр був на соту частину числа (найменша доступна одиниця оперування) більшим за радіус сфери, відповідно Near параметр встановлюємо на максимальне найближче

значення, що не перетинатиме оболонку моделі. Після налаштувань відстанні, огляд камери повинен включати лише поверхню сфери, що значно покращить продуктивність (рисунок 3.14).

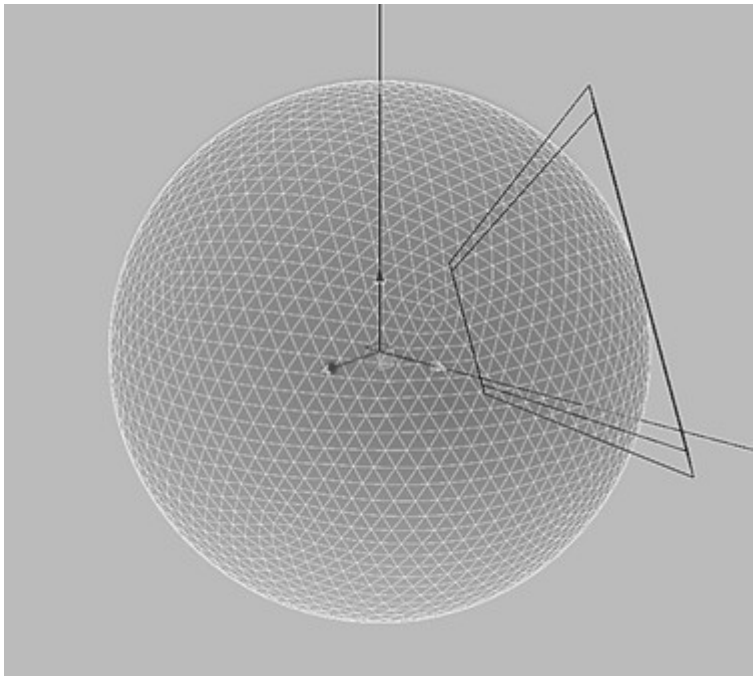


Рисунок 3.14 — Візуалізація поверхні сфери та видимого простору камери у вигляді зрізаної чотирикутної піраміди

Кут огляду камери встановлюємо на зручний для людського ока 70 градусів, враховуючи що візуалізація буде відбуватись на плоских моніторах персональних комп’ютерів. У розділі конфігурації рендерингу обираємо URP графічний конвеєр, що призначений для не складних математичних обчсилень спрямованих на використання пересічними потужностями пристроїв. Також додаємо параметр PostProcessing, що додасть можливість зчитувати кольори камерою, які виходять за межі звичайного кольорового діапазону RGB, й застосовувати графічні ефекти пост— обробки як світіння об’єктів (Bloom) з переважаючою інтенсивністю кольору, та поглинання кольору при недостатній інтенсивнсті кольору. Значеннння пріоритету рендерингу встановлюємо на найвижче, а саме — 1. Також використовуємо параметр Culling Mask, аби додати ігнорування виділених певними тегами об’єктів у світі, оскільки сцена в Unity

спільна для всіх розміщених у подальшому об’єктів, а нам не потрібна їх візуалізація на головній камері. Всі вище згадані налаштування камери в інспекторі рушія, показані на рисунку 3.15.

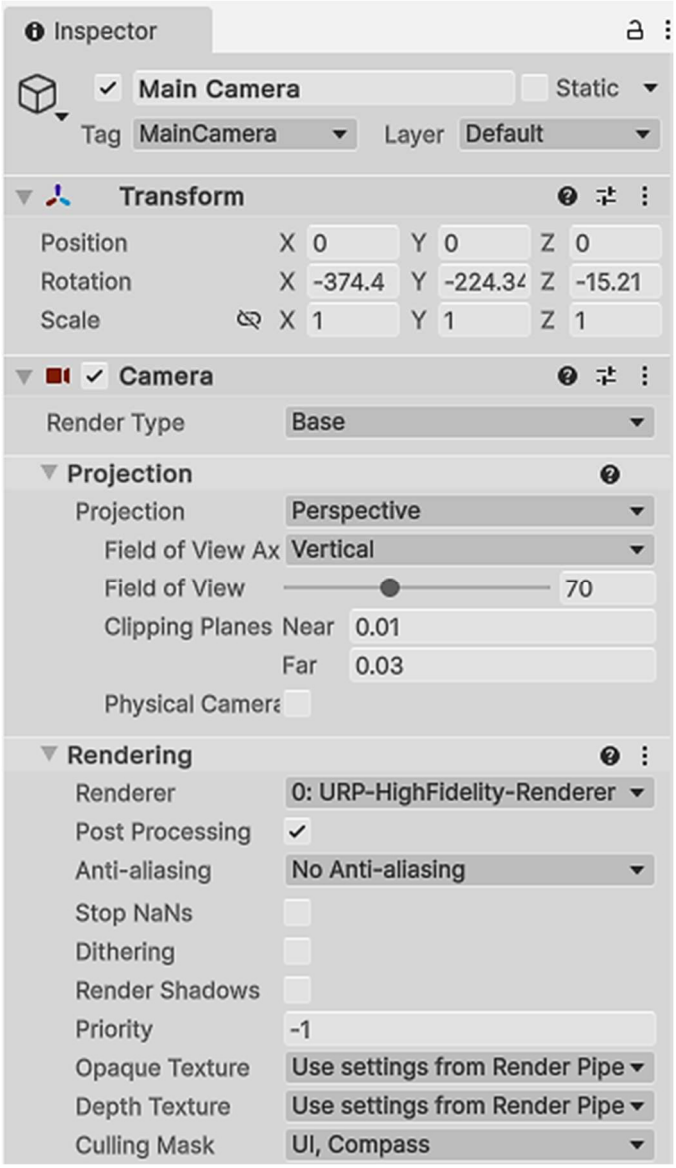


Рисунок 3.15 — Конфігурування головної камери огляду

Тепер під час запуску сцени ми маємо камеру що напрямлена в одну сторону, і поверхня сфери яка розташована зовні, завжди в області її видимості. Однак, оскільки шейдер буде створювати динамічне відображення фракталів, що прив’язане до поверхні моделі сфери, потрібно реалізувати можливість огляду й зміни його положення через зовнішні пристрої, як до прикладу мишу.

Тому створимо новий скрипт типу MonoBehaviour (рисунок 3.16), який відповідатиме за рух камери. Скрипти цього типу успадковують базовий клас MonoBehaviour від Unity, що надає доступ до взаємодії з редактором Unity та процесом відтворення застосунку.

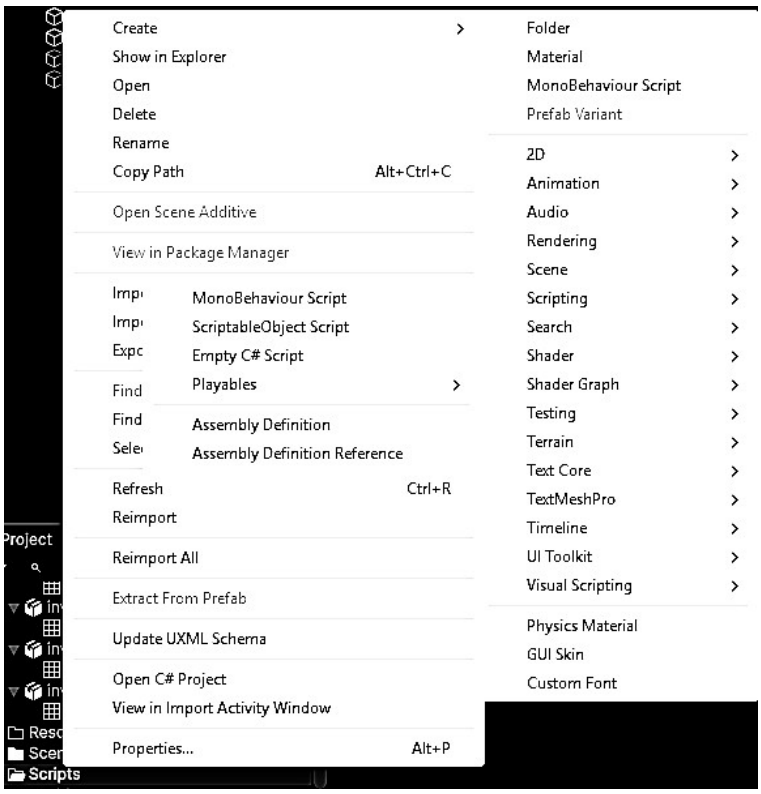


Рисунок 3.16 — Порядок стфорення скрипту керування обертанням камери

Успадкування MonoBehaviour дозволяє активізувати взаємодію з об’єктами ієрархії Unity, зокрема зчитувати чи змінбвати компоненти чи їх властивості, зокрема, як у нашому випадку властивості обертання трансформації камери. Також для таких скриптів відкривається доступ до методів життєвого циклу сцени, зокрема:

— метод Awake(), що викликається раз під час створення об’єкта на сцені;

— Start() викликається перед рендерингом першого кадру сцени, однак після Awake() та лише одина раз;

— метод Update() викликається щоразу перед рендеригом нового кадру гри;

— LateUpdate() викликається одразу після Update() аби врахувати нові зміни, які залежать від обчислень в Update();

— метод FixedUpdate() викликається з фіксованою частотою і не залежить від кількості кадрів на секнду, тому його часто використовують для обчислення фізики.

Вірним варіантом для зчитування даних зсуву курсору миші та відповідного обертання навколо власної осі камери, є вибір методу Update. Цей метод найкраще підходить для миттєвої передачі отриманих даних у сцену, оскільки FixedUpdate через визначенність оновлення часто може пропускати проміжні значення при великій частоті вводу, а LateUpdate створюватиме затримку некерованості після кожного вводу.

Для реалізації самого обертання було створено метод, що враховує зміну позиції миші на екрані, розраховуючи різницю по осі абсцис та ординат. Після чого, отримані горизонтальні дані з осі абсцис конвертуються з урахуванням визначеного множника шкидкості у кут обертання, який присвоюється оберту камери навколо осі «X». У свою чергу значення вісі ординат встановлюється на під час обертання навколо осі «Y». Під час Керування обертанням камери не використано вісь «Z», оскільки вона є напрямком руху, під час обертання якого, дуже легко втратити орієнтацію в просторі. Також, оскільки використовується Ейлерове обертання, під час обертання відносно горизонту, для забезпечення вірного обертання по осі ординат («Y»), обчислюється знак куту нахилу між вертикальною віссю камери та горизонтальною площиною світу відносно віссі абсцис (рисунок 3.17).

В залежності від отриманих даних знаку кута, під час обертання камери відносно осі «Y», значення руху миші по осі абсцис екрану модифікуються з його урахуванням. Для реалізації плавного змінення значень обертання камери, використовувався вбудований в Unity метод плавної зміни даних типу Vector2 під назвою SmoothDamp. Він приймає значення поточного положення обертання камери в просторі (по осі ординат та абсцис), потім приймає цільове значення

вектора, яке обчислюється зчитуванням даних миші, після чого використовує безпосередю змінну (через ref) для зберігання поточної швидкості зміни, а також враховує час шикдкості згладжування, який задається користувачем в довільному порядку й позначає час інтервалів за якими змінюються значення. Даний метод використовує експоненційну формулу, котра зменшує різницю між поточним та цільовим значенням враховуючи поточну швидкість зміни, котра оновлюється покадрово.

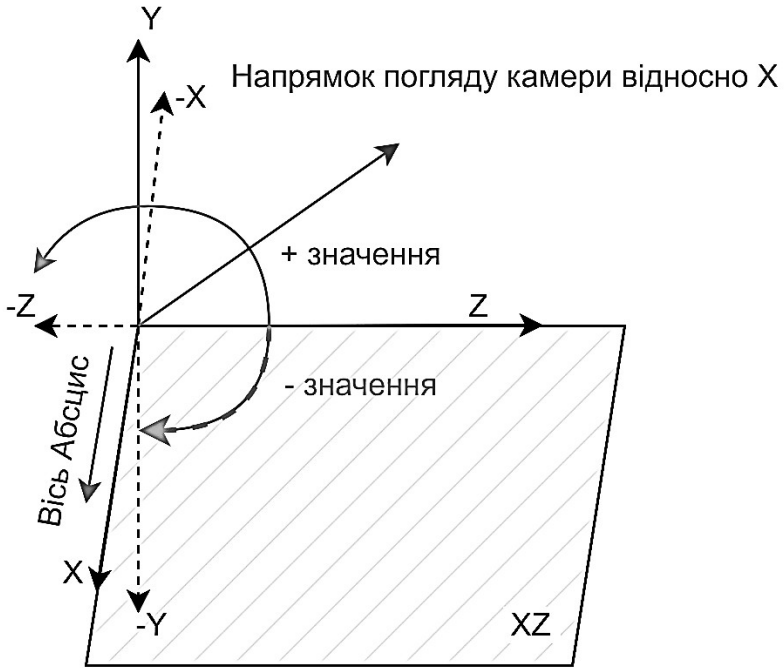


Рисунок 3.17 — Визначення знаку кута напрямку камери відносно горизонтальної площини

Завершивши налаштування скрипта руху, додаємо його до списку компонентів об’єкта камери в ієрархії сцени Unity. Після чого створимо невеликий тестовий шейдер для перевірки обертання камери під час керування мишею, оскільки створена сфера має звичайний білий матеріал, через що розпізнавання руху майже неможливе. Для цього створимо новий файл Shader Graphs типу Unlit та відкриємо його через інтерфейс середовища. Для імітації висоти використовувалися кольорові позначення світових значень позиції відносно моделі (фактично UV) та накладання кліткової текстури з полюсів.

Матеріал сформованого шейдера додаємо до нашої сфери та тестуємо під час запуску проєкту залежність руху миші по екрану до обертання камери в просторі, для зручності відкриємо вкладку загального вигляду сцени та вкладку ігового відтворення. Під час тестування бачимо, що камера обертається, відносно накладеної на сферу текстури, це ж проглядається і на загальному виді сцени під час візуалізації Gizmo позначень видимого простору камери (рисунок 3.18).

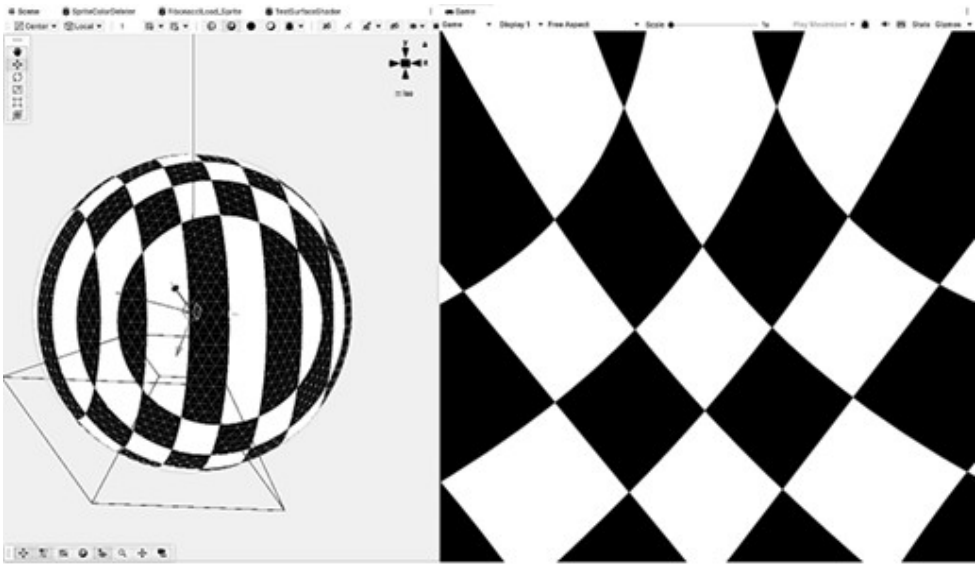


Рисунок 3.18 — Тестування скрипту обертання камери

3.5 Врахування позиції камери для шейдера

Як згадувалось раніше, при обчислені та візуалізації фракталів буде використовуватись шейдер із впровадженою методикою Raymarching. А оскільки текстура, котра має матеріал шейдера, накладатиметься на нерівну поверхню сфери, потрібно також враховувати відносний кут огляду камери спотерігача, а також одну систему позиціонування, аби умовні промені до пікселів проходили через локальні кординати камери.

Перш за все, камера в сформованій роботою методиці, слугуватиме початком з якого будуть прямувати й обчислюватися промені, важливо зрозуміти, оскільки промені обчислюватимуться з початку однієї точки, і прямуватимуть через відповідний кожен піксель екрану, для методу потрібно обрати лише

перспективний тип огляду камери. Також, маючи початок променя, потрібно знати його напрямок, в нашому випадку це буде світове положення сітки моделі нашої сфери. У частині вершинного шейдера, відносно кожної точки потрібно обчислювати відповідну позицію камери перетворену в локальний простір об’єкта (сфери). Цей підхід потрібен тому, що Raymarching працює в локальному просторі об’єкта й усі обчсилення відстанней були виконані в єдиній системі кординат. Також отримані дані перетину променя з поверхнею разом із його початковою точкою, згодом використовуватимуться під час обчислення напрямку для використання в основній функції Raymaching. Обчислення напрямку променя дорівнює нормалізованій різниці кординати зіткнення від кординати початку.

Якщо не реалізовувати враховування положення об’єкта відносно камери, за рахунок перетворення до локального простору об’єкта , під час зовнішнього спостереження за поврехнею об’кта, проглядатимуться явні скривлення обчислення накладуваної текстури (рисунок 3.19). Це зокрема тому, що враховується стандартна розгортка при обчисленні й накладанні текстури на сферу.

Рисунок 3.19 — Викривлення гріфічної візуалізації на поверхні сфери

Врахувавши перетворення зі світового простору в локальний, через використання функції матричного множення з відповідними позицією камери в світовому просторі та матрицею переходу від світового локального простору об’єкта. Серед основних практичних наслідків цього є залишення величин незмінним попри перетворення транформації камери абощо. Результат врахування цього методу, показано на рисунку 3.19, з відповідно встановленою позицією та кутом оберту камери до цього (див. рисунок 3.20).

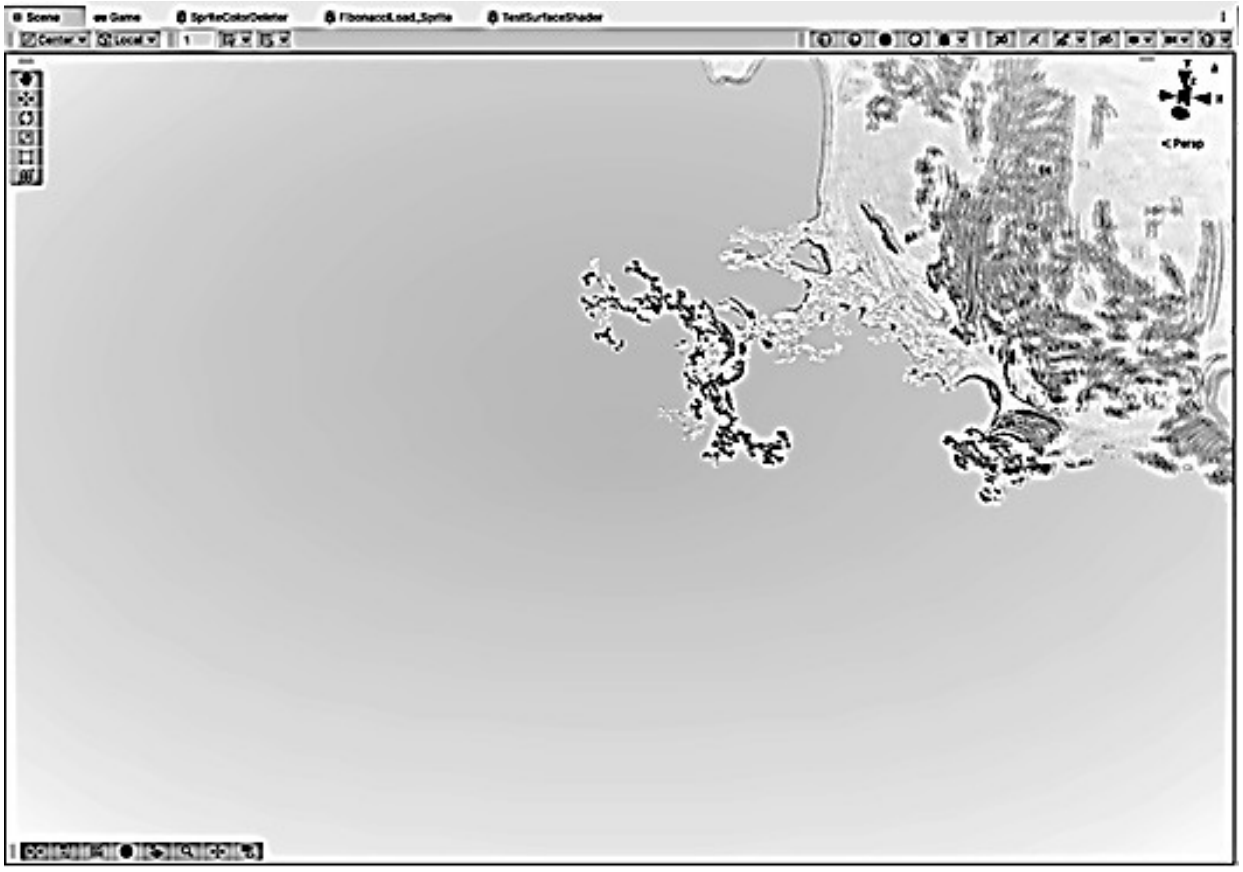


Рисунок 3.20 — Коректна графічна візуалізація на поверхні сфери

3.6 Альтернативний варіант реалізації методу

Попри існування головного методу реалізації поставленої мети з тривимірної візуалізації фракталів, існували й інші, менш успішні. Зокрема, одним із таких методів було створення моделі сфери засобами Unity сриптингу, та зміна її форми через вершинний шейдер.

3.6.1 Створення генератора моделі

Щоб реалізувати створення сітки моделі, що слугуватиме матеріалом для майбутньої побудови фракталу, використовувася спеціалізований скрипт в Unity. Аби зменшити ризики можливих спотворень під час деформації, найкращою формою для моделі буде сфера, яка зможе розташувати якумога більше вершин подалі від власного центру. Для її побудови було обчислено вершини, які згодом об’єднуватимуться в трикутники, а потім і саму UV— розгортку для належного розміщення текстури або ж матеріалу на поврехні, що важливо, оскільки форму фракатала буде реалізовувати фрагментний шейдер накладений на полігони. Основними ідеями реалізації є геометрична генерація поверхні полігональної кулі за допомогою сферичних кординат.

Побудова починається з визначення кількості паралеллей (горизонтальних розділень) та меридіанів (вертикальних), після чого цикл генерує кординати точок вершин сфери на основі сферичних, що трансформуються в декартову систему з використанням відповідної системи формул для кожної осі:

$$\begin{aligned} x &= \cos(\varphi) * \sin(\theta), \\ y &= \cos(\theta), \\ z &= \sin(\varphi) * \sin(\theta), \end{aligned} \tag{3.1}$$

де φ — кут довготи;

θ — кут широти.

Важливою деталю є те, що точки спочатку будуються як одинична сфера, згідно наведеної системи формул 3.1, тому при налаштуванні відповідного масштабування, достстньо просто перемножити отримані результати кординат, на конкретне число радіусу.

Кожні з чотирьох суміжних точок утворюють 2 трикутники, які й будуть формувати сітку з прямокутних полігонів моделі, тому обов’язково реалізується через збереження індексів у списку вершин, котрі в подальшому позначатимуть порядок з’єднання трьох вершин у трикутник (рисунок 3.21).

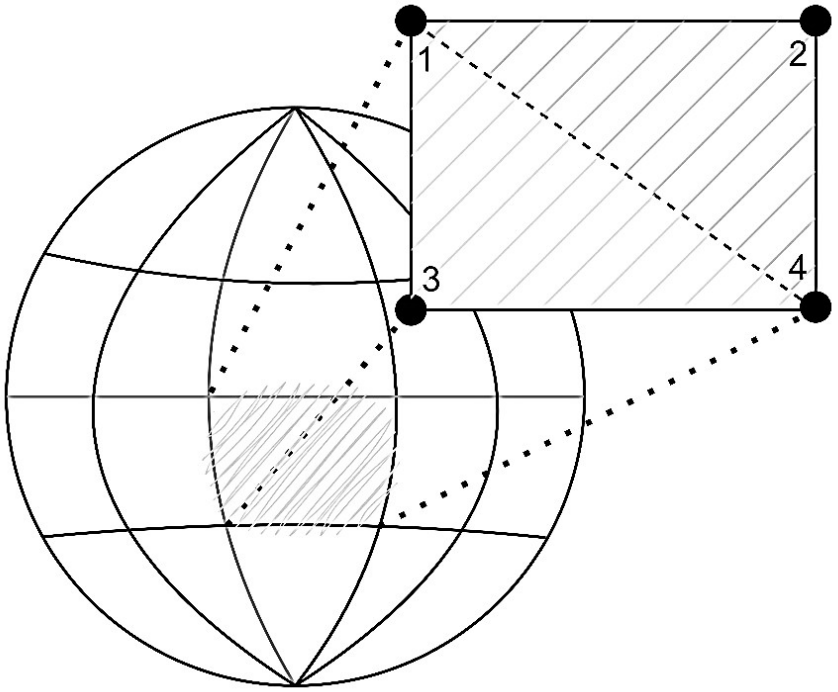


Рисунок 3.21 — Поділ полігонів сфери трикутники

Відповідно для корректного накладання матеріалу на модель, розраховуються паралоельно UV— кординати. Після чого на основі методів класу представлення 3D моделі «Mesh», формується безпосередньо сама модель з указуванням відповідно обчислених вершиш, трикутників та UV кординат.

Для зручності тестування, даний скрипт має публічні поля longitudeSegments та latitudeSegments, що відповідають кількості паралелям та меридіанам і можуть бути встановлені під час роботи запущеної сцени проєкту Unity.

3.6.2 Тестування та виправлення проблем

Перемістивши функцію генерації сфери в метод Update() та додавши відповідний тригер спрацювання перейдімо до тестування створення тривимірних моделей сфер на сцені обраного рушія. Для цього створимо два нових пустих об’єкти в ієрархії сцени, та додамо до них написаний скрипт генерації сітки. Встановимо різні значення кількості широт, довгот та радіусу

генерованих сфер та запустимро відігравач редактора, щоб побачити результати (рисунок 3.22).

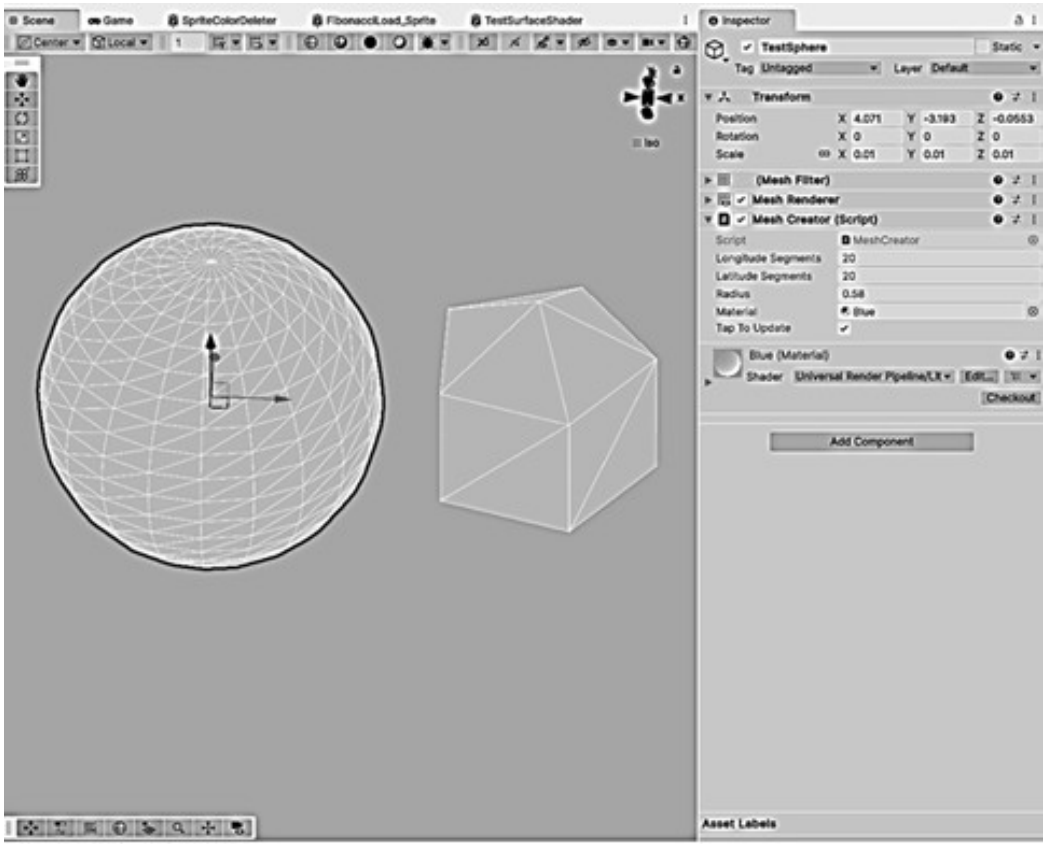


Рисунок 3.22 — Генерація сіток двох різних сфер

Як видно з рисунку 3.22, тестування завершенно успішно й скрипт, що створює сфери відповдіно до заданих параметрів — працює вірно. Слідуюча проблема полягає в тому, що навіть такої кількості точок як показано під час тестування — буде недостатньо для деталізованої візуалізацій фракталів, тому було проведено тестування з-зі збільшеною кількістю паралелей та меридіанів у сто разів. Результати такого тестування привели до того, що проєкт аварійно припиняв роботу через те, що запущена програма не відповідала. Пробелема полягає в тому, що така велика деталізація породжує незліченну кількість точок, які ми зберігаємо в пам’яті пристрою як вершини, індекси трикутників, а також розгортки.

Хоча виділена пам’ять потрібна лиш частково на момент обчислення, процес створення вимагає забагато часу, а також створює цикли з тисячною

кількістю ітерацій, тому аби виправити це, було використано підхід корутинів. Перевагою корутинів є можливість розподіляти довгі обчислення між кількома кадрами, що навідміну від звичайних функцій, котрі виконуються одразу, дозволяє призупинятись виконанню коду (в зазначених місцях) та відновлюватися пізніше. В Unity, коли програма має обчислення, що займає десятки мілесекунд, вона може аварійно завершити роботу або ж редактор видасть повідомлення про те, що додаток не відповідає. Відповідно вказавши yield return null; після кожної важкої ітерації обчислень у корутині, ми можемо надати процесу команду перерватись на іншу роботу, аби всі інші завдання продовжували виконуватися. Це вирішило проблему, хоча й тепер під час довгих створень, здається що нічого не відбувається, хоча на справдні поки не закінчитсья процес формування моделі, розподілений корутинами, то результат не буде виведений на екран.

3.6.3 Створення вершинного шейдера

Суть цього варіанту реалізації методу полягає у створенні шейдера, що буде змінювати положення існуючих вершин певної моделі, котра програмно створюватиметься як підгрунтя для візуалізації, в залежності від обраного рівня деталізації.

Для тестування придатності такого підходу створимо шейдер Shader Graphs типу Unlit. Після чого потрібно отримати позиції усіх вершин, для цього розмістимо вузол Position з обраним налаштуванням Object. Маючи позиції усіх вершин, спробуємо їх змінити до форми фракталу. До прикладу реалізуємо функцію в файлі HLSL, що відмальовуватиме множину мандельброта (лістинг 3.1).

Лістинг 3.1 — Будова функції візуалізації множини мандельброта

```
void Mand_float(float2 UV, float iterations, float scaler, out float dist)
{
   UV — = float2(.77, .5);
   UV *= 2.7 * scaler;
```

```
    float2 z = 0;
    float2 dz = 0;
       for (int i = 0; i < iterations; i++)
    {
       // f` = 2z * dz + 1
       dz = 2. * float2(z.x * dz.x — z.y * dz.y, z.x * dz.y + z.y * dz.x) + 1.9 /*.6*/;
       // f = z2 + c
       z = float2(z.x * z.x — z.y * z.y, 2 * z.x * z.y) + UV;
       float q = dot(z, z);
       if (q > 16)
       {
          dist = sqrt(q / dot(dz, dz)) * log(q);
          break;
       }
    }
}
```

Наведений вище код, обчислює значення множини мандельброта для кожного пікселя екрану, й передає отриманий результат у вигляді відстанні до краю множини за рахунок використання похідної до формули. Таке рішення плавного згладження по контурам, дозволяє уникнути різких перепадів та помилок при керуванні позиціями вершин.

Маючи написаний код функції Набору Мандельброта, повертаємось до інтерфейсу Shader Graphs та створюємо новий вузол під назвою Custom Function. В параметрах вузла вказуємо посилання на файл з власноруч написаною функцією, та прописуємо усі її відповідні параметри. Після чого за умови правильного вводу, графічний вузол одразу покаже вихідний результат (рисунок 3.23).

Успішний результат виводу візерунка фрактала Множини Мандельброта свідчить про те, що код написаний вірно. Тепер потрібно видозмінити позиціонування вершин у просторі, зокрема з використанням математичних операцій як множення та додавання. Для створення початкового прикладу візуалізації фракталу було створено інвертовано значення відстанні фрактала, після чого перемножено відносно осі ординат об’єкта, та додано до початкових

позицій сфери. Отриманий вихід значень позицій вершин, з’єднуємо з візуальною частиною підрозідлу шейдера Vertex, параметру Position.

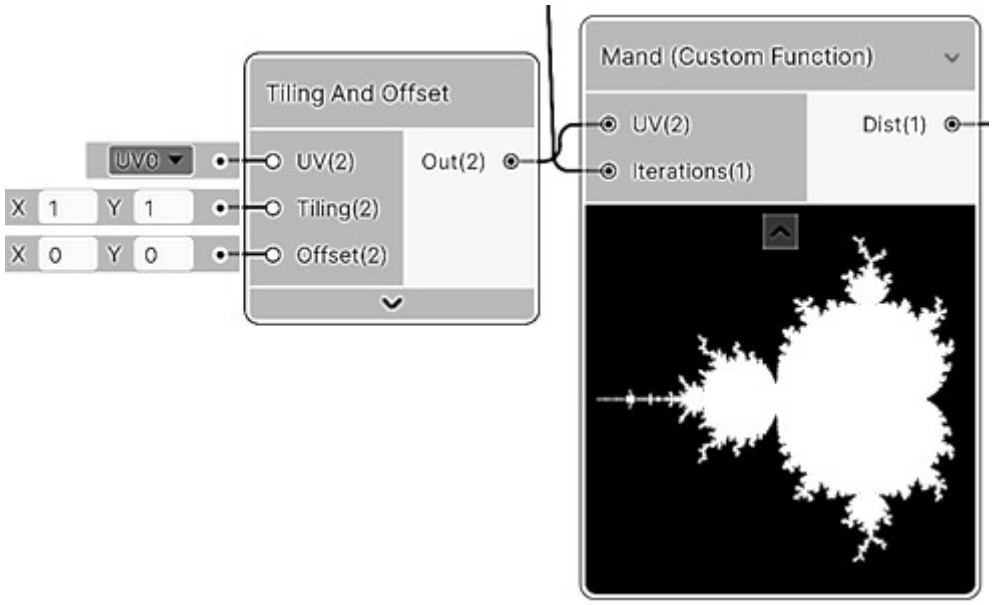


Рисунок 3.23 — Вузол функції візуалізації фрактала Множини Мандельброта

3.6.4 Виявляння недоліків методу

Закінчивши написання шейдера й розміщення основних вузлів, перейдемо до тестування. Для цього запустимо проєкт у режим відтворення, та створимо два пусті об’єкти в ієрархії сцени для загального прикладу, після чого додамо скрипт формування моделі до кожного з них. У вхідні дані скрипті, запишемо значення 1000 та 500 для позначення кількості генерованих паралелей та меридіан відповідно. Згненерувавши дві моделі додамо створений матеріал вершинного шейдера до моделей, та перевіримо, загальний результат зміни позицій вершин згенерованих сфер. В підсумку таких дій, маємо візуалізацію Множини Мандельброта у сферичних кординат тривимірного простору сцени рушія Unity (рисунок 3.24), створену методом вершинного шейдера.

Як видно з рисунку вище, візуалізована форма фракталу Множини Мандельброта дійсно має чіткі та розпізнавані контури оригінального фракталу, однак, першочерговою проблемою є його сферична спрямованість. Оскільки

базові кординати точок розташовані за принципом розподілу UV— сфери, як і в будь— якій іншій моделі, всі сусідні вершини мають налаштовані зв’язки між собою. Це означає, що навіть під час використання продуманого вершинного шейдера, ми не можемо мати достовірної інформації про те, які вершини мають зв’язки з іншими вершинами. Через що, загальні положення завжди будуть зберігатись, а спроба їх змінити таким способом, може зламати порядок побудови полігонів, через що візуалізація виглядатиме зовсім не так як очікувалось. Для вирішення цієї проблеми можна створити дешифратор моделі, що буде розділяти кожні зв’язки точок або ж сортирувати їх до відповідної нової візуалізації в правильному порядку. Однак в такого підходу є дві основних проблеми, перша це значні затрати на час та певні ресурси, а друга це не можливість вірної підготовки позицій для використання шейдером, оскільки його результат залишається закритим аж до закінчення виконання, яке відбувається кожен кадр. Зважаючи на всі ці недоліки, можна дійти свідомого висновку, що продовжувати розвивати цей метод далі — немає сенсу, зокрема в же й тому, що навіть система, котра дозволяє генерувати фактично нескінченно деталізовані сфери, кількість вершин яких обмежується лише оперативною пам’яттю пристрою, має значний недолік витрат по часу.

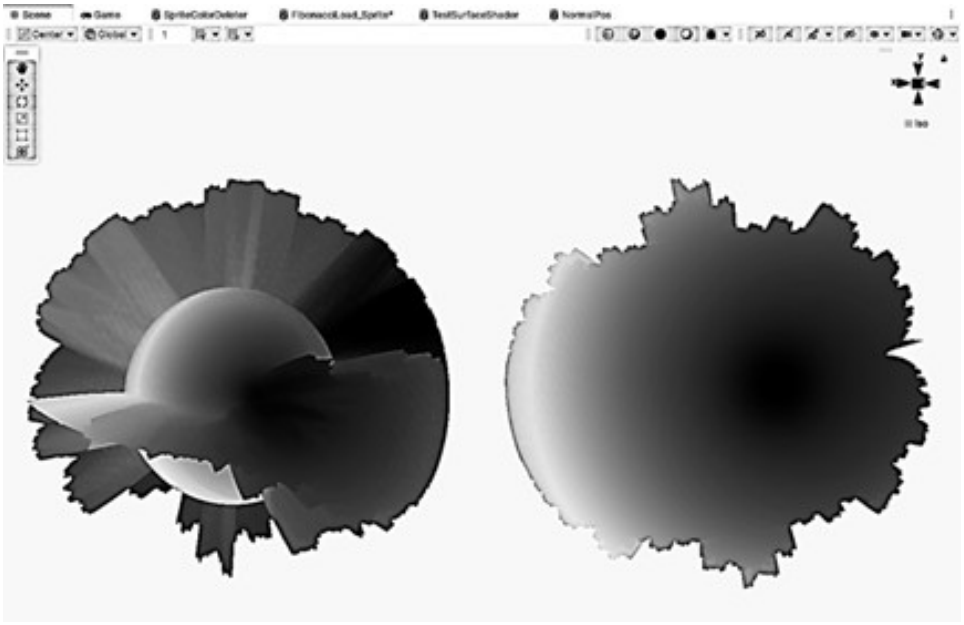


Рисунок 3.24 — Візуалізація Множини Мандельброта

Варто зауважити, що збільшення деталізацію планової фізуалізації фракталу, збільшуватиме кратно навантаження на генерацію моделі, під час створення, додаткового навантаження щокадру під час відмалювання трикутників сітки, а також покадрового навантаження від застосування вершинного шейдера. Проте в методиці з використанням фрагментного шейдеру з імітацією тривимірного простору — проблеми подібних візцуалізацій та навантажень немає, тому метод вершинного шейдера було відкинуто як невдалий підхід до тривимірної візуалізації фракталів.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕСТУВАННЯ ПРОДУКТИВНОСТІ

4.1 Побудова архітектури програмного забезпечення

Не менш важливим процесом у створенні будьякого програмного забезпечення, є його структурування організації коду, систем та зв’язків. З підрозділу 3.4 можна зрозуміти базові взаємодії між певними компонентами застосунку, а саме камера, ввід користувача для її керування, побудова шейдера, та можливість ним оперувати, що передбачає створення інтерфейсу користувача. Оскільки такі взаємодії передбачають власні типи файлів та різноманітне відношення до використання вони нагромаджуватимуть загальний простір структурних файлів програми без належного поділу.

Згідно головних вимог, поставлених до програмного забезпечення, а також типів даних, які в ньому використовуються, структура файлової ієрархії застосунку має такий вигляд:

— Assets є основною директорією файлів проєкту;

— Animations містить файли анімацій, в основному пов’язаних із рухом інтерфейсних елементів;

— Materials містить усі генеровані шейдерами матеріали;

— Models налічує моделі сфер, враховано декілька типів з відповідними рівнями деталізації полігонів для різних типів оптимізації;

— Scenes це основна директорія для зберігання сцени рушія Unity, для запропонованого програмного забезпечення використовувалась одна активна сцена;

— Scripst містить основні скрипти програми, написані мовою C#;

— Settings це автоматично створений шлях рушієм, для збереження налаштувань графічного конвеєра;

— ShaderGraphs являє собою відокремлену папку для зберігання шейдерів, побудовані візуальним інструментом Shader Graphs;

— Shader це директорія для зберігання шейдерів написаних у вигляді скриптів мови HLSL;

— Textures є основною папкою з вмістом усії використовуваних в програмі текстурних зображень.

Для розподілення функціональності програми на менші одиниці, було створено декілька окремих файлів з відповідними класами, кожен з яких має власну відповідальність та роботу. Зокрема скрипт AppManager з відповідним класом, відповідає за правильний порядок запуску й активації усіх скриптів створених у програмному забезпеченні. GuiManager являє собою ще одним важливим класом для роботи з інтерфейсом користувача, зокрема керування елементами, зчитування та зміни їх станів. Для обробки й керування об’єкта камери було створено відоповідні два скрипта, один з них є класом CameraRotation, що реалізує зв’язок з ігровим елементом в ієрархії сцени Unity для зчитування для цього даних та відповідного обертання камери. Також, для обробки натиснення клавіш, що керують елементами інтерфейсу та застосунку загалом створено клас UserInput, що звчитує відповідні дані з клавіатури. Для зв’язку між шейдером типу HLSL файлу, було створено відповідний клас ShaderSettings, що приймає відповідні дані зчитані чи внесені через інтерфейс та клас GuiManager, обробляє їх, та передає напряму до файлу шейдера. Для зручності передавання даних, їх було розділено за структурами:

— Light (містить інформацію про рендер сонця, його рух, інтенсивність світла, його колір, колір неба, колір поверхні, колір середовища й фракталу, позицію сонця та прозорість тіней);

— Raymarch (вміщує інформацію про кількість променів що випускаються з кожного пікселя й напряму впливають на промалювання деталізації, мінмальну дистанцію, яка позначає відхилення, що може вважатись досягненям поверхні, максимальну дистанцію, швидкість зсуву та саму позицію зсуву спостерігача);

— FractalParam (містить інформацію про обраний фрактал, як його масштаб, константа, кількість ітерацій, ідентифікатор, текстуру накладання та анімованість).

Клас ShaderSettings також використовує проміжні класи, що сигналізують про зміну того чи іншого типу елементу інтерфейсу, для оновлення відповідних даних у налаштуваннях шейдера.

Також було додано класи для реалізації додаткового функціоналу, пов’язаного з інтерфейсом, про що детальніше описано в підрозділі 3.5.

4.2 Структурна побудова основного шейдера

Кожен шейдер Unity написаний мовою HLSL, має три блоки побудови, як властивості (Properties), підшейдер (SubShader) та секція програмного коду (Pass).

Перша секція, властивостей, створена для розміщення певних вхідних параметрів, які в нашому випадку позначатимуть основні вхідні дані до обчислення Raymarching візуалізації на вихідній текстурі. Кожна із властивостей має назву змінної, після чого короткий опис у лапках, вказування типу даних та якщо треба вказування початкового значення. До прикладу, змінна цілочисельного типу, що матиме назву _RayIters (нижнє підкреслювання є обов’язковим форматом для віхдних змінних шейдера) та значення за замовуванням 250, під час оголошення виглядатиме як рядок коду: _RayIters("Iterations of rays", Integer) = 250.

Причому, синтаксис наведений вище потрібен для повідомлення інспектору про додавання фізичного поля в інтерфейс з відповідним типом даних, а для використання змінної всередині самого шейдера, потрібно оголосити відповідно до синтаксису HSLS таку ж за іменем змінну в блоці Pass, що виглядатиме для вищезгаданої властивості так: int _RayIters.

Оскільки шейдер містить два основних блока Properties та SubShader, основний код було написано в частині Pass, що належить другому блоку. Насамперед в ньому було оголошено усі змінні, котрі мають відповідників у

властивостях, після чого стандартні структури кожного шейдера, а саме appdata (структура, що приймає вершинни, нормалі та відповідні кординати UV для кожної точки, тому вважається структурою вершинного шейдера, що зчитує дані передан з центрального процесора), v2f (структура, яка передеає дані з вершинного шейдера до фрагментного піксельного), vert (безпосередньо функція вершинного шейдеру, що дозволяє оперувати кординатами вершин), frag (безпосердньо функція фрагментно шейдеру, що отримує перетворені вершини моделі у вигляді пікселів на екані, й дозволяє видозмунювати їх колірне представлення).

Оскільки кожен фрактал потребує повного ручного написання двовимірної та тривимірної функції з виведенням відповідного результату, це в підсумку створює проблем з великим нагромадженням по кількості функцій у файлі, тому було створено окремий файл для функцій реалізацій фракталів, під назвою Fractals.hlsl. Для початку використання в основному коді шейдера, написаних фрактальних функцій, було включено імпорт відповідного файлу ключовим записом #include "Fractals.hlsl". Відповідно у самому файлі з функціями, як і в будь— якому подальшому файлі hlsl типу що імпортується, потрібно перевірити оголошення макросу файлу, щоб запобігти включенню одного й того ж заголовку. Інакше кажучи на початку таких файлів потрібно оголосити перевірку на визначення довільного заголовку використовуючи директиву, наприклад, #ifndef FRACTALS_INCLUDED (якщо не оголошено FRACTALS_INCLUDED), а потім за умови справдження перевірки прописати його оголошення #define FRACTALS_INCLUDED, після чого можна писати основний код, однак в кінці обов’язково вказати кінець if конструкції з використанням синтаксису дерективи препроцесора #endif. Побудова основних фрактальних функцій також потребує значної кількості додаткових функцій, зокрема у вигляді стандартних Signed Distance Fields, тому було створено ще один файл під назвою SDF_Core.hlsl, що містив їх усі.

Хоч вхідною точкою основного шейдера є функція frag — піксельної частини шейдера, вона використовує й інші, зокрема Raymarching. Це найголовніша функція обраної методики графічної візуалізації. Вона шукає визначає перетин умовно випущеного променя з об’єктами визначеними функціями відстані та виводить відповідний результат (див. рисунок 2.12). Для її реалізації було створено цикл, кількість ітерацій якого залежать від значення ітерацій Raymarhing вказаного у вхідних параметрах шейдера, в якому визначається поточна точка променю, згідно цієї точки отримане значення передається як опорна точка для функції відстанні, результатом якої є числове значення відстанні. Із кожною ітерацією За умови що обчислена відстань отримана з функції відстанні менша за мінімальне значення відстанні, що може вважатись поверхнею (вказується також користувачем у вхідних параметрах) то цикл обривається й передається значення кольору, або є за умови що точка початку променя розташована на відстанні більшій за максимально визначену.

Для зручності передачі даних кольору відстанні та фігур, було створено стируктуру Figure, що містить поле дистанції об’єкта, його кольору, його можливості відбивати світло (булева зміна створена лише для відрізняння об’єкта сонця від інших) та булевої змінної на позначення чи фрактал є геометричним чи комплексним, оскільки для цих двох типів у візуалізації тривимірної форми, реалізовано різне накладання обраної текстури. Для геометричних фракталів, текстура накладається з урахуванням позиції поверхні відносно сторін простору та UV— кординат прив’язаних до вершин. У свою чергу для комплексних фраклатів накладання текстури використовується лише за рахунок врахування UV— кординат та нормалі, визначеної під час обчислення відповідною функцією GetNormal.

Функція шейдеру GetNormal Обчислює номаль (вектор, що перпендикулярний до поверхні у вказаній точці) поверхні у конкретній точці, для цього використовується метод апроксимування градієнта, який є відомим і класичним способом обчисленння нормалей в шейдерах з функціями полів

відстані. Інакше кажучи, суть такого методу в тому, що немаючи фізичної геометрії, а лише функцію відстанні об’єкта. Проблематика такого підходу полягає в тому, що для кожної осі викликаються нова функція відстані, однак із заміщеним конкретним занченням похибки відповідної осі в обох напрямках. З метою оптимізації, використовувалось одностороннє апроксимування. В кінці, отримані значення похибки нормалізуються, вигляд описаної вище функції наведено в лістингу 4.1.

Лістинг 4.1 — Функція визначення нормалі поверхні

```
float3 GetNormal(float3 p)
{
   float2 offset = float2(1e— 3, 0);
   float3 norm = GetDist(p).dist
               — float3(
                  GetDist(p — offset.xyy).dist,
                  GetDist(p — offset.yxy).dist,
                  GetDist(p — offset.yyx).dist);

   return normalize(norm);
}
```

Замість конкретного написання кординат у форматі float3(x,y,z), використовувався скорочений синтакис вказування порядку значень кординат осей. Як було зазначено вище, замість реалізації різниці між зміщеними по трьох осях в одну сторону значень відстаней та в іншу, використовувалась різниця між звичайним значенням відстанні та значенням із зміщенням в одну сторону відповідно до кожної осі.

Функція GetDist() в свою чергу реалізує використання функцій відстанні фракталів, визначення яких , в залежності від вазаного індентифікатора фрактала, перевіряється в головному операторі switch— case. Особливості підходу до реалізації кожних типів фракталів, являються в тому, що вище згадані на початку розділу 3.1 фрактали у двох з половиною вимірах, візуалізуються з додаванням світової горизонтальної площини XZ. Це зроблено для полегшення находження в просторі, для спостерігання за відповідною фактично двовимірною

візуалізацією фрактала в тривимірному просторі. Однак повністю тривимірні реалізації фракталів візуалізуються без об’єкта просторової площини, оскільки використання навіть простого додаткового об’єкта в методиці Raymarching вимагає збільшення кількісті ітерацій променів, через зміщення світової відстанні через змішування між об’єктами, та подальшого їх скривлення, а вони вимагають більше системних ресурсів через підвищену деталізацію структури. Також, додатковим об’єктом на вибір користувача, може бути умовна візуалізація яскравої сфери, що позначає сонце у відповідній позиції вказані користувацькими налаштуваннями.

4.3 Додавання ефектів освітлення

Умовна візуалізація сонця та відповідних обчислення променів світла й тіней, дозволяє краще візуалізувати перебування в тривимірному просторі. Для кращого спостереження геометрії візуалізованих фракталів, створено функцію SunAnimation, що змінює умовну змінну позиції сонця з використанням обчислювального значення кута, що залежить від часу та обчислень тривимірного вектора. Вісь X позиції сонця обчислюється з використанням функції косинусоїди, та відповідного кута, вісь Y для реалізації коливань вгору вниз використовує синусоїду, а вісь Z для реалізації горизонатльного колаЮ що доповнюється косинусоїдою X, обчислюється з використанням синусоїди також. Для вертикальної вісі, значення кута обчислюється з урахуванням добутку з— зі вказаним значенням частоти, а резільтат значення коливання множиться на відповідне значення амплітуди. Загальний принцип руху сонця, за відповдіною формулою, представлено на рисунку 4.1.

Також для запровадження імітації простору, було реалізовано обчислення променів світла, тіней та затіннення. Функція затінення використовує інтенсивність затінення точки на поверхні в залежності від її поверхні та вектору напрямку до джерела світла. Отримані значення лежать в діапазоні від — 1 до 1, оскільки у формулі використовується скалярний добуток між вектором напрямленим до позиції світила та нормалею поверхні (рисунок 4.2). В коді ці

значення було обмежено до діапазону від 0 до 1, що створювало ефект плавного затінення поверхні при множенні на основний колір світла поверхні.

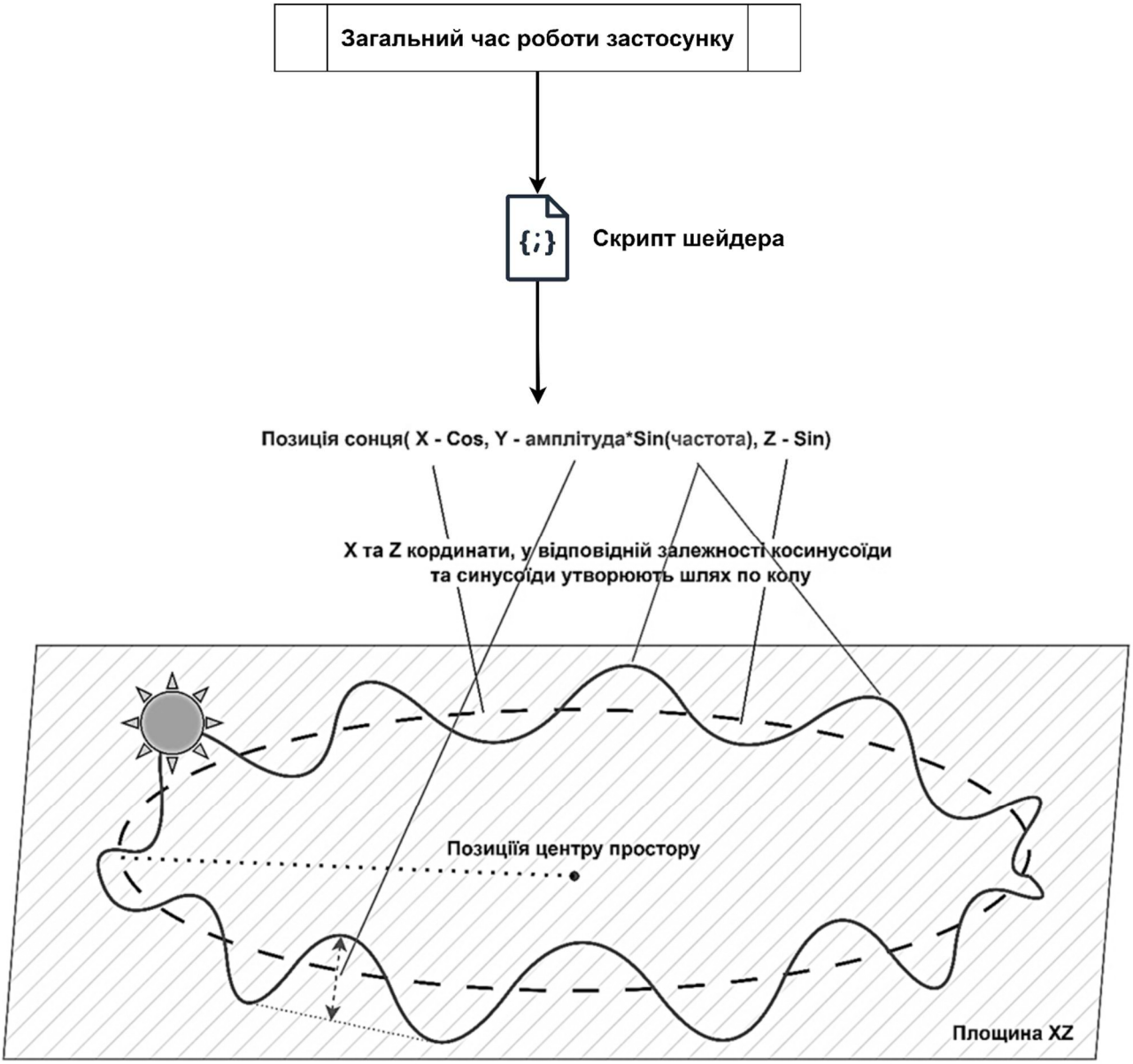


Рисунок 4.1 — Візуалізація руху сонця

Візуалізація руху сонця в тривимірному просторі Raymarching у поєднанні з обчисленням затінення поверхні об’єктів на освнові їх нормалі та вектора напрямку світла, дозволяють створити реалістичну динамічну сцену, що дозволить краще побачити різні частини фрактальної структури, включно з генерованою поверхнею, в залежності від різної позиції променів світла. Крім того, застосування техніки глобального освітлення та корекції кольору може

додатково покращити сприйняття сцени, надаючи їй природності та глибини. Використання простих математичних дій над векторами дозволяє оптимізувати обчислювальні ресурси, підвищуючи продуктивність рендерингу без втрати якості візуалізації. Завдяки цим методам, можна досягти більш точного моделювання поведінки світла, що сприяє створенню реалістичного віртуального середовища.

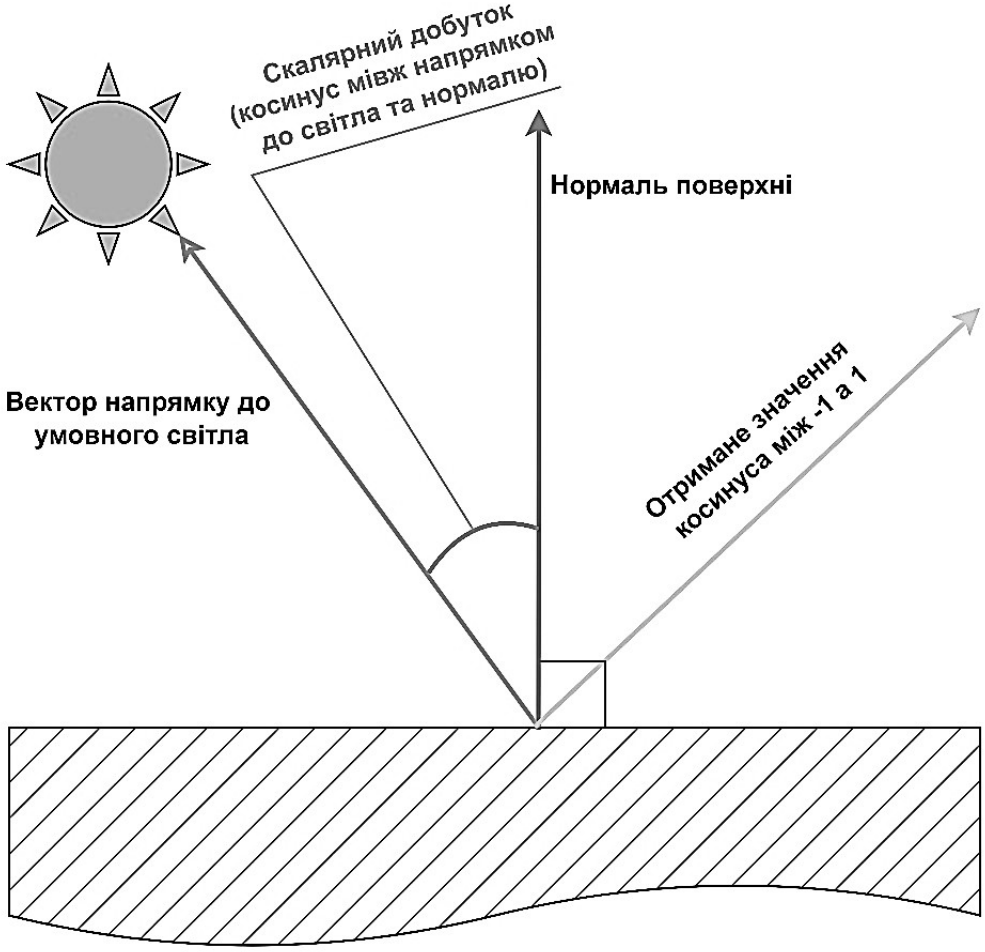


Рисунок 4.2 — Діаграма визначення значення затінення нормалі.

Однак затінення дозволає передати лише основну геометрію поверхні відсносно світила, тобто воно не реалізує конкретні тіні, тому було прийнято рішення створити функцію, що реалізовуватиме відмалювання тіней, як накладання кольору з градації сірого. Для такої реалізації, було створено умовні запуски променів у циклі, що залежатиме від кількості ітереацій Raymarching.

Кожен з променів перевіряє чи промінь напрямлений в напрямку джерела світла, відносно обраної точки знаходиться в тіні, для цього відбувається перевірка на перетин з поверхнею. Інакше кажучи дана функція не має плавного затінення в розрахованому результаті, а виводе остаточний колір (визначений попередніми налаштуваннями) у вигляді є перешкода в напрямку світла чи немає, відповідно колір темний (менше одиниці) чи одиниця (добуток на значення поверхні не змінить його). Перевірка також враховує відповідне вихідне значення перекриття поверхні, щоб потім реалізувати ще в одній функції обчислень відблисків перпендикулярних поверхонь лише з освітлених частин моделі. Для перевірки того чи промінь перетнув якусь перешкоду, викликається функція обчислення дистанції фігур сцени, яка приймає за параметр не просто точку, азначення верктоpя напрямку до світла від відповідної точки у просторі і якщо значення відстанні отримане функцією менше за мінімальну дистанцію до поверзні, то цикл обривається через break та виводиться відповідне значення тіні. На рисунку 4.3 зображена загальна діаграма принципу роботи методу.

Реалізація більш реалістичнішої поведінки сонця, передбачала створення відповідниї відблисків. Для цього усі точки нормалей поверхні, враховувались відповідно у скалярному добутку для пошуку кута відносно сонця, також було обраховано значення скалярного добутку між нормалею та поглядом камери. Потім, оскільки значення отримані лежать в діапазоні від — 1 до 1, то зважаючи що від’ємні значеннння це протилежність напрямку світла, а 1 це буквальна перпендикулярність до світла, то було обраховано умову відблисків. Суть умови полягає в тому, що значення кута свіла до нормалі поверні та значення камери над нормалю поверхні наблидаються до 1, то вони наближаються до перпендикулярності й тоді, ми можемо повертати не просто текстуру, а підняту в інтенсивності по яскравості текстуру, що буде імітувати відблиск сонця. Приклад результату застосування такої функції в порівнянні з її відсутністю, наведено на рисунку 4.4.

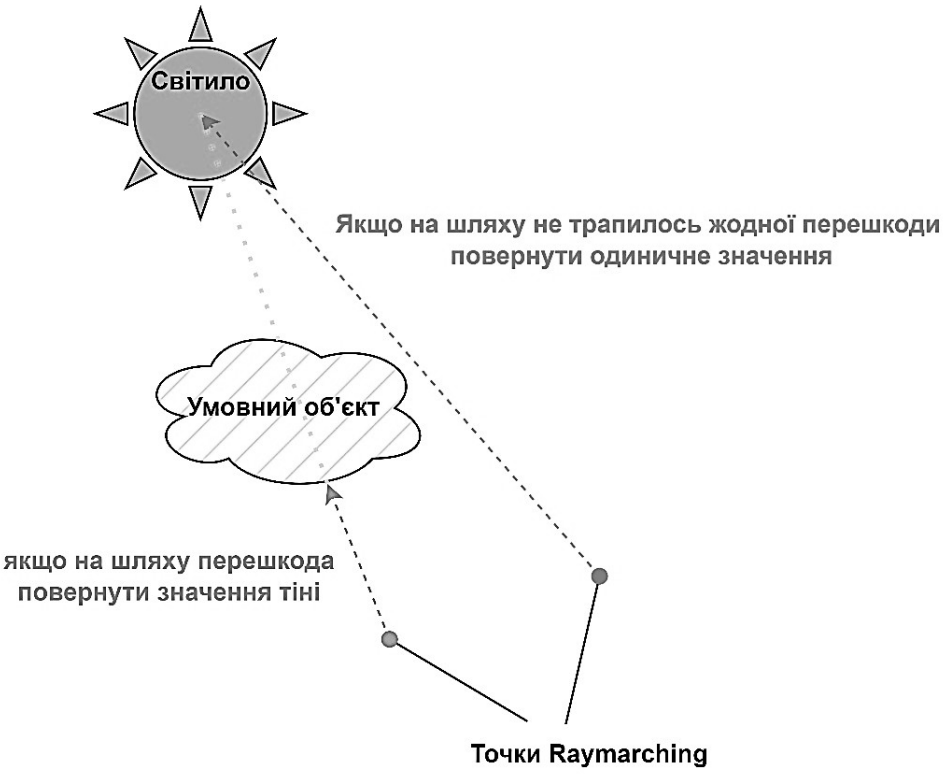


Рисунок 4.3 — Візуалізація функції обчислення тіней

Рисунок 4.4 — Результат використання функції відблисків

Щоб стфорити ефект присутності та масштабу генерованих структур, було також написано функцію для обчислення вінь’єтки, а також візуальний шейдер екрану для створення розмиття поза зоною фокусування камери огляду. Обидві реалізації використовують поняття довжини вектора, що застососовується до

UV— кординат прив’язаних до пікеселів екрану. Як відомо з підрозділу 2.2, UV— розгортка являє собою нормалізовані значення кординат відносно пікселей чи вершин моделі. Оскільки нулевим початком таких кординат вважається лівий нижній кут, UV кординати було зсунуто по обох осях на пів діапазону (+0.5) чим досягнено центрування початку кординатної площини відносно центру екрану. Звідси, при використанні довжини вектора (у мові HLSL — функція length), ми отримаємо відстань ід контурів до центру екрана з нормалізованими значеннями градації сірого. Перемноживши отримане значення на фактичний колір відображення шейдеру, що візуалізує простір із фракталами, ми отримуємо чітко виражене плавне затемнення відносно центру екрану. Аналонічно під час створення ефекту розмиття використовувалась довжина вектора кординат UV— розгортки екрану, аби виділити з використанням лінійної інтерполяції накладання шейдера розмиття (накладатиметься лише на контури екрану, навідміну від центра). Суть реалізації шейдера розмиття полягає в отриманні кольорового буферу екрану, зсуву його в певну сторону площини, та накладання на оригінальн екран, після накладання декількох буферів зсунутих на певне значення в різні сторони, утворюється візуальний ефект розмиття.

4.4 Налаголження зв’язку між користувачем та шейдером

Для забезпечення коректної взаємодії користувача запропонованого програмного забезпечення з основним шейдером візуалізації фракталів у тривимірному просторі, було реалізовано графічний інтерфейс та обробку гарячих клавіш.

Для побудови інтерфейсу в редакторі Unity, потрібно створити новий об’єкт у ієрархії сцени, що буде відповідати за основне полотно всього графічного інтерфейсу користувача. До цього об’єкта додаємо стандартні компоненти з бібліотеки рушія, як Canvas, Canvas Scaler та Graphic Raycaster (рисунок 4.5). Відповідно перший компонент під назвою Canvas, дозволяє визначити область відображення інтерфейсу, в ній було налаштовано тип рендерингу Overlay (рендер поверх усіх об’єктів сцени) та обрано відображення

усіх каналів шейдерів (оскільки інтерфейс мамиме навівпрозорий фон, і візуально краще буде продовження рендерингу основного шейдеру навіть на задньому фоні). Також було налаштовано компонент Canvas Scaler на Constant Pixel Size тип, що буде змерігти вказані піксельні розміри компонентів інтерфейсу незалежно від розміру екрану, однак враховуючи прив’язки відносних положень. У свою чергу Graphic Raycaster юудо додано для того, аби обрані нами певні елементи могли перехоплювати натиски, наводження чи інші події взаємодії з користувачем.

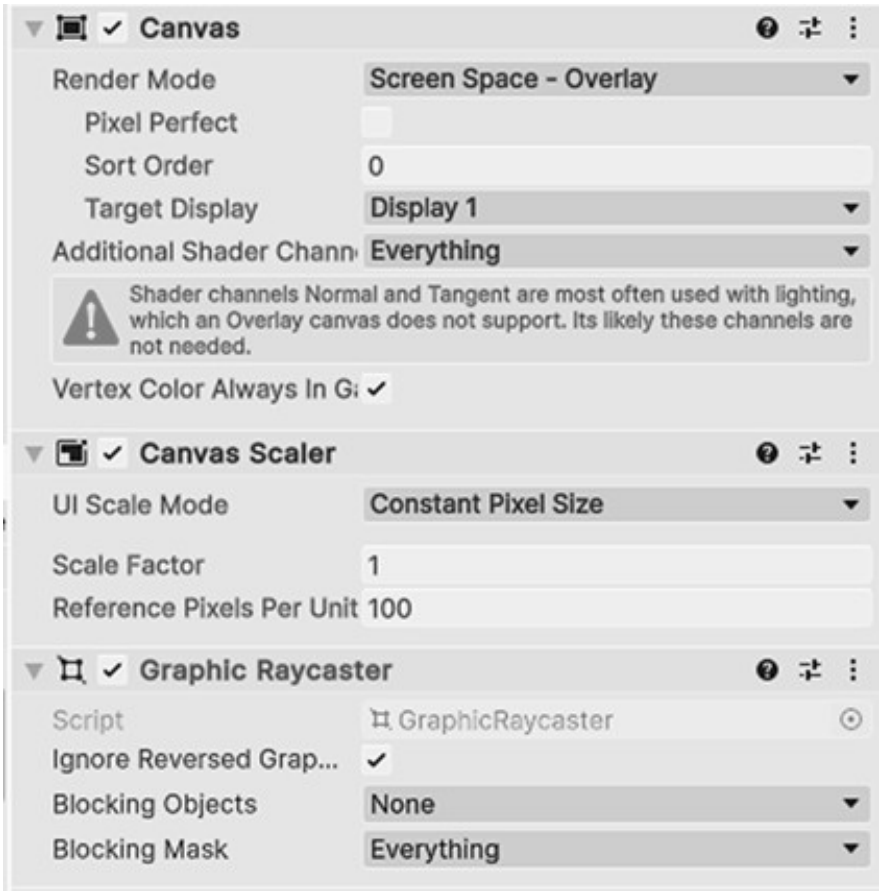


Рисунок 4.5 — Компоненти полотна інтерфейсу додатка

4.4.1 Головне меню

Головний інтерфейс програми передбачає лаконічне меню, що вміщує основні можливості програми, серед яких вибір та налаштування параметрів візуалізації різних типів фракталів з їх відповідними параметрами. Таке меню,

згідно розробленого макету на рисунку 4.6, має основний блок, розташований зліва на якому розміщений перелік фракталів та загальні налаштування шейдера в іншій. Фрактали планується розподілити в інтерфейсі за певними групами, наприклад за способом обчислення. Сам перелік фракталів буде мати статичний характер і міститиме одні з найпопулярніших прикладів фракталів. Планується додати допоміжні клавіші для переходу або відкриття додаткових діалогових вікон другорядної інформації чи дії.

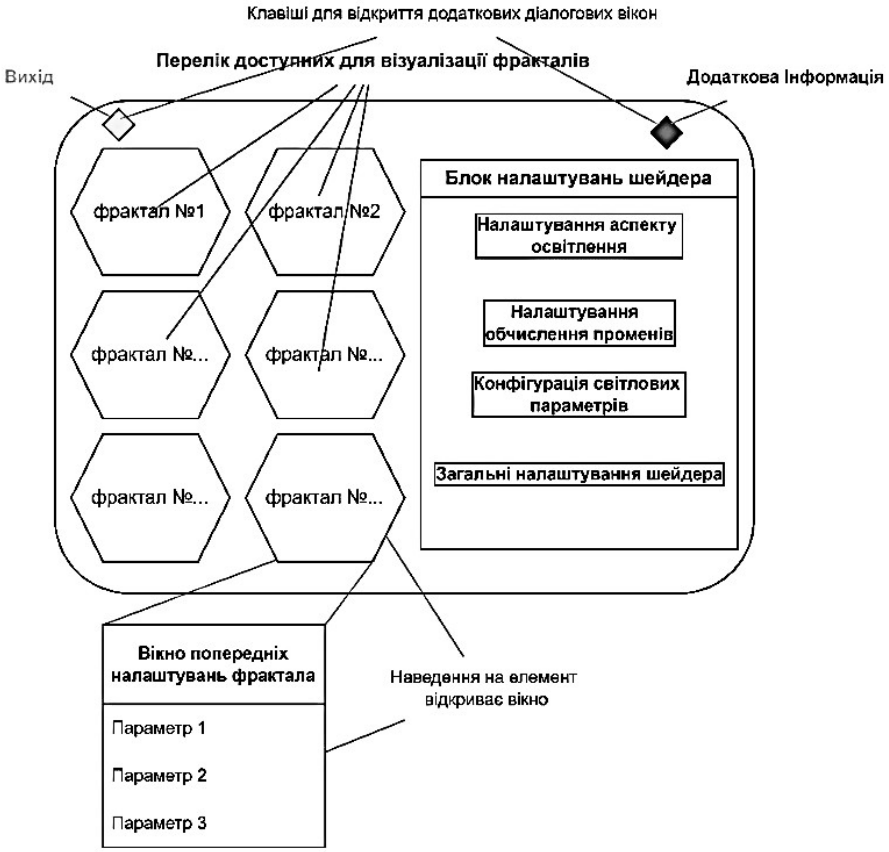


Рисунок 4.6 — Концептуальний вигляд інтерфейсу запропонованого додатка

Також, попри основний інтерфейс меню, додано й реалізовано додаткові під меню у вигляді вікна перед налаштувань обраного фрактала, вікна додаткової інформації щодо керування клавішами та вікна виходу з програми. Загальний вигляд реалізованого головного меню з деякими відкритимивікнами підменю показано на рисунку 4.7.

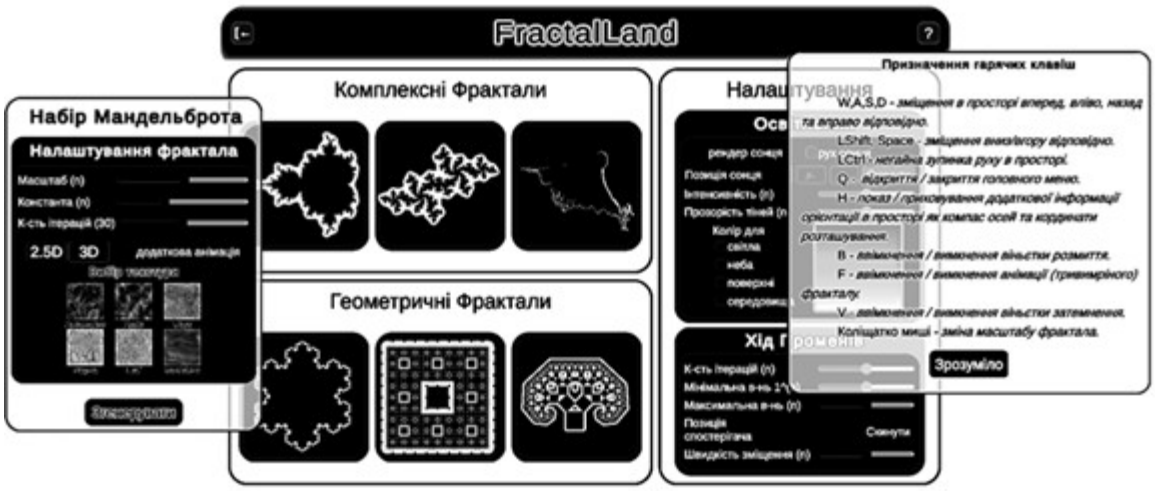


Рисунок 4.7 — Загальний вигляд основної частини головного меню програмного забезпечення

Як видно з рисунку 4.7, для зручності вибору кожного з фракталів, було реалізовано їх візуалізацію в на двовимірних текстурах у вигляді іконок переліку меню. Всі використовувані фрактали в запропонованому додатку, поділені за типами, а саме геометричні та комплексні, що дозволить краще орієнтуватися користувачу в інтерфейсі.

Вікно додаткових передналаштувань генерованого фракталу містить змінні параметри як масштаб, фрактальна констранта (використовується в більшості типів фракталів і зазвичай відповідає за різні значення, що можуть впливати на безпосередній результат генерації, зокрема кут нахилу чи доданок до обчислень), кількість ітерацій обчислення фрактальної структури, вибір типу візуалізації як тривимірну (передбачає повністю написаний під час роботи підхід до реалізації фракталу в тривимірному просторі з урахуванням самоподібності двовимірної версії) та двох з половиною вимірному просторі, що додає глибину до двовимірної візуалізації, а також місце для позначення активації анімації і вибору накладання текстури на поверхню. Варто хазначити, що під час зміни ітерацій та константних значень, іконки фрактальних відображень у меню, візуалізують ці зміни в двовимірному просторі, що дозволить передбачити можливий результат обчислень.

Правий блок осноного полотна меню, представляє детальні налаштування безпосередньо шейдера. Вони поділені на дві основних групи, а саме параметри освітлення та параметри обчислення променів. Це інтуїтивно дозволить швидше згнаходити відповідні налаштування та змінювати їх. Оскільки ці параметри є головними під час візуалізації, як от збільшення деталізації під час певного масштабування фракталу чи підвищення яскравості світла аби побачити приховані деталі, то всі налаштування головного меню доступні і під час використання режиму вільного польоту. Даний режим активується при генерації фракталу й згортає інтерфейс головного меню. Можливість зміни позиції в умовному тривимірному просторі шейдера, здійснюється через відповідні гарячі клавіші, що напряму змінюють параметри позиції спостерігача, яку можна змінювати й через інтерфейс.

Розділ налаштувань «Освітлення» містить частину з кольоровим призначенням простору, для зручності вводу якого було реалізовано шейдер кольорової палітри (рисунок 4.8). Для цього кольорову ґаму було стиснено відносно юв та перемножено на вертикальний градієнт сірого, що дозволив отримати світлі та темні відтінки кожного з кольорів. Також, у отриманому вільному місці після стиснення UV— кординат палітри, було розміщено чисту градацію сірого без будь— якого забарвлення. Щоб користувач міг бачити обраний курсором колір, під час вибору виводиться збільшений блок із забарвленням на яке вказує курсор. Колір забарвлення обчислюється відносно текстури зображення враховуючи послідовність обчислення самої текстури палітри. Щоб не запам’ятовувати попередні значення кольорів, призначені тій чи іншій частині середовища, було реалізовано фарбування в обраний колір відповідних чекбоксів.

4.4.2 Тривимірний компас

Під час режиму польоту, з вимкненим інтерфейсом головного меню, для кращої орієнтації в просторі було створено тривимірний компас, що вказував на умовні вісі простору шейдера, а також показу відносних кординат. Для цього було

створено окрему модель та розміщено на сцені подалі від основної моделі інвертованої сфери та камери. На додачу до моделі було створено нову камеру, що прямо напрямлена на відповідну модель. В налаштуваннях камери було вимкнено усі можливі шейдери та згладження, а також встановлено реакцію лише на моделі осей, на все інше середовище реагувати виведенням яскравого рожевого кольору (рисунок 4.9).

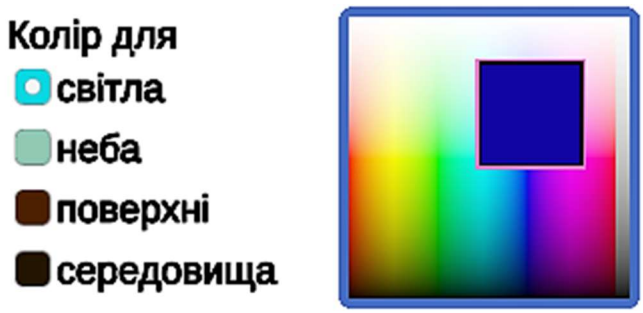


Рисунок 4.8 — Зовнішній вигляд реалізованої палітри кольорів

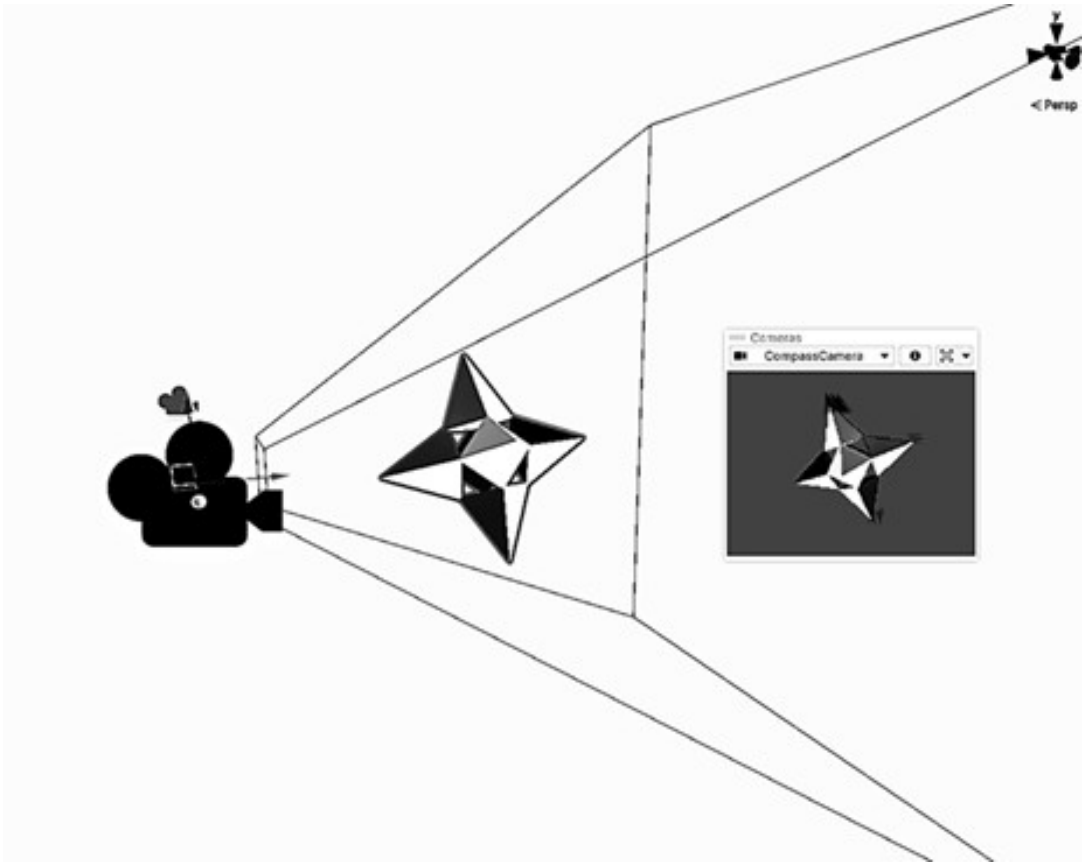


Рисунок 4.9 — Вигляд реалізації тривимірного компасу на сцені Unity

Закінчивши налаштовувати камеру, було створено рендер-текстуру обмеженої роздільної здатності (500 на 500 пікселів) восьмибітного кольорового формату. В налаштуваннях текстури було обрано тип фільтрації Point (це повна пікселізація без присутності жодних фільтрів згладжування), для реалізації шейдера, про який буде описано згодом. У інтерфейсі об’єкта камери компасу, забираємо вивід на головний екран, та обираємо вивід на текстуру, вибравши створену рендер— текстуру. Тепер, створивши елемент інтерфейсу RawImage, ми можемо відображувати зняту камерою сцену в реальному часу у вигляді відображення рендер— текстури.

Проблема відображення компаса, полягає в тому, що оскільки ми використовуємо ігнорування усього середовища, замінюючи його на певний колір, то нам потрібно його позбутись оскільки він не несе ніякої корисної інформації, а також загромаджує інтерфейс. Тому було створено шейдер, що зчитуватиме кожен піксель текстури, в нашому випадку ми знаємо який колір замінюємо, а саме яскраво рожевий, тому при умові що конкретний піксель рівний тому кольору, забороняємо на рівні графічного процесора, його рендирити, що в результаті реалізує своєрідний альфа канал прозорості для рожевого фону. Звідси, оскільки до уваги береться кожен піксель окремо і значення кольору в ньому повинні бути точними, ми не використовуємо ніяких згладжень як наприклад bilinear чи trilinear, тому що вони створюють проміжні відтінки між різкими переломами кольорів на пікселях, для імітації візуалізації більшої точності. Такий підхід дозволяє реалізувати прозорість для текстури, що є прямою трансляцією знятого з камери. Матеріал, що формується шейдером було додано до елемента інтерфейсу RawImage з відповідним відображенням рендер— текстури, що в результаті додало показ тривимірного компасу, яка показана на рисунку 4.10. Показ кординат здійснюється через зчитування під час руху, параметру зсуву позиції спостергіача в шейдері. Для обертання самого компаса відносно його камери, було написнано скрипт, який залежить від

основної камери й обертається з використанням відповідних методів обертання кватерніонами Unity.

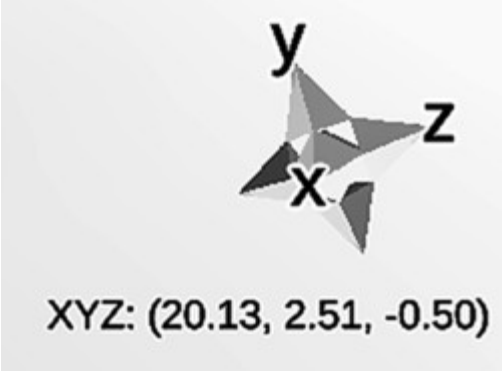


Рисунок 4.10 — Зовнішній вигляд тривимірного компасу в інтерфейсі

Тривимірний компас попри показ обертання моделі осей відносно камери огляду користувача, надає умовні півдписи головних додатніх світових осей як X, Z та Y. Це було реалізовано шляхом розміщення пустих об’єктів, що знаходяться на кінцях моделі відповідних осей. Після чого під час обертання камери, позиції елеменьів зчитувались і передавались до відповідного скрипта. Той у свою чергу керував уже існуючими текстовими елементами в інтерфейсі, змінючи їх позицію на текстурі, відносно конвертованих світових кординат зчитаних камерою. За умови що відстанні від камери до пустого об’єка края осі дорівнюють більше ніж, відстань між камерою і центром моделі (при розвороті осі в протилежних до камери напрямках), отримувалась відповідна дельта значення відхилення, котре потім конвертувалось скриптом у альфа значення каналу тексту. Такі махінації дозволили імітувати наявність тексту з двовимірного інтерфейсу у тривимірному й при цьому дозволили зручно орієнтуватися в просторі під час режиму вільного польоту. Для прикладу, розглянемо випадок, коли усі головні осі розташовані до на камери в різних положеннях (рисунок 4.111). В цьому випадку вісь Y є найближчою до камери і знаходиться на відстані менші ніж відстанні між камерою та центром моделі, тому її альфа канал має одиничне значення (повна непрозорість), навідміну від цього вісь Z знаходься частково далі за межу початку зміни альфа каналу

(відстань трішки більша за відстань між камерою й центрмо моделі), що породжує ледь помітне зниження значення альфа відповідного текстового елемента. Вісь X знаходиться на рисунку 3.35 найдалі, від усіх розглянутих, тому має найнижче значення альфа каналу, що рівне близько 25%, воно зроблено не нулевим аби навіть при повному протилежному до вісі оберту, можна було її все ще побачити на компасі для полегшення орієнтації.

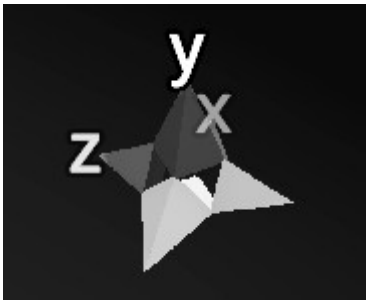


Рисунок 4.11 — Вигляд залежності альфа каналів осей від відстаней до камери

4.4.3 Прогрес-бар завантаження та меню виходу

Під час запуску застосунку, після вступної заставки рушія, відбувається завантаження сцени, що в нашому випадку триває не дуже довго, однак було реалізовано візуалізацію завантаження застосунку, де за бар прогрусу завантаження відповідала візуалізація фракталу Слова Фібоначчі (рисунок 4.12). Також, на додачу до цього було сторено тестову візуалізацію, що поєднує декілька типів фракталів операціями гладкого змішування й за рахунок високої деталізації, дозволяє оцінити користувачем спроможність його пристрою до значних навантажень, що можуть створюватись розробленим програмним забезпеченням.

Реалізація слова Фібоначчі, як і інших L— системних фракталів, потребує збереження кожної деталі під час крокування в ітеративному циклі. Потреба у великому збереженні данних для подальшого їх окремого читання та обргобки, робить такі фрактали майже неможливими до реалізації за рівнем деталізації як в комплексних чи графічних. Тому, з метою тестування, було розроблено такий

тестовий шейдер, що зберігає дані кожної частини фракталу (відрізка) у вигляді бітів безнакового цілочисельного числа. А для того аби не прощатись із потраченими на розробку ресурсами, шейдер даного фракталу було використано для прогрес— бару завантаження.

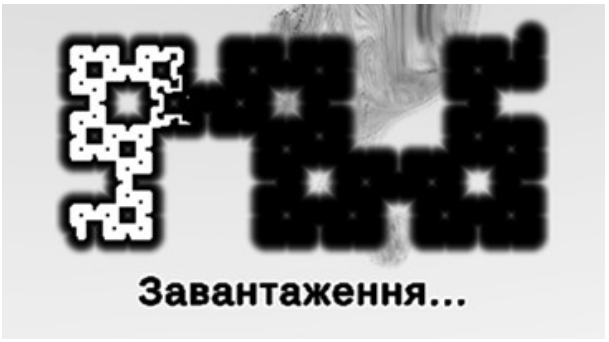


Рисунок 4.12 — Прогрес-бар для початкового завантаження головного меню

Фіксація натисків гарячих клавіш, передбачає обробку й подальшу зміну тих чи інших параметрів шейдера. Тому окрім вище згаданого пересування по умовному просторі Raymarching, було додано специфічні гарячі клавіші для активації шейдером ефекту віньєтки, роміття, масштабу фрактала за рахунок використання коліщатка миші та запуску анімації з використанням відповідної клавіші.

Під час спроби натиснути на клавішу зі знаком виходу, з’являється діалогове вікно, що дозволяє користувачу впевнетись у власних намірах і випадково не закрити застосунок і втратити усі налаштування. Дане діалогове вікно передбаче опцію відміни виходу та його погодження (рисунок 4.13).

Рисунок 4.13 — Діалогове вікно виходу з додатку

4.5 Тестування продуктивності

Під час розробки запропонованого програмного забезпечення, не менш важливим фактором враховувалась оптимізація, що була реалізована в більшості аспектах застосунку, починаючи від методу обчислення й візуалізації фрактальних структур і закінчуючи обмеженням стандартних елементів інтерфейсу на відмалювання та реакції на події користувача.

Оскільки інноваційність застосунку полягає в зменшеному використанні ресурсів сучасних існуючих персональних комп'ютерів, то було проведено тестування продуктивності на двох пристроях різних цінових сегментів, однак із комплектуючими та збірками п'яти й шести річної давності. В мірках розвитку цифрових технологій, такий часовий проміжок є досить великим, тому за умови успішних результатів, можна вважати, що програмне забезпечення буде показувати ще більш продуктивні результати на сучасних персональних комп'ютерах чи ноутбуках.

4.5.1 Тестування продуктивності на бюджетному ноутбуці

Для продуктивності було обрано досить застарілий і слабкий ноутбук 2019 років випуску, Lenovo ideapad 3i 15IML05 з розширенням екрану 1920x1080, частотою монітору 30 Гц та графічною картою nVidia GeForce MX 130 з об'ємом пам'яті 2048МБ. Він використовує центральний процесор Intel Pentium Gold 6405U з кількістю ядер — 2, та номінальною частотою 2.4GHz. Модулі ОЗП, обраного пристрою налічують загальну пам'ять у розмірі 20ГБ з тактовою частотою 2.5 GHz, а операційна система розміщена на швидкому SSD диску типу M2 з номіналом пам'ятті у 500ГБ. Пристрій був випущений понад 6 років тому та вважається слабким у співвідношенні з іншими бюджетними пристроями того часу.

Початковою перевіркою продуктивності пристроя буде зняття відповідних показників програми диспетчер задач під час відсутності навантаження (рисунок 4.14). Оскільки процесор має лише два ядра, а пристрій має операційну

систему Windows 11, то навіть при відсутності фонової роботи більшості програм навантаження на центральний процесор не складає менше 37% завантаженості. В свою чергу базові програми використовують 5.5ГБ ОЗП пам’ятті, а диск з операційною системою наватажується мінімум на 5% для основних операцій з файлами, як читання директорії робочого стола. Відповідно графічний процесор, навантажений на 3%, що свідчить про його непервну звичайну роботу з візуалізації.

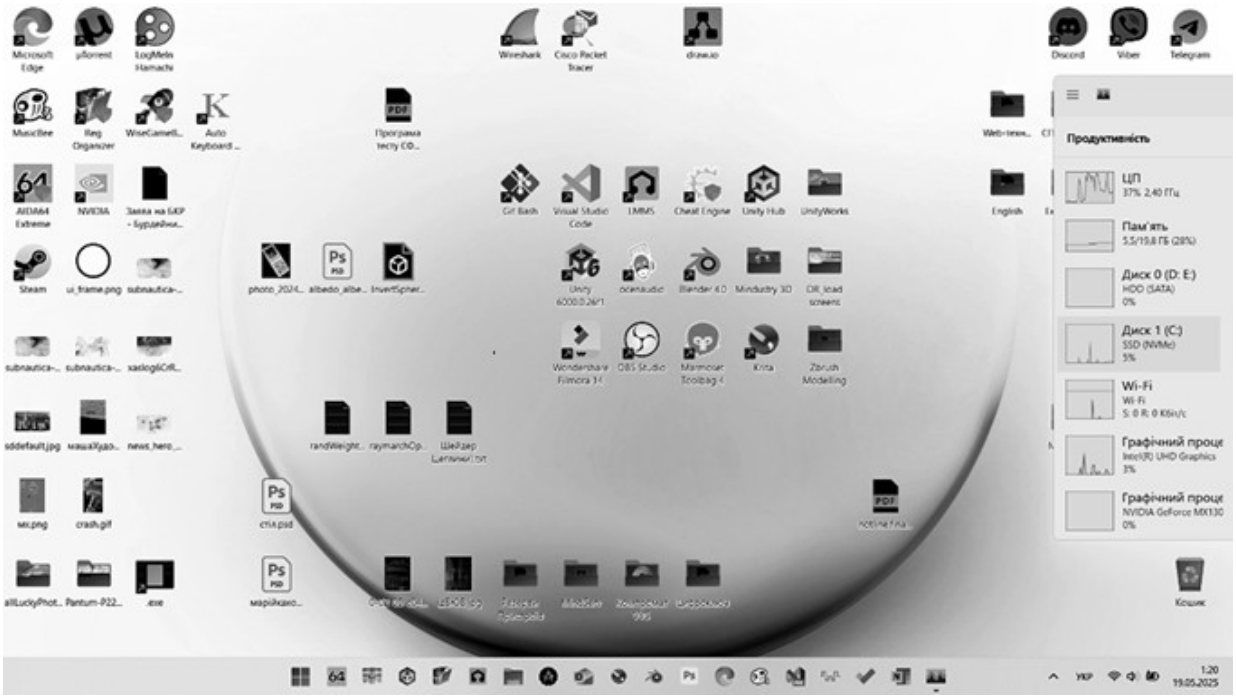


Рисунок 4.14 — Продуктивність ноутбука без навантаження

Слідуючим етапом буде запуск розробленого програмного забезпеченян з фракталів та тестування показників (зокрема кількості кадрів за секунду) під час значного навантаження з візуалізації динамічної високодеталізованої заставки. Ця візуалізація включає підібрані відповідно до ресурсів пристрою, майже максимальні навантаження на графічний процесор, що дозволяє пересвідчитись у тому, наскільки користувацький пристрій може сприймати великі потоки обчислювальних даних (рисунок 4.15).

Рисунок 4.15 — Навантаження на пристрій під час вступної заставки

Аналізуючи рисунок вище (див. рисунок 4.15), видно що попри повне навантаження на центральний процеосор, відеопам’ять даного пристрою, що налічує 2ГБ, використовується лише на 15%, в свою чергу для роботи самої програми, процесор постійно навантажений на 70 і більше відсотків. На противагу цьому, використання ОЗП пам’ятті навіть не збільшилося, ба більше того зменшилось на 1%, що свідчиться в даному випадку на основне навантаження гріфічного процесора, що дозволяє робити динамічні візуалізації обчислень великих обсягів даних. Кількість кадрів налічує стабільні 7, що є непоганим результатом при такому навантаженні, зважаючи на максимально можливі 30 кадрів на секунду в даному моніторі ноутбука.

Наступними тестуваннями буде перевірка продуктивності під час візуалізації одного з фракталів, наприклад Набору Джулії в двох з половиною та тривимірному просторі при стандартних налаштуваннях визначених програмою та стандартній позиції спостереження. Зображення відповідних візуалізацій фракталу показано на рисунках 4.16 та 4.17.

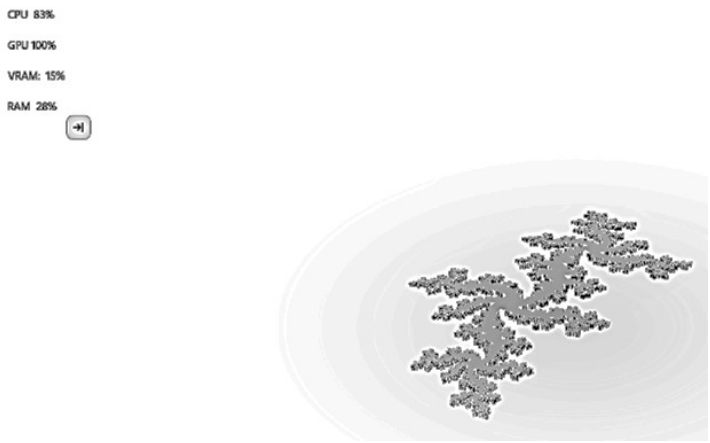


Рисунок 4.16 — Візуалізація Набору Джулії в 2.5 просторовому представленні

Зважачи на отримані дані з рисунку 4.16, бачимо, що показники навантажень на пристрої материнської плати майже не змінились, однак кількість кадрів на секунду збільшилась до 23, що є ідеальними робочими значеннями для даного ноутбука. Така візуалізація здійснює оптимальне навантаження на компоненти, що дозволяє навіть слабкому пристрою динамічно відтворювати складні структури.

Рисунок 4.17 — Візуалізація Набору Джулії в тривимірному представленні

Дані отримані з рисунку 4.17, вказують що під час кратного збільшення обчислювального навантаження, при переході до тривимірного представлення фракталу — кількість кадрів оновлення екрану, зменшилась вдвічі, що є нормою для такого збільшення деталізації.

Тепер, для достовірності експерименту з тестування продуктивності, було забрано візуалізацію будь— яких сруктур у відповідному просторі шейдера, однак відкрито інтерфейс головного меню, та вказано максимально можливі параметри для кращої деталізації результатів. Результатом є значний приріст кадрів за секунду до значення 28 (рисунок 4.18), що фактично є майже максимально можливим на даному пристрої.

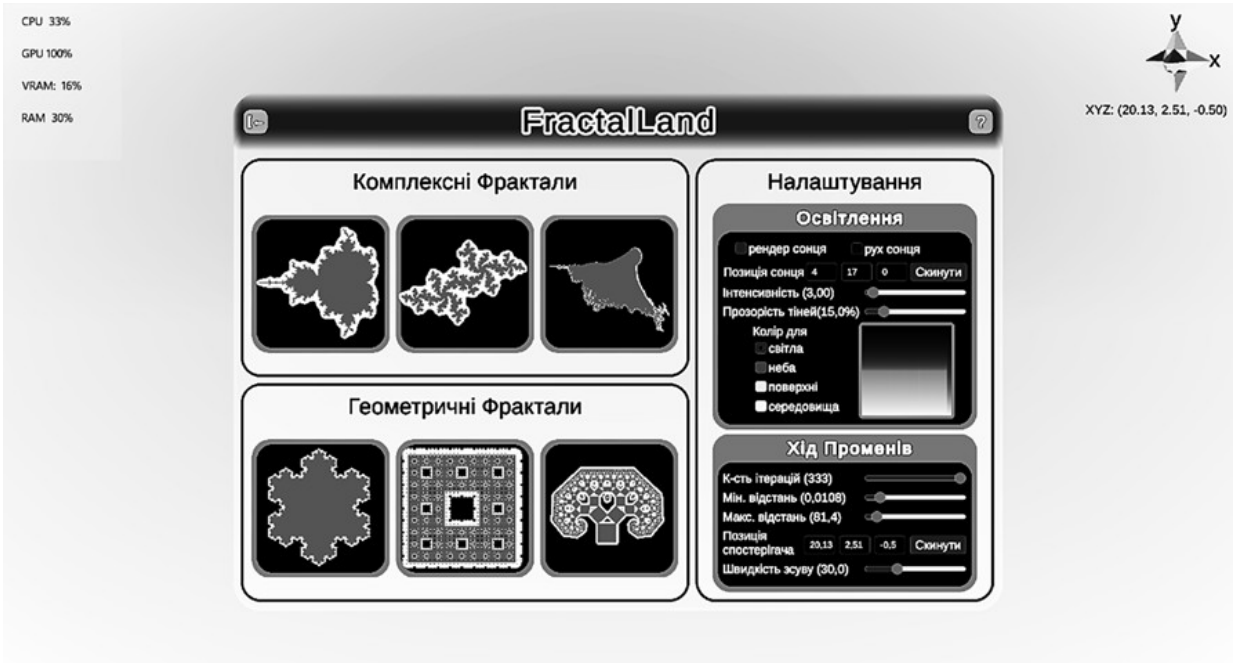


Рисунок 4.18 — Тестування продуктивності під час відстуності візуалізації фракталів

Хоча через використання шедеру, програмне забезпечення продовжує використовувати графічний процесор для передачі даних. У свою чергу наватаження на центральний процесор ще менше, ніж під час показу робочого стола без запущених фонових програм (33%). Ці всі показники вказують на те, що пристрій за будь— якого навантаження використовує графічний процесор на

100%, та попри те під час відстуності візуалізації фрактальних структур, програмне забезпечення не виділяє потрібні для цього ресурси, що робить його оптимізованим для роботи без та під час навантажень відповідними обчисленнями.

4.5.2 Тестування продуктивності на середньостатичнимоу персональному комп’ютері

Другим пристроєм для тестування було використано персональний комп’ютер користувацької збірки з монітором 1920x1080. Роль графічного процесора в ньому, відіграє Evga GeForce RTX 3060ti з об’ємом пам’яті 8ГБ. Процесор AMD Ryzen 5 5600 на 6 ядер з можливістю тактової частоти в 3.5GHz, а також ОЗП на загальну кількість пам’ятті 32ГБ із швикдістю 3600 МГц і диском пам’ятті для операційної системи номіналом 1ТБ.

Перед запуском тестування продуктивності, відкриємо диспетчер задач з відповідною вкладкою, та переглянемо стандартне навантаження на систему, за умови бездіяльності (рисунок 4.19). Під час стандартного навантаження на систему центральний процесор ледь навантажений (4%) як і жорсткий диск, оскільки вонує пересічні процеси операційної системи, в свою чергу Windows 11 та процеси за замовчуванням використовують 5.8ГБ пам’ятті озп. Графічний процесор навантажений не більш ніж на 1% виконуючи всю роботу з візуалізації графіки на моніторі, оскільки вбудованої графіки в центральному процесорі немає.

Запустимо розроблене програмне забезпечення для перевірки продуктивності під час значних навантажень вступної заставки, що показано на рисунку 4.20. Згідно цього етапу перевірки, бачимо що завантаженість графічного процесора перейшла фактично на повну, оскільки більшість обчислень використовуються саме ним. Із доступних 8ГБ графічної па’ятті використвоується лише 7%, що є дуже мало, зважаючи, що операцій над збереженням та передачою текстур майже не відбуваються, тому використовуєтсья в основному обчислювальна потужність процесора.

Рисунок 4.19 — Параметри продуктивності під час бездіяльності пристрою

Аналогічно з попереднім пристроєм, під час тестування розробленого програмного забезпечення обсяг ОЗП пам’ятті зменшився в порівнянні з мінімальними навантаженнями на робочому столі, оскільки система зосередила більшість ресурсів на додаток, які він використовує обмаль. Також, присутній рівень кадрів за секнду складає 68, що є ідеальним результатом для пристрою бюджетного сегменту з комплектуючими 5-ти річної давності. Це забезпечує приємну гладкість динамічної візуалізації з непомітними гальмуваннями кадрів для людського ока.

Як і в 4.5.1, проведемо тестування з активованою візуаліхацією фрактала Набору Джулії в двох з половиною вимірному просторі (рисунок 4.21). Згідно якої бачимо помітне зниження навантаження на центральний процесор (6%) у порівнянні з навантаженням під час візуалізації тривимірної фрактальної заставки запуску застосунка. Використання відеопам’ятті як і навантаження графічного процесора з оперативною пам’яттю не змінились (7%, ~100% та 11% відповідно), зважаючи на фіксоване використання ресурсів програмою . В той же час як кількість кадрів рендерингу обчслень, оброблених графічною картою,

налічує 255штук. Це значний приріст продуктивності й він виправданий, зважаючи на використання спрощеної візуалізації.

Рисунок 4.20 — Тестування під час запуску застосунка

Рисунок 4.21 — Тестування візуалізації Набору Джулія

Тепер перейдемо до тривимірно представлення фрактала Набору Джулії у розробленому програмному забезпечені, запущеному на тестовому персональному ком’ютері користувацької збірки (рисунок 4.22).

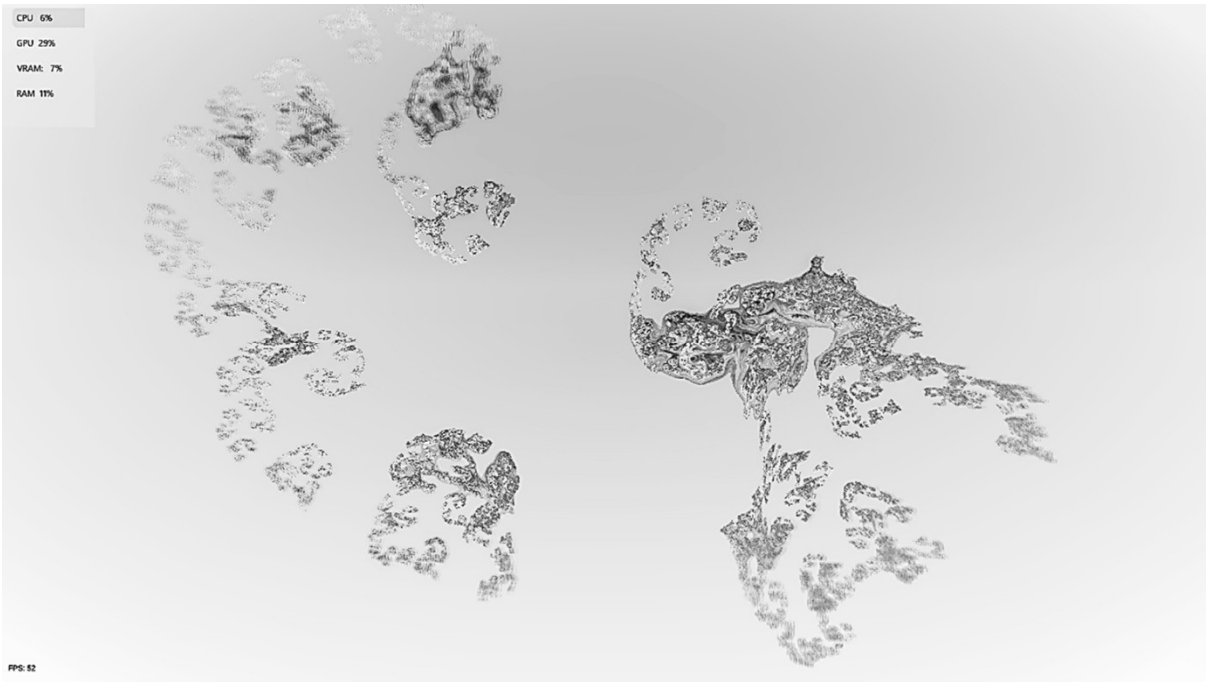


Рисунок 4.23 — Візуалізація фрактала Набору Джулії в тривимірному просторі

Під час приросту навантаження в тривимірнорму просторі (див. рисунок 3.46), обчислення Набору Джулії збільшили навантаження на графічний процесор, через що кількість кадрів оновлення саягає 52, що є хорошим результатом, зважаючи на високу деталізацію та масштаб структури.

Фінальною перевіркою продуктивності є тестування на відсутність основного навантаження візуалізації фракталів у розробленому програмному забезпеченні з відкритим головним меню (рисунок 4.24). Це показує приріст навантаження під час відкритого застосунку, однак без будь— якої активної роботи і зважаючи на отримані результати в 675 кадрів на секунду, можна зробити висновок, що система майже не наванатажується й процесори мають час реагувати на зовнішні переривання інших процесів. Однак оскільки застосунок працює лише в повноекранному режимі, для забезпечення кращої продутивності

використання зокрема й графічного процесору, то цей процесор використовує всі свої обчислювальні потужності (90— 100%) для роботи.

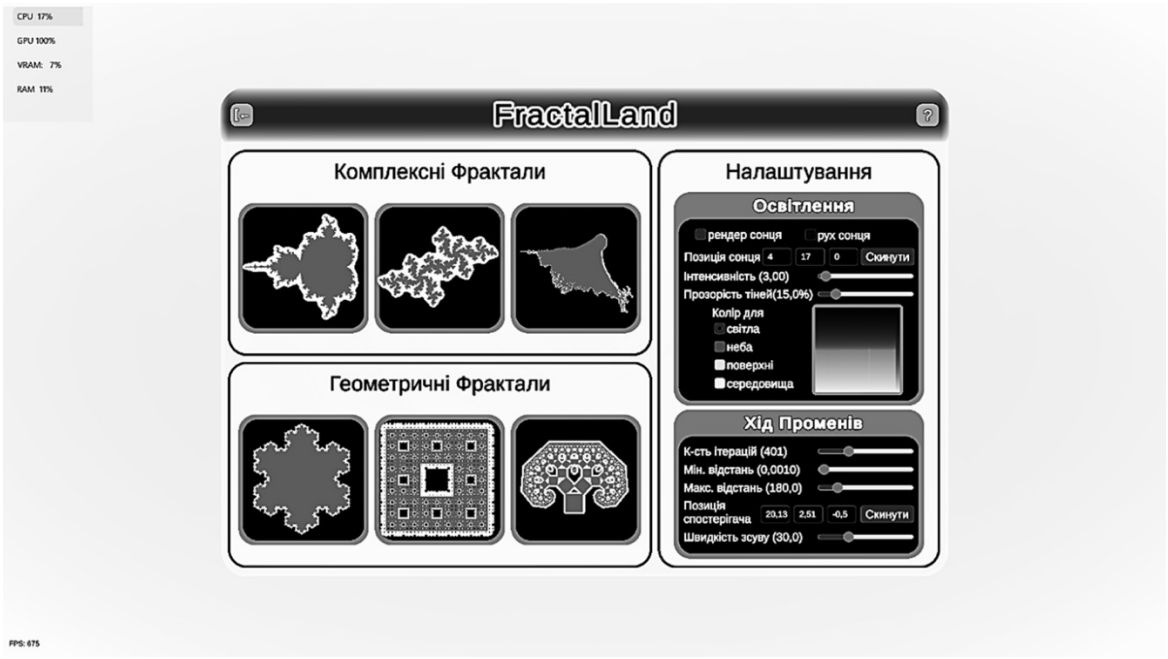


Рисунок 4.24 — Стандартні значення продуктивності під час бездіяльності обраноко персонального комп’ютера

4.5.3 Результати продуктивності розробленого програмного забезпечення

Під час тестування запропонованого додатку, що представляє візуалізацію певних фракталів у тривимірному просторі, було використано два пристрої з різними обчислювальними потужностями та технологіями, під час різноманітних умов використання.

Перший тест продуктивності включав ряд із перевірок з використанням ноутбука 2018 року, з надзвичайно низькими обчислювальним можливостями. Однак, перевірка показала, що навіть за використання вбудованої графіки в материнську плату та звичайного бюджетного процесора та графічної карти, програмне забезпечення працювало вірно з повноцінною та довершеною візуалізаціїю представлених фрактальних структур. Середній значення кількості обчислених процесором кадрів, складає 21 з 30 можливих на моніторі.

За період другого тестування використовувався персональний комп’ютер користувацької збірки з компонентами 2020 року, що показав відмінний високий результат продуктивності при низьких та великих обсягах навантажень обчислень. Варто зазначити, що графічний процесор, попри мінімальне використання доступної пам’ятті, використовувася на максимум, тобто запропоноване програмне забезпечення використовує усі обчсилювальні потужності графічної карти, незважаючи на навантаження аби відтворити гладку й динамічну візуалізацію. Після проведення тестувань з різними обсягами навантажень, було визначено, що даний тип простроію мав середньо— обчислену гарфічним процесором частоту в 100 кадрів з 180 можливих на моніторі виводу.

Закінчивши тестування, можна дійти висновку, що запропоноване програмне забезпечення є адаптивним відносно різних типів пристроїв, та показує високодеталізовані динамічні сцени з візуалізації фрактальних структур в тривимірному просторі із оптимальним використанням доступних ресурсів системи.

ВИСНОВКИ

Під час виконання дипломної кваліфікаційної роботи, було проведено аналіз характеристик фрактальних структур та способи їх побудови. Зокрема було розкрито поняття фракталів, їх використання у сучасних технологіях інженерії та відомих способів їх візуалізації. Також було оглянуто аналоги програмного забезпечення, що надають змогу візуалізувати фрактали й проведено їх порівняльну характеристику переваг і недоліків відносно запропонованого програмного забезпечення.

Для побудови застосунку з візуалізації фрактальних структур у тривимірному просторі було оглягуто існуючі рушії пришвидшення розробки, серед яких через ряд переваг, зокрема високої продуктивності, було обрано Unity. Опираючись на цей вибір, основною мовою побудови архітектури та зв’язків між компонентами в програмі стала C#. Було детально розглянуто основні аспекти мови програмування HLSL та поннять їх застосування для розробки осноного шейдера. Не менш важливим було розглянути розподіл фракталів за способами їх обчислень, зокрема ітеративні, рекурсивні, комплексні та з L-системи. Останні з попередньо названих типів є фактично неможливими до реалізації в шейдерах у з рівнем деталізації, як у інших.

У ході розробки програмного забезпечення було розроблено метод імітації тривимірного простору та візуалізації фракталів. Розроблений шейдер, що візуалізує всі фрактальні структури за основу бере методику Raymarching для обчислення крокування променів та візуалізації результату їх перетину з простором відносно кожного пікселя екрана. В свою чергу реалізація фракталів у імітованому тривимірному просторі залежала від типу його обчислення, зокрема фрактали, що спитраються на закономірності комплексних чисел на площині, було переформовано до тривимірного простору за рахунок сферичних кординат. Геометричні типи фракталів було реалізовано з методами поєднання тривимірних функції відстанні з використанням ітерацій та дублюванням абсолютних значень простору в методиці Raymarhing.

Попри розробку основного методу тривимірної візуалізації фракталів, було розроблено й альтернативний. Під час його опрацювання, було виявлено значну втрату продуктивності підходу, через що, його подальша розробка була припинена.

Оскільки запропоноване програмне забезпечення має як освітні, так і мистецькі напрямки використання, було додано графічні доповнення до загальної шейдерної візуалізації, як: конфігурація колірної палітри навколишнього середовища, налаштування освітлення, додавання відблисків до поверхні, системи затінення та тіней, ефекти вінь'єтки та розмиття.

Забезпечення зв'язку між користувачем та основним шейдером програми, було реалізовано через графічний інтерфейс, що дозволяє налаштувати всі вхідні параметри для візуалізації фрактальних структур. Зокрема, було додано можливість оперування головними параметрами через клавіатурні гарячі клавіші, їх комбінації та управління мишею.

Було проведено побудову структури основного шейдеру з використанням підходу тривимірної візуалізації — Raymarching. Не менш важливою була розробка архітекутри та класових взаємозв'язків у програмі. По завершенню реалізації усіх аспектів розробленого методу з візуалізації тривимірних фракталів, було проведено тестування продуктивності застосунку на пристроях із різними системними потужносями. Під час тестування було відзначено достатню продуктивність та відносно застарілих пристроях, що означає низьку потребу до обчислювальних можливостей орієнтованих на дане програмне забезпечення комп'ютерів.

У підсумку було створено програмне забезпечення з візуалізації фракталів у тривимірному просторі, що забезпечує значний рівень продуктивності, попри високодеталізований рендер динамічних сцен з-зі складними математичними структурами. Отже всі поставлені завдання були вирішені і таким чином мета роботи була досягнута.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мандельброт Б. Фрактальна геометрія природи. Нью-Йорк: W. H. Freeman and Co., 1982. – 468 с. ISBN 0-7167-1186-9.

2. Азаров О. Д., Богомолов С. В., Швець С. І. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Системне програмування»). Вінниця: ВНТУ, 2023. 105 с.

3. Мандельброт Б. Фрактали: форма, випадковість і розмірність. Сан-Франциско: W. H. Freeman, 1977. – 365 с. ISBN 0-7167-0473-0.

4. Бурдейний О. В., Черняк О. І. Програмне забезпечення для тривимірної візуалізації фракталів. Тези доповіді на LIV Всеукраїнській науково — технічній конференції факультету інформаційних технологій та комп'ютерної інженерії. Вінниця: ВНТУ, 2025. – 3 с.

5. Фрактальна геометрія в природі. Частина 2. Запорізький національний університет, 2025. URL: https://moodle.znu.edu.ua/pluginfile.php/.

6. Мандельброт Б. Фрактали і хаос: множина Мандельброта та інші чудеса. Нью— Йорк: Springer, 2004. – 308 с. ISBN 978-0387201580.

7. Функція Веєрштрасса. Вікіпедія — Вільна енциклопедія, 2024. URL: https://uk.wikipedia.org/wiki/Функція_Веєрштрасса.

8. Крива Коха. Вікіпедія — вільна енциклопедія, 2025. URL: https://uk.wikipedia.org/wiki/Крива_Коха.

9. Кітова Т. Парадокс берегової лінії: чому довжину океанічного узбережжя ніхто не може виміряти точно. Фокус, 12 листопада 2024. URL: https://focus.ua/uk/technologies/678193-paradoks-beregovoji-liniji-chomu-dovzhinu-okeanichnogo-uzberezhzhya-nihto-ne-mozhe-vimiryati-tochno.

10. Мандельброт Б. Яка довжина берегової лінії Британії? Статистична самоподібність і дробова розмірність. Science. — 1967. – Т. 156, № 3775. – С. 636— 638.

11. Scruss. The Mandelbrot Set, before Mandelbrot (Множина Мандельброта до Мандельброта). We Saw a Chicken, 7 липня 2020. URL: https://scruss.com/blog/2020/07/07/the-mandelbrot-set-before-mandelbrot/.

12. Бенуа Мандельброт. Molecular Frontiers (Молекулярні Кордони), 20 грудня 2022. URL: https://molecularfrontiers.org/content/benoit-mandelbrot.

13. Тейлор Р. П., Міколіч А. П., Ньюбері Р., Деттманн К. П., Фромхолд Т. М. Фрактальні транзистори. Semiconductor Science and Technology. — 1997. – Т. 12, № 11. – С. 1459. DOI: 10.1088/0268-1242/12/11/023.

14. Бенхадда О., Ахмад С., Саїх М., Чаджі К., Реа А., Гаффар А., Хан С., Алібахшікенарі М., Ліміті Е. Компактна широкосмугова антена з фрактальними слотами Віцсека для застосувань WLAN та WiMAX. Applied Sciences. — 2022. – Т. 12, № 3. – С. 1142. DOI: 10.3390/app12031142.

15. Алізаде М., Хоссейнзаде Х., Мехрзаді Х., Ганджі Д. Д. Дослідження LHTESS, заповненого гібридним нанопідсиленим PCM з фрактальним перерізом Коха в присутності теплового випромінювання. Journal of Molecular Liquids. — 2019. – Т. 273. – С. 414-424. DOI: 10.1016/j.molliq.2018.10.049.

16. Фрактальний сигнал. ScienceDirect, 2018. URL: https://www.sciencedirect.com/topics/engineering/fractal-signal.

17. Мандельброт Б., Гадсон Р. Л. (Не)поведінка ринків: фрактальний погляд на ризик, руйнування та винагороду. Нью-Йорк: Basic Books, 2004. – 328 с. ISBN: 978— 0465043552.

18. Фрактальне програмне забезпечення. Fractal Foundation, 2025. URL: https://fractalfoundation.org/resources/fractal-software/.

19. Сібоні Ж. Найкращі ігрові рушії, які варто розглянути у 2024 році. Incredibuild, 24 квітня 2024. URL: https://www.incredibuild.com/blog/top-gaming-engines-you-should-consider.

20. Верма С. К. Choosing the Right Programming Language. DevOps, 3 січня 2024. URL: https://devops.com/choosing-the-right-programming-language/.

ДОДАТОК А

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«____» _____2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи
«Програмне забезпечення для тривимірної візуалізації фракталів»
08–54.БКР.004.00.000 ТЗ

Науковий керівник: к.т.н., доцент
_____Черняк О.І.

Студент групи 1СП-21б
_____Бурдейний О.В.

ВНТУ 2025

1 Підставою для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 У сучасних програмних рішеннях для візуалізації фракталів переважають методи, що не підтримують динамічну генерацію у реальному часі. Більшість існуючих аналогів зосереджені на підході рендерингу статичних зображень чи покадрового відео з використанянм ресурсів центрального процесора. На відміну від класичних методів, ця розробка забезпечує миттєву візуалізацію фракталів у тривимірному просторі завдяки застосуванню шейдерної обробки на графічних процесорах сучансних персональних комп'ютерів. Це дозволяє користувачеві вносити зміни та одразу отримувати результат, без очікування тривалого рендеру;

1.2 Наказ про затвердження теми БКР.

2 Мета БКР і призначення розробки

2.1 Мета роботи — аналіз сучасного стану та основних підходів до візуалізації фракталів у тривимірному просторі, визначення їхніх недоліків щодо динамічного рендерингу в режимі реального часу та розробка програмного забезпечення, що забезпечує миттєву візуалізацію складних фрактальних структур шляхом оптимізованої шейдерної обробки на графічному процесорі для досягнення високої продуктивності навіть на слабких пристроях, з інтерактивною модифікацією параметрів фракталів у реальному часі;

2.2 Призначення розробки — програмне забезпечення для динамічної тривимірної візуалізації фракталів призначене для миттєвого формування складних фрактальних структур у реальному часі за допомогою шейдерної графіки, що забезпечує інтерактивне управління параметрами фракталів через графічний інтерфейс користувача та їх візуалізацію у тривимірному просторі.

3 Вихідні дані для виконання БКР

3.1 Проведення аналізу існуючих підходів та технологій візуалізації фракталів;

3.2 Розробка структурної та функціональної схем програмного забезпечення для динамічного рендерингу фракталів у режимі реального часу з використанням шейдерної обробки;

3.3 На основі структурних та функціональних схем здійснено розрахунок та моделювання алгоритмів обробки фрактальних даних у шейдері, а також написання скриптів для інтерактивної взаємодії з користувачем;

3.4 Виконання оптимізаційних розрахунків для доведення доцільності застосування GPU замість CPU для генерації складних фрактальних структур, обґрунтування продуктивності та економічної ефективності нової розробки;

3.5 Аналіз технічних обмежень, пов’язаних із ресурсоємністю шейдерної графіки, рекомендації щодо оптимізації коду та архітектури для забезпечення стабільної роботи на слабких пристроях;

3.6 Дослідження можливостей інтерактивного управління фрактальними параметрами (наприклад, зміна констант, глибини ітерацій, кутів проекції) у реальному часі з візуальним фідбеком;

3.7 Тестування реалізованого програмного забезпечення на різних апаратних платформах із фіксацією ключових показників продуктивності (FPS, час генерації кадру, завантаження GPU/CPU) з подальшим аналізом стабільності та адаптивності системи;

3.8 Розробка графічного інтерфейсу користувача (GUI) для інтерактивного редагування параметрів шейдера у реальному часі, з можливістю зміни значень констант, колірних схем, рівня ітерацій та інших атрибутів фрактальної візуалізації.

4 Вимоги до виконання БКР

Головна вимога — використати, як основний, метод рендерингу фракталів у реальному часі шляхом оптимізованої шейдерної обробки на графічному процесорі, що забезпечує миттєве формування сцени без необхідності довготривалих обчислень на центральному процесорі.

5 Етапи БКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи БКР

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1.	Вибір, узгодження та затвердження теми БКР	21.03.2025	21.03.2025	
2.	Аналіз літературних джерел. Попередня розробка основних розділів	22.03.2025	25.03.2025	
3.	Огляд існуючих програмних технологій для реалізації фракталів	26.03.2025	29.03.2025	
4.	Розробка основної методики реалізації імітації тривимірного простору	30.03.2025	14.04.2022	
5.	Створення двовимірних функцій фрактальних структур з їх оцінкою відстанні	15.04.2025	18.04.2025	
6.	Написання методик для візуалізації двовимірних структур у тривимірному вигляді	19.04.2025	23.04.2025	
7.	Розробка архітекутри програмного забезпечення	24.04.2025	25.04.2025	
8.	Побуда графічного інтерфейсу користувача та проведення тестування продуктивності	26.04.2025	27.04.2025	
9.	Оформлення ПЗ та ілюстративного матеріалу	28.04.2025	11.05.2025	
10.	Попередній захист, доопрацювання, рецензування	13.05.2025	13.05.2025	

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні і ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БКР

8.1 При оформлювання БКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки, структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання, загальні положення та правила складання»;

— міждержавний ГОСТ 2.104— 2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 — «Комп'ютерна інженерія» (освітня програма «Системне програмування»), кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ— 03.02.02— П.001.01:21».

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмне забезпечення для тривимірної візуалізації фракталів

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,5 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Азаров О.Д., зав. каф. ОТ _____ _____
(прізвище, ініціали, посада) (підпис)

Тарновський М.Г., доц. каф. ОТ _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Черняк О.І.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Бурдейний О.В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

ГРАФІЧНА ЧАСТИНА

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ТРИВИМІРНОЇ ВІЗУАЛІЗАЦІЇ ФРАКТАЛІВ

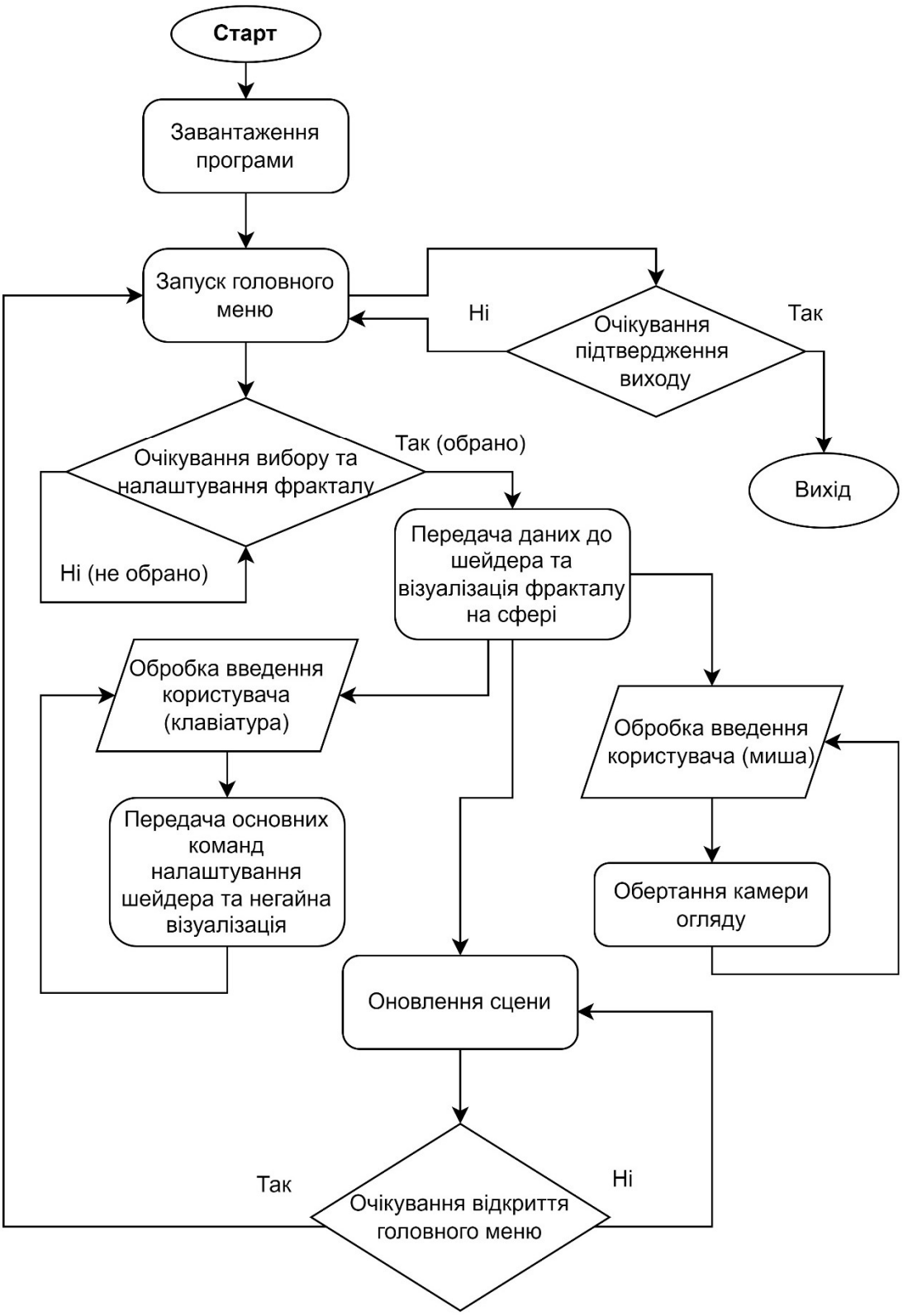


Рисунок В.1 — Діаграма алгоритму роботи програмного забезпечення

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ТРИВИМІРНОЇ ВІЗУАЛІЗАЦІЇ ФРАКТАЛІВ

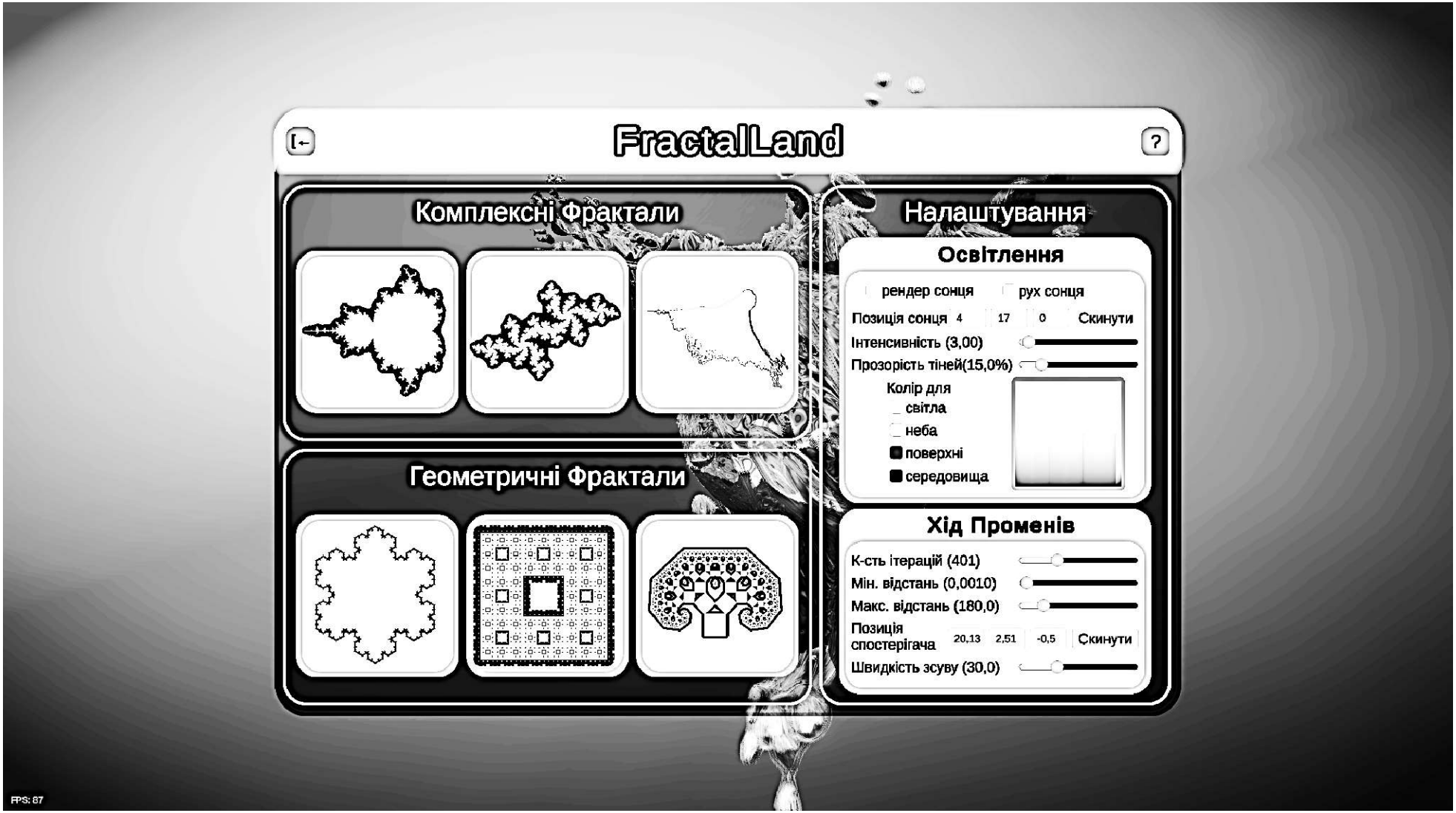


Рисунок Г.1 — Вигляд головного меню розробленого додатка

ДОДАТОК Д

Лістинг коду SDF_Core

```
float Box(float3 p, float3 b)
{
   float3 q = abs(p) — b;
   return length(max(q, 0.0)) + min(max(q.x, max(q.y, q.z)), 0.0);
}
float Cross(float3 p, float b)
{
   // float w = b/3.;
   float x = Box(float3(p.xy, 0), b);
   float y = Box(float3(p.yz, 0), b);
   float z = Box(float3(p.zx, 0), b);
     return min(x, min(y, z));
}
float Transform(float3 p, float3 position, float3 scale)
{
   p — = position; // Translate to the object's position
   p /= scale; // Scale the position
   return p;
}
float RoundBox(float3 p, float3 b, float r)
{
   float3 q = abs(p) — b + r;
   return length(max(q, 0.0)) + min(max(q.x, max(q.y, q.z)), 0.0) — r;
}
float HexagonalPrism(float3 p, float2 h)
{
   const float3 k = float3(— .8660254, .5, .57735);
   p = abs(p);
   p.xy — = 2. * min(dot(k.xy, p.xy), 0.) * k.xy;
   float2 d = float2(
                  length(p.xy —
                     float2(clamp(p.x, — k.z * h.x, k.z * h.x), h.x))
                     * sign(p.y — h.x),
                  p.z — h.y);

   return min(max(d.x, d.y), 0.0) + length(max(d, 0.0));
}
```